Др Слободан Обрадовић

ОСНОВИ
РАЧУНАРСКЕ ТЕХНИКЕ

Висока школа електротехнике и рачунарства струковних студија

Београд, 2014.

Аутор: др Слободан Обрадовић, проф.

Рецезенти: др Александар Жорић, проф. Универзитета К. Митровици
др Ристо Бојовић, проф. Универзитета у К. Митровици

Издавач: Висока школа електротехнике и рачунарства струковних студија, Београд

Лектор: Весна Милосављевић

Штампа: Школски сервис Гајић, Београд

Тираж: 50

CIP - Каталогизација у публикацији
Народна библиотека Србије, Београд

004 (075 . 8)

ОБРАДОВИЋ, Слободан, 1955-
Основи рачунарске технике / Слободан Обрадовић. - 9. неизмењено изд. - Београд : Висока школа електротехнике и рачунарства струковних студија, 2014 (Београд : Школски сервис Гајић). - IV, 428 стр. : илустр. ; 24 cm

Тираж 50. - Опис појмова: стр. 411-428. - Библиографија: стр. 372.

ISBN 978-86-85081-27-9

а) Рачунарство
COBISS.SR-ID 210940940

ПРЕДГОВОР

Уџбеник **Основе рачунарске технике** је првенствено намењен студентима виших школа, а настао је на бази предавања која је аутор, на Вишој електротехничкој школи у Београду, држао у периоду 1992–2002. студентима прве године из предмета "Основи рачунарске технике и програмирања".

При одређивању садржаја и обима излагања аутор је следио програм предмета "Основе рачунарске технике и програмирања", али је обим неких поглавља превазишао потребе овог основног курса због тога што се предмети који ове области детаљно обрађују не слушају на свим смеровима Више електротехничке школе. На тај начин пружена је могућност студентима да се, по сопственом нахођењу, детаљније упознају са овим областима.

Сама предавања и концепт овог уџбеника урађени су по узору на одговарајућу уџбеничку литературу [5], [8], [10], [12], [18], [22], из које је преузет и један део примера и објашњења, обзиром да су аутори познати стручњаци и педагози. Други део примера настао је као резултат сопствених искустава аутора током предавања студентима. Део другог поглавља написао је мр Милан Мијалковић и тиме допринео да књига додатно добије на квалитету.

Аутор се захваљује рецензентима др Александру Жорићу и др Ристу Бојовићу, као и колеги мр Милану Мијалковићу на низу корисних сугестија које су допринеле да делови овог текста буду јаснији.

2013, Београд АУТОР

УВОД

Од свог настанка човек се трудио да себи олакша обављање разних послова тако што је у почетку користио разне алатке, а касније све сложеније машине. Првих неколико хиљада година то су углавном биле механичке машине које су олакшавале обављање разних физичких послова, док се последњих неколико стотина година човек све више ослобађа и обављања разних умних послова. Рачунар је оруђе, алатка или машина која повећава снагу и могућности људског ума у обављању рутинских ителектуалних операција.

На садашњем степену развоја рачунари су врло ефикасни у замени људског умног рада у разним рачунским, научно–техничким и логичким операцијама и масовним обрадама података, а при томе је знатно смањена могућност настанка грешке. Сваки умни рад који се може разложити на коначан број елементарних операција, и изразити у облику неког поступка обраде (алгоритам), може се преточити у програм, и предати рачунарском систему на извршавање. Рачунарски системи нису у могућности да замене човека у креативним пословима.

У првој глави: **Увод у рачунарске системе** дат је кратак опис рада и састава рачунарских система које чине: разни технички уређаји (хардвер), програми (софтвер), подаци, процедуре и људи. Хардвер је тако конструисан да под контролом корисничких и системских програма обавља: прихватање, смештање, чување и поновни приступ подацима, обраду података и саопштавање добијених резултата. За сваку од ових операција у рачунарском систему задужени су посебни уређаји, мада неки уређаји могу обављати и више поменутих функција. Прихватање података обављају улазне јединице, смештање и чување података обављају разне врсте меморија, обраду података обавља централна јединица (рачунар), а приказивање резултата обавља се посредством излазних јединица.

Информације, тј. подаци и инструкције, у рачунару су смештене и обрађују се у облику бинарних бројева. У другој глави: **Математичке основе рачунара** дат је кратак опис разних врста бројних система, уз детаљан опис бинарног бројног система и начина кодирања нумеричких и ненумеричких података.

У трећој глави: **Електронске основе рачунара** описане су основне логичке операције и логичка кола која чине основу функционисања рачунара. У наставку су описана основна аритметичка кола и основна меморијска кола: флип–флоп, регистри и разне врсте меморија, као и на које се ова кола међу собом повезују (магистрале).

Машинске инструкције се изводе у две фазе: прибављање и извршавање. Извршавање појединачних инструкција и програма у рачунару обавља се аутоматски, без интервенције човека. Аутоматско извршење програма омогућено је на тај начин што процесор у сваком тренутку, док извршава једну наредбу, зна која је следећа наредба коју треба извршити. То се постиже уз помоћ једног регистра у процесору који се зове програмски бројач. Да би се отпочело извршавање програма треба само саопштити процесору која је прва наредба програма коју треба да изврши. У четвртој глави: **Архитектура рачунара**, поред описа фаза извођења машинских инструкција, дати су опис разних начина

реализације управљачких јединица и разних начина обраде података, као што су паралелна обрада, мултипрограмирање и мултипроцесирање.

Шта ће се и у ком тренутку радити у рачунару, и са којим подацима, одређује низ инструкција (које одговарају неким елементарним обрадама), а које називамо програмом. У петој глави: **Врсте наредби и начини адресирања** описане су разне врсте наредби и разни начини адресирања који се користе у различитим рачунарским системима, са посебним освртом на микрорачунаре, с обзиром на њихову велику распрострањеност и доступност.

Иако је рачунар централни део рачунарског система, у састав система улази и читав низ других уређаја које називамо периферијски уређаји. Ови уређаји врше унос, смештање (меморисање) података и резултата обраде, као и издавање резултата. Ови уређаји, такође, врше и неке елементарне обраде података (кодирање, контрола парности, итд). Они раде под контролом централног процесора, али имају истовремено и велику аутономију. Кратак опис начина рада ових уређаја дат је у шестој глави – **Периферијске јединице**.

Основни предмет обраде у рачунарским системима су информације кодиране у облику бинарних бројева који чине инструкције и податке. Подаци се у рачунару појединачно обрађују у централном процесору, али се у меморији никада не чувају у изолованом облику, већ организовано, у облику разних структура података, на начин како је то описано у седмој глави – **Структуре података**.

Програми који се извршавају, као и подаци који се обрађују морају бити смештени у главну меморију рачунара, јер су само тада доступни централном процесору. Сваки програм, да би могао да се изведе, мора да буде написан у складу са правилима (синтакса програмског језика), али и преведен у форму коју машина разуме (машински језик). Основни појмови везани за ову проблематику дати су осмој глави – **Програмски језици и језички преводиоци**.

Један од најважнијих делова рачунарског система, који омогућава додатно ослобађање човека од рутинских послова, у области употребе рачунарске опреме и уређаја, у обради, организовању и смештању података и програмирању и припреми програма за извршавање, чине системски програми, а међу њима су од посебног значаја оперативни системи, који су описани у деветој глави – **Оперативни системи**. Оперативни системи управљају радом свих делова рачунара и рачунарског система. Они појединим програмима додељују централни процесор, оперативну меморију и периферијске уређаје. На тај начин они омогућавају ефикасно коришћење рачунарског система, уз истовремено извршавање више програма (вишепрограмски режим рада), и једновремено коришћење рачунара од стране више корисника (вишекорисnички режим рада).

У десетој глави: **Развој рачунарске технике** дат је кратак преглед развоја рачунара и технологија за израду електронских уређаја, као и њихов утицај на ширење примене рачунара.

У почетку ере електронских рачунара, рачунари су се користили углавном у разним научно–техничким делатностима, а рад на рачунару је био привилегија малог броја врхунских стручњака – програмера. Области примене се непрекидно шире, па су данас рачунари, захваљујући пре свега примени микропроцесора, ушли у све области људских делатности и практично су постали део свакодневног живота свих људи.

САДРЖАЈ

страна

1. Увод у рачунарске системе 1

1.1 Општи модел рачунарског система 1
1.2 Функционална блок шема рачунара 4
1.2.1 Ручна обрада података 4
1.2.2 Аутоматизована обрада података помоћу рачунара 5
1.3 Хијерархијски модел рачунарског система 9
1.4 Закључак 11
1.5 Питања 12
1.6 Кључне речи 12

2. Математичке основе рачунара 13

2.1 Бројни системи 13

2.2 Конверзија бројева из једног бројног система у други 16
2.2.1 Конверзија декадних бројева у друге бројне системе 16
2.2.2 Конверзија бин, оцт и хеџ бројева у друге бројне системе 18

2.3 Појам комплемента 19
2.3.1 Комплементи бинарног броја 21

2.4 Бинарни бројни систем 22
2.4.1 Бинарно сабирање 23
2.5 Бинарни бројеви са знаком, кодирање негативних бројева 23
2.5.1 Кодирање негативних бројева - други комплемент 26
2.5.2 Представљање негативних бројева другим техникама 30
2.5.3 Анализа резултата сабирања - заставице 33
2.5.4 Бројеви са непокретном запетом 35
2.5.5 Представљање бројева са покретном тачком 44
2.5.6 Бинарно кодирани декадни бројеви 46

2.6 Кодирање ненумеричких података 50
2.7 Закључак 51
2.8 Питања 51
2.9 Кључне речи 52

3. Електронске основе рачунара 53

3.1 Дигитална логика 53
3.1.1 Булова алгебра 54
3.1.2 Основне логичке операције над бинарним цифрама 55
3.1.3 Елементарна логичка кола 58
3.1.4 Минимизација логичких функција 62

3.2 Елементарна меморијска кола 69
3.2.1 Флип-флоп 70

3.3 Елементарна аритметичка кола 72
3.4 Регистри 75
3.5 Магистрале 78
3.6 Меморије 84
3.6.1 Главна меморија рачунара 89
3.6.2 Различити типови организација меморије 92
3.6.3 Меморија **ROM** 98
3.6.4 Меморија са више модула и преклапање 100
3.7 Закључак 100
3.8 Питања 101
3.9 Кључне речи 102

4. Архитектура рачунара 103

4.1 Поједностављена архитектура рачунара 103
4.1.1 Улаз података 105
4.1.2 Приступ меморији 107
4.1.3 Излаз података 110
4.1.4 Прибављање инструкција 112

4.2 Инструкције у машинском језику 114
4.2.1 Фаза прибављања машинских инструкција 117
4.2.2 Фаза извршавања машинских инструкција 122

4.3 Микропрограмска организација управљачких јединица 133
4.3.1 Фаза прибављања микроинструкција 135
4.3.2 Фаза извршења микроинструкција 137

4.4 Хардверска организација управљачких јединица 138
4.5 Архитектура текуће траке и паралелна обрада 139
4.6 Мултипроцесорски системи 141
4.7 Закључак 145
4.8 Питања 146
4.9 Кључне речи 146

5 Врсте наредби и начини адресирања 147

5.1 Регистри 148

5.2 Аритметичко-логичка јединица 155

5.3 Типови машинских инструкција 156
5.3.1 Аритметичке и логичке инструкције 156
5.3.2 Инструкције за пренос података 165
5.3.3 Улазно-излазне операције 167
5.3.4 Инструкције за управљање током програма 167

5.4 Начини адресирања 177
5.4.1 Начини адресирања меморијских локација 178
5.4.2 Адресирање улазно-излазног простора 190

5.5 Закључак 191
5.6 Питања 191
5.7 Кључне речи 192

6 Периферијске јединице 193

6.1 Начини преноса улазно-излазних података 193
6.1.1 Програмирани У/И преноси 194
6.1.2 Директан приступ меморији 197
6.1.3 Прекидни У/И пренос и периферијски процесори 198
6.1.4 Стандарди за периферијске међусклопове 201

6.2 Уређаји за унос и издавање података 204
6.3 Секундарне меморије 215
6.3.1 Организационе јединице података 216

6.4 Секундарни меморијски уређаји 219
6.4.1 Јединице са секвенцијалним приступом 219
6.4.2 Јединице са директним приступом 226

6.5 Закључак 234
6.6 Питања 235
6.7 Кључне речи 236

7 Структуре података 237

7.1 Скаларни подаци 238
7.2 Једноставни нескаларни подаци 241
7.3 Структуре података 242
7.4 Сложене структуре података 244
7.4.1 Стабла 245
7.4.2 Магацини и редови 248

7.5 Класе меморија 252
7.6 Приступ меморији 257
7.7 Закључак 266
7.8 Питања 267
7.9 Кључне речи 267

8. Програмски језици и језички процесори 269

8.1 О системском софтверу 269
8.2 Програмски језици 271
8.2.1 Машински зависни језици 272
8.2.2 Машински независни језици 275
8.2.3 Синтакса језика 277

8.3 Језички процесори 279
8.3.1 Асемблери 281
8.3.2 Компилатори 284
8.3.3 Интерпретатори 285
8.3.4 Повезивачи и пуниоци 286

8.4 Едитори 289
8.5 Закључак 290
8.6 Питања 291
8.7 Кључне речи 292

9. Оперативни системи 293

9.1 Дефиниције и модел 293
9.1.1 Језгро 296
9.1.2 Састав оперативног система 298

9.2 Процеси и промене стања 299
9.3 Додељивање процесора 308
9.4 Механизам прекида 310
9.4.1 Врсте прекида 312
9.4.2 Обрада прекида 313

9.5 Распоређивање 318

9.6 Управљање меморијом 319
9.6.1 Једнокориснички мониторски системи 321
9.6.2 Додељивање меморије у партицијама 323
9.6.3 Техника преклапања 325
9.6.4 Систем виртуелних меморија 326
9.6.5 Страничење и сегментација 330

9.7 Управљање улазом и излазом 333
9.8 Управљање датотекама 335
9.8.1 Директоријуми 336
9.8.2 Операције са датотекама 341
9.8.3 Права приступа датотекама и директоријумима 344
9.8.4 Смештање датотека и директоријума 346

9.9 Закључак 351
9.10 Питања 351
9.11 Кључне речи 352

10. Развој рачунарске технике 353

10.1 Технолошке ере 353

10.2 Електронски цифарски рачунари 358
10.2.1 Рачунари прве генерације 358
10.2.2 Рачунари друге генерације 360
10.2.3 Трећа генерација рачунара 362

10.3 Микропроцесори и микрорачунари 366
10.4 Закључак 370
10.5 Питања 370
10.6 Кључне речи 371

Литература 372

Прилог А**:** Табеле **ASCII и EBCDIC** кодова 373

Прилог Б: Врсте рачунарских система 375

Б-1 **Симплец** рачунарски системи 376

Б-2 Мултипроцесорски системи 376

Б-3 Дистрибуирана обрада 378

Б-3.1 Остваривање везе 378

Б-3.2 Топологија дистрибуираних система 386

Б-3.3 Управљање дистрибуираним системима 390

Б-3.4 ЛАН мреже 399

ETHERNET 400

TOKEN RING 401

Б-3.5 ЊАН мреже 403

Интернет 405

Б-4 Закључак 409

Б-5 Кључне речи 410

Прилог Ц**:** Опис појмова 385

1. УВОД У РАЧУНАРСКЕ СИСТЕМЕ

Кроз читаву своју историју, људи су били принуђени да врше различита израчунавања и обраду информација добијених из света који их окружује. Обим и сложеност ових израчунавања непрекидно су се повећавали, а мануелно израчунавање, у којем је човек основно средство, има два велика ограничења у обављању ових послова:

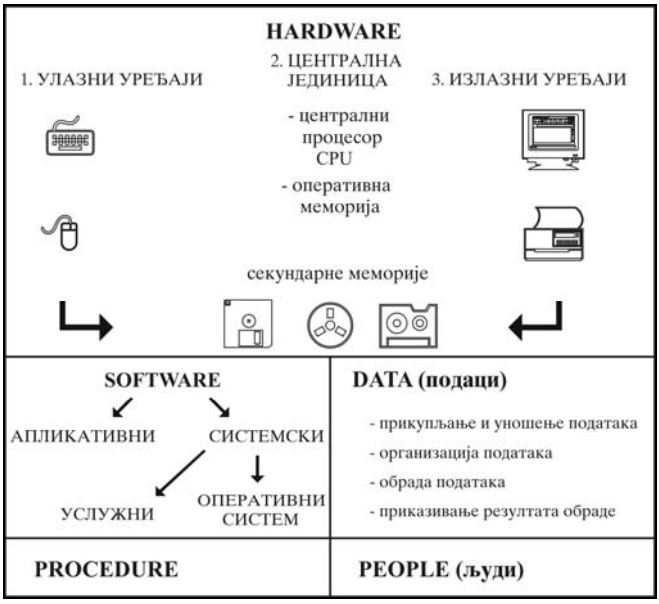
1. *Човекова брзина је врло ограничена. За обављање елементарних операција сабирања или множења, човеку је потребно од неколико секунди до неколико минута.*
2. *Човек показује склоност ка прављењу грешака, тако да су резултати врло сложених израчунавања и обраде, које обавља човек, често непоуздани. За разлику од човека, машине су потпуно имуне на низ људских грешака у процесу обраде информација, које су последица: растројености, забринутости, умора и сл.*

У процесу израчунавања људи користе разна помоћна средства, са циљем да себи олакшају посао и да вишеструко убрзају поступке обраде, да добију тачне резултате и да донесу правилне одлуке. Та средства су се непрекидно развијала и усавршавала да би, средином двадесетог века, довела до појаве аутоматских електронских цифарских рачунских машина, или скраћено, рачунара. *Рачунар је уређај који самостално обавља обраду података извршавајући дигиталне логичке операције на основу унесеног програма.*

Рачунарски систем представља скуп свих средстава која користимо у процесу решавања једног или групе задатака.

1.1. ОПШТИ МОДЕЛ РАЧУНАРСКОГ СИСТЕМА

Један од најопштијих модела рачунарског система је онај који је дао **David Kroenke**, у којем се рачунарски систем састоји од пет компоненти: техничког дела (**hardware,** хардвер), програмског дела (**software,** софтвер), података (**data**), процедура (**procedures**) и људи (**people**), слика 1.1.



Слика 1.1 Општи модел рачунарског система

Хардвер рачунарског система чини технички део, а састоји се од скупа различитих уређаја, од којих сваки обавља више једноставних операција и обрада, које се могу груписати у неке основне функције. На основу основних функција које обављају, технички део рачунарског система чине: улазни уређаји, уређаји за обраду, тј. рачунар у ужем смислу, или централна јединица и излазни уређаји.

Улазни уређаји омогућавају приступ рачунарском систему. Посредством ових уређаја корисници задају команде, траже информације, уносе нове податке или програме у рачунар. Рачунар прихвата улазе врло различитог типа, на пример притисак на дугме или тастер, додир прста на екран, оптичко сканирање, изговарање речи, итд.

Уређај за обраду, централна јединица или рачунар, садржи електронске компоненте које на основу уграђеног или унетог програма (низа инструкција) извршавају различите аритметичке или логичке операције, модификују текст, доносе различите одлуке, прихватају улазне податке и генеришу излазне податке.

Када је обрада података извршена, резултат мора бити доступан кориснику. Ову функцију обављају излазни уређаји, а у њих спадају видео дисплеји (монитори, екрани), штампачи (printers), плотери, звучници итд.

Посебну групу уређаја, који истовремено обављају функције улаза и излаза, као и функцију дуготрајног памћења и чувања података и програма, чине спољне меморије у које спадају разне врсте дискова, трака итд.

Да би технички део рачунарског система (хардвер) остварио своје функције обраде, неопходан му је низ инструкција, тј. елементарних операција. Инструкције се групишу тако да заједно омогућавају решавање одређеног задатка, или обављање неке сложене операције и обраде, и тада чине

програме. Скуп свих програма представља софтверски део рачунарског система. Програми могу да садрже низ детаљних инструкција у облику низова цифара **0** и **1** (у тзв. машинском језику), које рачунар може да користи непосредно. Програми могу бити написани и на неком од програмских језика као што су **Assembler, COBOL**, **Pascal, BASIC** итд., који су људима лакши за разумевање, али их рачунарски систем, пре него што их рачунар изврши, мора превести на машински језик. Програми који улазе у састав рачунарског система сврставају се у две групе:

1. *Системски програми који управљају радом и омогућавају коришћење техничког дела рачунарског система с једне стране, и корисницима олакшавају израду сопствених програма, комуницирање са појединим уређајима и омогућавају њихово лако и ефикасно коришћење. Системски програми су посредници између корисника и његовог програма с једне стране, и хардвера рачунарског система с друге стране.*

2. *Кориснички програми решавају неки конкретан задатак или обраду. Они кориснику дају неке конкретне резултате, на основу којих он предузима одређене активности. У току свог извођења, кориснички програми нужно користе различите функције из групе системских програма. Ови програми се називају још и апликативни програми или проблемски програми.*

Подаци су трећи део модела. Подаци могу имати облик имена, бројева и адреса. Они могу представљати меру најразличитијих физичких величина: од нивоа воде у посуди, до спектра светлости звезда. Подаци су основни објекат обраде у рачунарском систему, односно предмет рада и уједно главни разлог коришћења рачунара. Без рачунара наше могућности да црпемо знања из података биле би значајно смањене.

Подаци су дискретни, записани факти о појавама и догађајима из света који нас окружује, из којих добијамо информације о свету. Другим речима, подаци су чињенице, ознаке или запажања настала у току неког процеса, а која су записана, односно кодирана помоћу неких физичких симбола, или симбола неке азбуке, и имају својство да могу да се бележе, чувају, преносе и обрађују. Подаци су средства за изражавање и добијање информација, и они представљају изоловане и неинтерпретиране чињенице. Податке прикупљамо и записујемо, да би их могли чувати, и по потреби користити. Податак постаје информација у моменту његовог коришћења. Обрада података може се поделити у четири фазе:

1. *прикупљање и унос података,*
2. *организација података и њихово чување, тј. складиштење,*
3. *обрада података помоћу рачунара, тј. обрада у ужем смислу (рачунање, сортирање, груписање итд.),*
4. *издавање података и резултата обраде (штампање, цртање, итд.).*

У неким применама рачунара не морају постојати све четири поменуте фазе обраде. Тако, рецимо, у многим научно–техничким обрадама нема уопште улазних података, већ се подаци израчунавају у рачунару као, рецимо, при израчунавању логаритамских таблица, случајних бројева итд.

Процедуре представљају скуп поступака и правила, која се морају поштовати и примењивати у циљу правилне употребе рачунарског система, и правилне обраде података, да би корисници, тј. људи, помоћу рачунара,

решили неки задатак. Процедуре представљају инструкције упућене људима, које кораке треба извршити, како и које податке обрадити, и које излазне величине треба добити.

Пету компоненту рачунарског система чине људи. Људи се придржавају одређених процедура приликом коришћења рачунара. Они обављају активности које стварају или прикупљају податке. Људи развијају програме којима решавају нове поступке обраде и мењају постојеће програме, да би се задовољили новонастали захтеви. Људи за друге људе креирају процедуре, којих се они у процесу коришћења рачунара придржавају.

1.2. ФУНКЦИОНАЛНА БЛОК-ШЕМА РАЧУНАРА

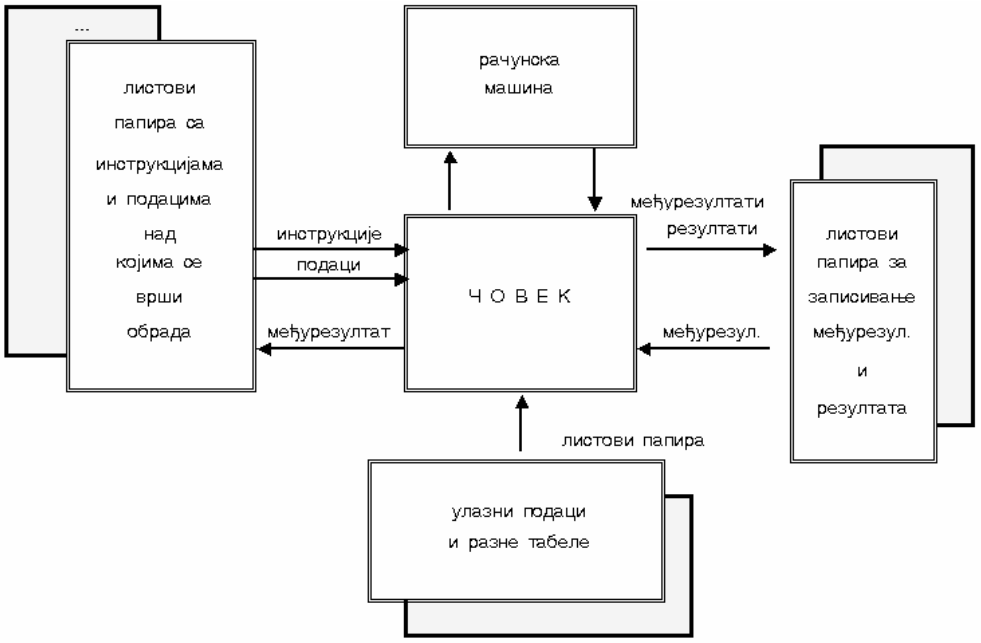
Пре описа рада рачунара размотримо најпре како човек решава неки задатак уз помоћ најједноставније стоне рачунске машине која обавља само четири основне рачунске операције.

1.2.1. РУЧНА ОБРАДА ПОДАТАКА

У ручној обради података (слика 1.2.), поступак рачунања, тј. решавање било ког проблема, састоји се од низа појединачних корака од којих сваки садржи по једну од елементарних операција (+, –, * , /). Сваки појединачни корак прописује на који начин и над којим подацима треба извршити израчунавање, да би се на крају добио тачан резултат, односно урадио постављен задатак. Човек сваки задатак обраде, ма колико био прост или сложен, најчешће формализује у облику колона табеле, коју записује на једном или више листова папира. У сваку врсту табеле човек записује по једну елементарну операцију и податке над којима се она тренутно извршава. Свака врста ове табеле, тј. елементарна операција, представља **елементарну инструкцију**, или краће речено инструкцију.

Одређени низ инструкција који решава постављени задатак, а након чијег извршавања се добија тражено решење, зове се **програм** обраде. У програму, елементарне обраде, тј. инструкције, најчешће су поређане оним редоследом којим се извршавају. То значи да је редослед извршавања елементарних операција строго дефинисан редоследом инструкција у програму обраде, и свакој од ових инструкција придружује се редни број. По правилу се нпр. после пете инструкције извршава шеста, а после шесте извршава се седма инструкција, и тако редом. Од овог природног редоследа може се одступити када се стекну одређени услови, тј. зависно од конкретних вредности почетних података или добијених резултата.

На пример, зависно од тога какве су вредности коефицијента $\mathbf{a}$ и дискриминанте $\mathbf{D}$ ($\mathbf{D=b^2-4ac}$), поступци за израчунавање реалних корена $\mathbf{x_1}$ и $\mathbf{x_2}$ квадратне једначине ($\mathbf{ax^2 + bx + c = 0}$) су различити, и израчунавају се по различитим формулама. Наиме, ако је $\mathbf{a = 0}$ једначина није квадратна, ако је $\mathbf{D = 0}$ онда су корени једнаки $\mathbf{x_1 = x_2 = - b / (2 a)}$, а ако је $\mathbf{D < 0}$, једначина нема реалних корена, већ су корени конјуговано комплексни, па се реални и имагинарни део морају приказати као посебни делови једног сложеног податка.



Слика 1.2 Ручна обрада података помоћу рачунске машине

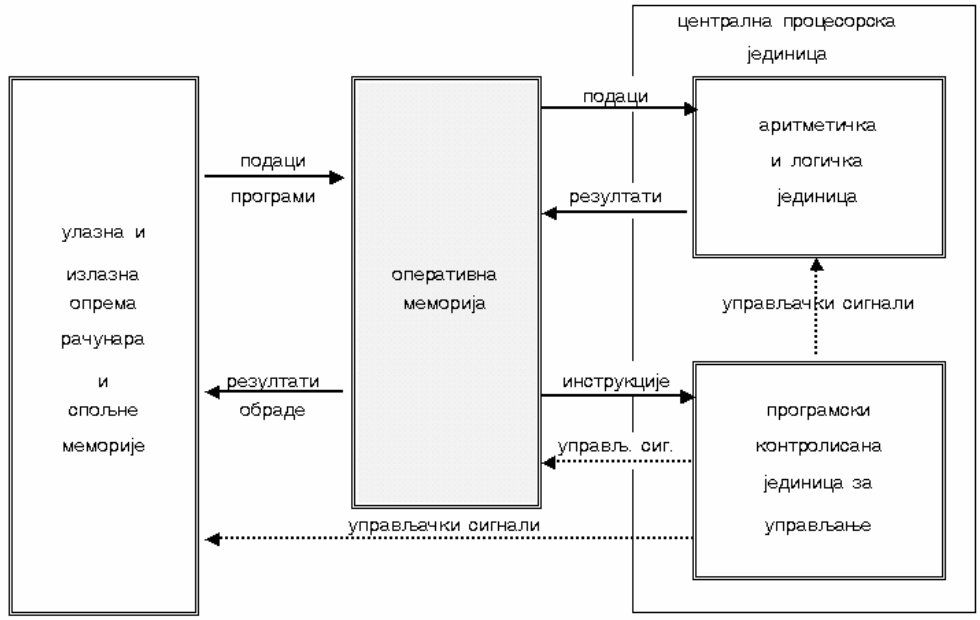
Подаци над којима се врше елементарне операције обично се прикупљају пре самог процеса обраде (у краћем или дужем периоду времена), и налазе се записани (меморисани) на посебним листовима папира. Врло често се, у циљу добијања различитих информација над једним истим подацима врше различите обраде. У појединим корацима обраде учествују само неки подаци, па се само ти подаци привремено преписују на листове папира, или придружују инструкцијама у табелама које садрже резултате обраде.

Човек који врши израчунавање, често у циљу решавања свог проблема, тј. задатка, користи општепознате величине, као и разне математичке, статистичке и друге табеле (логаритме, тригонометријске функције итд.).

У току обраде формирају се међурезултати (које човек привремено памти, или записује), а на крају се добијају коначни резултати обраде које човек записује на папиру. Међурезултати и коначни резултати једног поступка обраде, могу бити почетни подаци у неком другом поступку обраде.

1.2.2. АУТОМАТИЗОВАНА ОБРАДА ПОДАТАКА ПОМОЋУ РАЧУНАРА

Структура електронског система за аутоматизовану обраду података помоћу рачунара (слика 1.3) је врло слична структури при ручној обради података (која је приказана на слици 1.2), али је састав другачији.



Слика 1.3 Главне компоненте електронског уређаја за обраду података

Уместо рачунске машине, у аутоматизованој обради појављује се елемент за рачунање, тј. **аритметичко-логичка јединица** рачунара (**arithmetic and logic unit, ALU**).

Функцију папира, који је човеку служио како за формулисање задатака обраде, тако и за уписивање табела са почетним подацима, међурезултатима и финалним резултатима обраде, преузимају сада разне врсте меморија. Човека сада замењује програмски контролисана управљачка јединица (**control unit, CU**), односно командна или контролно–управљачка јединица. Како је човек на овај начин, у поступку обраде података, знатно растерећен од извршавања низа рутинских послова обраде, онда он може да се посвети искључиво формулисању задатака и поступака обраде, тј. изради програма.

Унутрашња, односно оперативна меморија је основна или главна меморија рачунара (**main memory, operating memory, primary memory**). Она садржи податке над којима се врши обрада, међурезултате, као и сам начин на који се врши обрада, тј. задатак који треба обавити. Начин како се врши обрада садржан је у програму који је писао човек, и то у облику низа појединачних инструкција, односно елементарних операција.

Управљачка јединица узима из унутрашње меморије једну инструкцију за другом, интерпретира их и извршава непосредно, или уз помоћ аритметичко–логичке јединице.

Аритметичко–логичка јединица најчешће извршава само основне рачунске операције и радње, и не може да извршава сложеније рачунске операције. Контролна јединица, заједно са рачунском јединицом, чини јединицу за обраду, тј. централну јединицу за обраду (**central processing unit, CPU**), или краће **централни процесор** рачунара.

Централни процесор заједно са унутрашњом меморијом чини централну јединицу рачунарског система, односно **рачунар** у ужем смислу. Данас су рачунари реализовани у облику електронских, дигиталних, логичких склопова у полупроводничкој технологији са великом брзином рада.

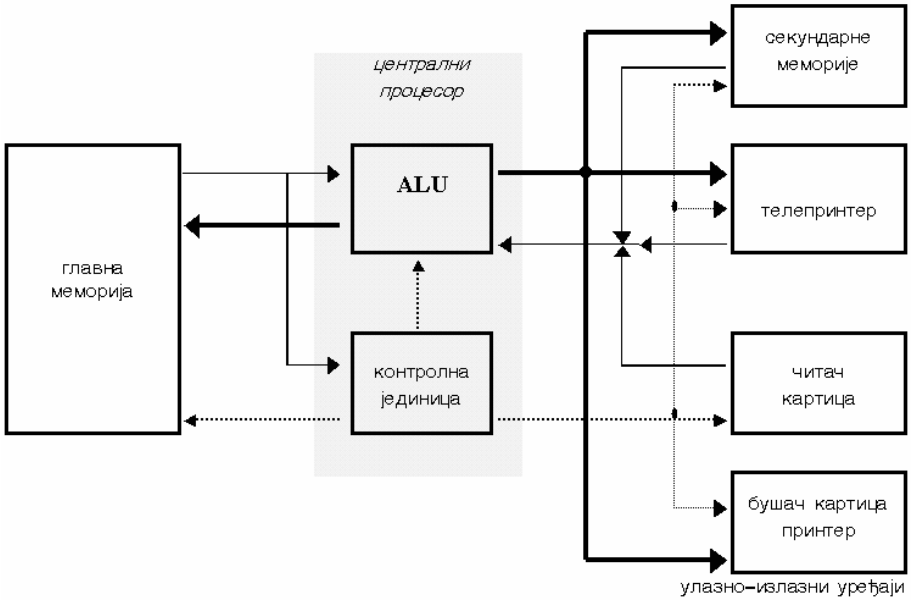
Спољашње, масовне меморије (**external memory, secondary storage, mass storage**) служе за улаз, излаз и трајно чување велике количине података, међурезултата и резултата обраде. Ове меморије су реализоване на различитим медијима (носиоци информација), који имају својство трајног памћења, као што су: магнетне траке и дискови, оптички дискови, итд.

Ови носиоци информација омогућавају уједно да се подаци и резултати обраде на једном рачунарском систему, пренесу на други уређај за аутоматску обраду података ради даље обраде, без успостављања физичке везе између ових уређаја.

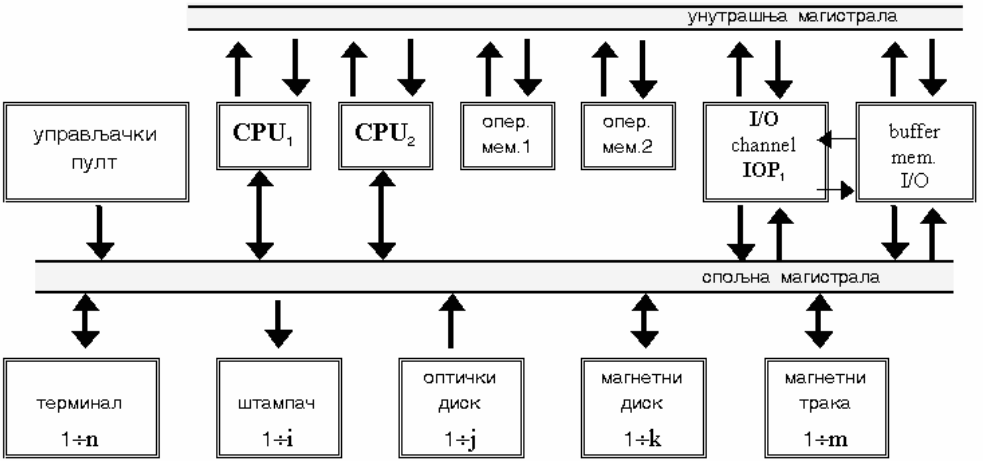
Рачунари су у свом развоју од настанка до данашњих дана прошли кроз неколико фаза развоја које називамо генерацијама. Први рачунари имали су архитектуру која је слична оној са слике 1.3., и која се поједностављено може приказати као на слици 1.4. Може се рећи да је архитектура рачунара прве генерације готово идентична структури ручне обраде података, јер се пренос података обављао не само уз контролу, већ и уз учешће централног процесора. На сликама 1.3. и 1.4. приказана је архитектура у којој су улазно–излазни уређаји директно повезани са централним процесором или са главном меморијом рачунара. У највећем броју савремених рачунара ово комуницирање обавља се преко одговарајућих уређаја, канала (**input-output processors, IOP, channels**), и уз посредовање прихватних меморија за спрегу, међумеморија (**buffer memory**), као што је приказано на слици 1.5.

Уочавамо да у оквиру једног рачунарског система можемо имати више централних процесора који паралелно раде, а омогућавају знатно убрзање рада рачунара и паралелно извршавање једног или више програма. За контролу преноса података између улазних јединица, излазних јединица и јединица масовних меморија, које се једним именом називају периферијске јединице, с једне стране, и централних процесора и унутрашње меморије с друге стране, најчешће се користе улазно–излазни канали и процесори. Ови уређаји често садрже аутономне меморије за спрегу (**buffer memory**), чиме се остварује додатна аутономност рада уређаја. Непосредну контролу ових уређаја врше централни процесори, а уређаји самостално контролишу пренос података, и уједно врше прилагођење (по брзини) између врло брзих

процесора и унутрашње меморије, и врло спорих механичких склопова и спољних меморија.



Слика 1.4. Архитектура рачунара прве генерације



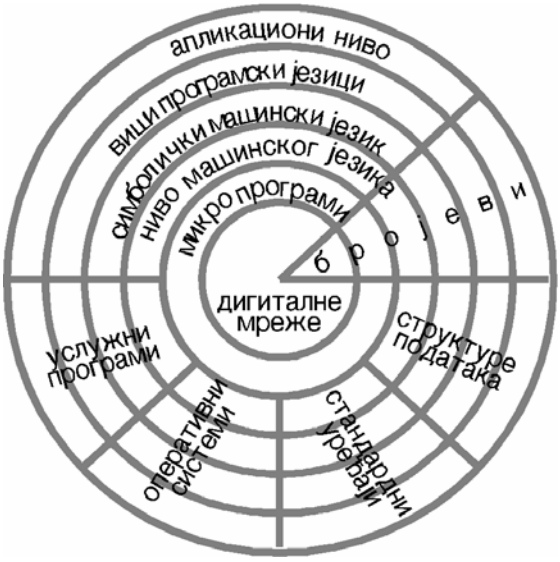
Слика 1.5. Структура рачунарског система са улазно–излазним каналима

Један канал у једном интервалу времена може опслуживати више периферијских уређаја, такозвани мултиплексорски канал, или само једну релативно брзу спољну јединицу (магнетни диск, оптички диск, ...), такозвани селекторски канал.

1.3. ХИЈЕРАРХИЈСКИ МОДЕЛ РАЧУНАРСКОГ СИСТЕМА

Многи аутори под појмом рачунарског система подразумевају углавном прве две компоненте општег модела, односно хардвер (технички део), и софтвер (програмски део у чији састав улазе и подаци), што чини рачунарски систем у ужем смислу.

Овакав рачунарски систем може се приказати хијерархијским моделом у облику концентричних кругова (слика 1.6).



Слика 1.6. Хијерархијски модел рачунарског система

У центру овог модела су дигитална логичка кола (**digital logic**) из састава рачунарског хардвера. Овај прстен одређује основне аритметичке и логичке операције рачунара. Овом прстену (језгру) припадају и основна меморијска кола. Свака компонента ових логичких мрежа извршава по једну врло једноставну операцију. Језгро може садржати десетине, или чак стотине операција, које употпуњавају нешто што људи у рачунској техници сматрају скупом елементарних обрада.

У многим модерним рачунарима, рад ових елементарних логичких мрежа координира се помоћу низа програма који се називају **микропрограми** (**microprogram, microcode**).

Микропрограми садрже низ инструкција познатих као микрооперације, које координирају рад логичких кола, и стварају могућност да људима познате, једноставне инструкције, попут сабирања или одузимања, буду реализоване као низови мноштва још простијих операција.

Трећи ниво чини машински језик (**conventional machine level**) и он представља најнижи ниво заједнички за све програме, јер централни процесор разуме само ове инструкције, и способан је да их интерпретира и изврши. Свака инструкција која нам је на располагању на овом нивоу у стварности је подржана – реализована или као један микропрограмом или као једна логичка мрежа.

Следећа два нивоа су симболички машински језик (асемблер, **assembly language**) и виши програмски језици (**high-level languange**) који чине програмску спрегу човека са рачунаром. Асемблерски језик је ослоњен на микропрограмски ниво и дигиталне логичке мреже. Писање програма на овом језику је знатно лакше него на машинском, али такође захтева велико познавање архитектуре и организације централног процесора и рачунара. Виши програмски језици (**BASIC, Pascal, FORTRAN**) су људима знатно разумљивији, траже врло мало познавање рачунара, а такође не зависе од типа рачунара.

Најопштији ниво је кориснички (**applications**), који чине људи који чак не морају бити ни програмери већ су само крајњи корисници. Они рачунар посматрају кроз одређену групу програма, посебно пројектованих за решавање специфичних проблема, тако да рачунарски ресурси оптимално служе кориснику.

Бројеви (**numbers**) повезују све нивое хијерархијског модела, односно саставни су део свих нивоа. Дигитална логичка кола оперишу само са две цифре: **0** и **1**, и на бази њих производе нове податке који су, такође, приказани као нуле и јединице. На нивоу микропрограма и машинског језика, низови нула и јединица представљају инструкције, податке и меморијске локације које обрађују дигитална логичка кола. Програмери у асемблеру и језицима вишег нивоа, користе бројеве и симболичке кодове, да означе шта да се обради, како да се то уради и над којим објектом да се уради.

Бројеви и кодови се често организују у групе које представљају врло различите појаве из света који нас окружује. Организоване групе података познате су као структуре података, и значајне су како програмерима, тако и оперативном систему.

Чак и мали персонални рачунари за већину људи сувише су сложени да би их они користили директно. Читав низ системских програма, познатих као оперативни систем (**operating system**), служи као заштитна школка или спрега која раздваја корисника и апликационе програме од сложености хардвера. Оперативни систем, сам за себе, ради врло мало времена и без конкретних резултата обраде. Његов задатак је опслуживање, односно он чини окружење у коме се ради.

Ма колико неки рачунарски систем био велики, он ипак има ограничене ресурсе (уређаји из састава рачунарског система), унутар којих се корисснички програми морају извршавати. Оперативни систем у процесу извођења програма врши поделу тих ресурса. Један од главних ресурса које он контролише је оперативна меморија. Сваки корисник и апликација захтева да меморија рачунара садржи његове програме и податке. Ко ће, када и колико оперативне меморије добити, одређује оперативни систем.

Логичка мрежа која извршава аритметичке и логичке операције, налази се у централној јединици за обраду (**central processing unit, CPU**), а многи рачунари имају само једну CPU која мора да се дели на више корисника или програма. Оперативни систем одређује ко и колико дуго има приступ централном процесору.

Уређаји из састава рачунарског система раде под контролом оперативног система, који у себи садржи програме који кориснику и његовом програму омогућавају коришћење дискета, дискова или других меморијских медија, затим штампача (принтера), терминала, комуникационих линија итд.

1.4. ЗАКЉУЧАК

Од давнина, људи у процесу израчунавања користе разна помоћна средства са циљем да себи олакшају и вишеструко убрзају поступке обраде, добијања тачних резултата и доношења правилних одлука. Данас су рачунарски системи постали неизоставни саставни део обављања многих послова, нарочито у сфери науке, технике и обраде података. Но, исто тако велику примену нашли су и у свакодневном животу, првенствено захваљујући развоју и примени микрорачунара (нарочито са појавом микропроцесора).

У састав рачунарског система улазе разни технички уређаји, програми који решавају неки проблем или помажу у коришћењу уређаја или решавању конкретних проблема, подаци који се обрађују, процедуре које обезбеђују ефикасно коришћење рачунарског система и људи, програмери или корисници који уз помоћ рачунара решавају неке своје проблеме.

Сваки интелектуални поступак, који можемо изразити помоћу коначног броја елементарних операција, може се предати рачунарском систему да га изведе. Тиме се постиже преношење једног дела рутинског интелектуалног рада на машину. Међутим, за реализацију овог врло важног вида ослобађања човека од заморног рутинског рада, потребно је много сложених поступака, а њихова израда, проверавање, извођење и коришћење тражи много труда и посла.

Основу за израду рачунара представљају дигитална логичка кола која оперишу са само две цифре: 1 и 0. Све информације и подаци унутар рачунарског система приказани су у облику низова бинарних бројева. Ови бројеви се организују у веће целине ради лакшег памћења и обраде.

Један од кључних и, слободно се може рећи, најсложенијих делова рачунарског система, који омогућава преношење великог дела интелектуалног рада на машину, јесте системски софтвер. Главни део системског софтвера чини оперативни систем. Под контролом оперативног система ради не само корисников програм већ и сви уређаји из састава рачунарског система.

1.5. ПИТАЊА

1. Описати сваки од пет саставних делова општег модела рачунарског система.
2. Која се средства користе у поступку ручне обраде података?
3. Како се аутоматизује поступак обраде података помоћу рачунара?
4. Објаснити функционалну блок шему рачунара и функције сваког од блокова.
5. Какве везе постоје између разних нивоа рачунарског хардвера и софтвера, садржаних у хијерархијском моделу рачунарског система?
6. Да ли су ресурси рачунарског система ограничени, и шта ради оперативни систем?

1.6. КЉУЧНЕ РЕЧИ

- аритметичко–логичка јединица (**Arithmetic and Logic Unit, ALU**)
- асемблер (**assembler**)
- централна јединица за обраду (**Central Processing Unit, CPU**)
- дигитална логичка кола (**digital logical circuit**)
- инструкције (**instructions**)
- излаз (**output**)
- контролно–управљачка јединица (**Control Unit, CU**)
- машински језик (**machine language**)
- меморија (**memory, store**)
- микропрограм (**microprogram**)
- обрада (**processing**)
- оперативни систем (**operating system**)
- податак (**data**)
- програм (**program**)
- програми (**software**)
- програмски језик (**programming language**)
- рачунар (**computer**)
- рачунарски систем (**computer system**)
- структуре података (**data structures**)
- технички део система (**hardware**)
- улаз (**input**)
- виши програмски језик (**high level language**)

2. МАТЕМАТИЧКЕ ОСНОВЕ РАЧУНАРА

Основни објекти који се памте и обрађују у рачунарском систему су бројеви. Бројеви су основа функционисања и коришћења рачунара на свим нивоима, почев од нивоа дигиталних логичких кола, до обраде на корисничком нивоу. Бројеви се користе не само у аритметичким операцијама, већ се и саме рачунарске инструкције приказују као низови бројева. Специјални знаци–симболи такође се кодирају као бројеви. На почетку изучавања бројева који се користе у рачунару, треба најпре размотрити структуру бројних система.

2.1. БРОЈНИ СИСТЕМИ

Бројни систем представља начин приказивања било ког броја помоћу низа симбола који се називају цифре бројног система, но, истовремено, он представља и скуп правила по којима се реализују основне операције над бројевима.

Скуп појединих цифара које се могу приказати у рачунару је једна од компоненти његовог бројног система. Модерни рачунари користе бинарни бројни систем, који има само две цифре: **0** и **1**. Бинарни систем је изабран, јер рачунар мора бити способан да прикаже било коју цифру на јединствен начин, а постоји велики број електронских склопова који могу да се налазе у два јединствена стабилна стања. Ова стања могу бити: **отворен–затворен**, **висок–низак**, **лево–десно**, или **укључен–искључен**. Много је теже реализовати електронске склопове који ће имати три, четири или више различитих стабилних стања. Сем тога бинарни систем је погодан за представљање једне области математичке логике, познате као Булова (**Boolean**) алгебра, која оперише са бинарним бројевима и обезбеђује теоријску основу за развој основних рачунарских компоненти (дигиталних логичких кола и мрежа).

Постоје две основне врсте бројних система: **позициони** и **непозициони**.

Ако једна цифра има увек исту вредност, без обзира на ком се месту у запису броја налази, онда је то непозициони систем. Типичан пример је **римски бројни систем** са цифрама: **I, V, X, L, C, D, M**.

Посматрајмо римске бројеве **III** и **IX**.

Свака цифра Римског система увек означава једну исту вредност, у овом примеру цифра I увек има вредност један (1). У првом примеру три **I** имају заједно вредност три, а у другом примеру **I** опет има вредност један само се одузима од **X** (десет), јер се мања цифра налази испред веће.

Ако једна иста цифра има различите вредности, одређене положајем цифре у низу (цифара) који представља запис броја, онда је то позициони, или тежински бројни систем.

Посматрајмо арапске бројеве 111, 27, и 207.

Код **тежинског бројног система,** у првом броју, свака од три јединице има различиту вредност: прва с лева вреди **100** (сто), средња **10** (десет), а десна **1** (један), тј. свака позиција има своју тежину. Пошто је позиција врло битна, у оваквим системима нужно мора да постоји специјална цифра **НУЛА** (**0**), помоћу које се означава позиција која не садржи ни једну цифру. Без нуле (**0**) између бројева двадесет седам (**27**) и двеста седам (**207**) не би било разлике у запису.

Свака позиција има тежину, тј. вредност која зависи од базе (основе) бројног система. Ми у свакодневном животу користимо систем са основом **10 - декадни бројни систем**. Позициони бројни систем има више јединствених цифара, укључујући и нулу, али ни у једном бројном систему не може постојати појединачна цифра једнака основи система.

База система једнака је броју различитих цифара **S**, а саме цифре бројног система су:

0,1, . . . , (**S-1**), а у декадном систему цифре су (0,1,2,3,4,5,6,7,8, и 9).

Основа бројног система увек се записује као сложени број. У декадном систему то је двоцифрен број 10, тј. састоји се од јединице иза које следи цифра нула.

У општем случају, произвољан број **x** се може у бројном позиционом систему са основом **S** представити као сума:

$$\mathbf{x = a_r S^r + a_{r-1} S^{r-1} + ... + a_1 S^1 + a_0 S^0 + a_{-1} S^{-1} + ... + a_{-p} S^{-p} + ...}$$

где су: $\mathbf{a_r , a_{r-1} , ... , a_2 , a_1 , a_0 , a_{-1} , ... , a_{-p} ,...}$ цифре броја, и припадају скупу $\{\mathbf{0, 1, ... , S-1}\}$, тј. припадају групи од **S** различитих цифара.

Број **x** се може приказати у сажетом облику као низ:

$$\mathbf{x = a_r\ a_{r-1}\ ...\ a_2\ a_1\ a_0 .\ a_{-1}\ a_{-2}\ ...\ a_{-p}}$$

где тачка (**.**) или запета (,), одваја цео део од разломљеног дела, и при чему свака цифра $\mathbf{a_i}$ у запису има неку тежину $\mathbf{S^i}$, зависно од њене позиције **i,** рачунајући лево и десно од децималне тачке.

За унос нумеричких информација у рачунар и за њихово штампање користе се:

- декадни,
- хексадекадни (хексадецимални),
- октални и
- бинарни бројни системи.

У литератури се за означавање ових бројних система користе скраћенице: **DEC, HEX, OCT** и **BIN**.

ДЕКАДНИ бројни систем има десет цифара које узимају целобројне вредности **0,1,2,3,4,5,6,7,8,9,** а сваки број **x** се може представити као:

$$\mathbf{x = d_r\,10^r + d_{r-1}\,10^{r-1} + ... + d_1\,10^1 + d_0\,10^0 + d_{-1}\,10^{-1} + ... + d_{-p}\,10^{-p} + ...,}$$

Тежинске вредности одређене су позицијом цифре и износе $10^r,\ldots\ 10^2,\ 10^1,\ 10^0,\ 10^{-1},\ 10^{-2},\ldots,\ 10^{-p}$, а $d_r,\ldots,\ d_0,\ d_{-1},\ldots,\ d_{-p}$ су децималне цифре. Примери децималних бројева су:

$$1992 = 1*10^3 + 9*10^2 + 9*10^1 + 2*10^0\ ,$$

$$738{,}387 = 7*10^2 + 3*10^1 + 8*10^0 + 3*10^{-1} + 8*10^{-2} + 7*10^{-3}\ .$$

ХЕКСАДЕЦИМАЛНИ бројни систем има 16 цифара које узимају целобројне вредности од **0** до **15,** при чему се за приказ цифара од **10** до **15** користе слова **A,B,C,D,E,F.** У овом бројном систему сваки број се може написати у облику:

$$\mathbf{x = h_r\,16^r + ... + h_1\,16^1 + h_0\,16^0 + h_{-1}\,16^{-1} + ... + h_{-p}16^{-p} + ...\ ,}$$

што значи да су тежинске вредности одређене позицијом цифре и износе $16^r,\ ...\ ,\ 16^2,\ 16^1,\ 16^0,\ 16^{-1},\ 16^{-2},\ ...\ ,\ 16^{-p}$, а $h_r,\ ...\ ,\ h_0,\ h_{-1},\ ...\ ,\ h_{-p}$ су хексадецималне цифре. Примери хексадецималних бројева су:

$$12F_{16} = 1*16^2 + 2*16^1 + 15*16^0 = 303_{10}\ ,$$

$$1F9A.A_{16} = 1*16^3 + 15*16^2 + 9*16^1 + 10*16^0 + 10*16^{-1} = 8090.625_{10}.$$

ОКТАЛНИ бројни систем има 8 цифара **0,1,2,3,4,5,6,7**, тј. основа му је **8**, а број се представља као:

$$\mathbf{x = O_r\,8^r + ... + O_1\,8^1 + O_0\,8^0 + O_{-1}\,8^{-1} + ... + O_{-p}8^{-p} + ...,}$$

где свака цифра $\mathbf{O_i}$ узима вредност из скупа $\{\mathbf{0,1,2,....,6,7}\}$, а тежинска вредност јој зависи од позиције **i,** и износи 8^i, као у следећем примеру:

$$2057_8 = 2*8^3 + 0*8^2 + 5*8^1 + 7*8^0 = 1071_{10}.$$

БИНАРНИ бројни систем такође спада у позиционе бројне системе, и при томе сваки број **x** може се приказати само уз помоћ две цифре: **0** и **1,** јер му је основа **2,**

$$x = b_r 2^r + ... + b_1 2^1 + b_0 2^0 + b_{-1} 2^{-1} + ... + b_{-p} 2^{-p} + ...$$

где је: b_i или **0** или **1** за свако **i**. Примери бинарних бројних записа су:

$11001.1101_2 = 1*2^4 + 1*2^3 + 0*2^2 + 0*2^1 + 1*2^0 + 1*2^{-1} + 1*2^{-2} + 0*2^{-3} + 1*2^{-4} = 25.8125_{10}.$

$1101011.01_2 = 107{,}25_{10}$.

2.2. КОНВЕРЗИЈА БРОЈЕВА ИЗ ЈЕДНОГ БРОЈНОГ СИСТЕМА У ДРУГИ

Конверзија бројева из **BIN, OCT, HEX**, у декадни бројни систем врши се једноставном операцијом сумирања елементарних производа цифара и њихових тежинских вредности, при чему се читава аритметика изводи у декадном систему:

$1F9A_{16} = 1*16^3 + 15*16^2 + 9*16^1 + 10*16^0 = 8090_{10}.$

$1267_{(8)} = 1*8^3 + 2*8^2 + 6*8^1 + 7*8^0 = 695_{10}.$

2.2.1. КОНВЕРЗИЈА ДЕКАДНИХ БРОЈЕВА У ДРУГЕ БРОЈНЕ СИСТЕМЕ

2.2.1.1 Цели бројеви

Декадни број **x** конвертује се у број са основом **S** методом *сукцесивних дељења*, преко следећег низа корака:

- Број **x** се подели основом **S**. Остатак при дељењу памти се као цифра најмање тежине (S^0) траженог броја са основом S,
- Добијени количник се поново подели са **S**. Остатак представља цифру са тежином S^1, траженог броја са основом **S**.
- Поступак се понавља све док резултат дељења не постане једнак нули.

Поступак конверзије целог декадног броја у бинарни број приказаћемо на примеру броја 120_{10}:

120 : 2 = 60 60 : 2 = 30 30 : 2 = 15 15 : 2 = 7 7 : 2 = 3 3 : 2 = 1 1 : 2 = 0

0 0 0 1 1 1 1 *остатак*

цифра најмање тежине — *цифра највеће тежине*

Низ бинарних цифара треба прочитати од краја ка почетку, од последњег остатка ка првом, а записује се у једном од следећих облика:

$$120_{(10)} = 1111000_{(2)},$$
$$(120)_{10} = (1111000)_2,$$
$$120_{10} = 1111000_2.$$

2.2.1.2 Декадни број мањи од јединице

Декадни број $\mathbf{x}$ мањи од један $(\mathbf{x=0.a_{-1}a_{-2}} \ldots)$ претвара се у одговарајући број са основом $\mathbf{S}$ методом *сукцесивних множења*, на следећи начин:

- Помноже се број $\mathbf{x}$ и основа $\mathbf{S}$. Целобројна вредност овог производа представља цифру са тежином $\mathbf{S^{-1}}$ траженог броја у систему са основом $\mathbf{S}$. Децимални део производа служи за добијање других цифара,
- Друга цифра са тежином $\mathbf{S^{-2}}$ добија се множењем децималног дела из претходног множења са основом S, и представља целобројну вредност овог производа.
- Поступак се понавља све док се не добије тражена тачност.

Поступак није у општем случају коначан, а прекида се када се добије одређен број цифара којим се може задовољити тражена тачност пред–стављања бројева, или када је добијен цео број (нема део < 1). Ово даље значи, да тачни декадни децимални бројеви немају тачан бинарни еквива–лент, па се морају тражити и други начини представљања декадних бројева. Поступак сукцесивних множења приказаћемо на примеру конверзије разломљеног декадног броја 0.375, у бинарни бројни систем:

	целобројни део
0.375 . 2 = 0,75	0
0.75 . 2 = 1,5	1
0.5 . 2 = 1,0	1

где је као резултат конверзије добијен број:

$$(0{,}375)_{10} = (0{,}011)_2.$$

2.2.1.3 Конверзија мешовитих декадних бројева

Конверзија мешовитих декадних бројева (цео део + разломљен), врши се тако што се цео део конвертује методом сукцесивних дељења, док се разломљен део конвертује методом сукцесивних множења. При томе се цео део увек претвара у коначан низ **0** и **1,** док разломљени део не мора имати коначан приказ, тј. његова конверзија се не може извести до краја већ са одређеном тачношћу, односно апроксимативно.

2.2.2. КОНВЕРЗИЈА BIN, OCT и HEX БРОЈЕВА У ДРУГЕ БРОЈНЕ СИСТЕМЕ

Бинарни број се конвертује у декадни (BIN → DEC) већ описаним поступком сумирања елементарних прозвода бинарних цифара и њихових тежинских коефицијената 2^i, као у следећем примеру:

$$1101.11_2 = 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 + 1*2^{-1} + 1*2^{-2} = 13.75_{10}.$$

2.2.2.1 Конверзија бинарног броја у октални и хексадецимални

Конверзија бинарног броја у октални и хексадецимални врши се груписањем три, односно четири бинарне цифре, почев од децималне тачке улево и удесно. Када се бинарни број конвертује у октални (BIN → OCT), групишу се по три бинарне цифре, и на основу табеле **1** одређује се одговарајућа октална цифра. Ако при груписању у крајње левој и крајње десној групи нема довољног броја бинарних цифара, дописује се потребан број нула (једна или две). При конвертовању из бинарног у хексадецимални систем (BIN → HEX), поступак је еквивалентан, само се праве групе од по четири бинарне цифре, почев од децималне тачке.

Поступак конверзије ћемо приказати на примеру бинарног броја $10111011011101.11111100010_2$:

$10111011011101.11111100010_2$= 0 10 111 011 011 101.111 111 000 10 0 =
= 27335.7704_8.

$10111011011101.11111100010_2$= 00 10 1110 1101 1101.1111 1100 010 0 =
= $2EDD.FC4_{16}$.

2.2.2.2 Конверзија окталног и хексадецималног броја у бинарни

Обзиром на чињеницу да је $8=2^3$, а $16=2^4$, то се свака октална цифра може кодирати помоћу 3 бинарне цифре, а хексадецимална помоћу 4 бинарне цифре, са тежинским факторима $\mathbf{P_1=2^0=1,\ P_2=2^1=2,\ P_3=2^2=4}$, а за хексадецималне $\mathbf{P_1=2^0=1,\ P_2=2^1=2,\ P_3=2^2=4,\ P_4=2^3=8}$ **.**

То значи да се октални број претвара у бинарни(OCT → BIN), заменом сваке окталне цифре њеним троцифреним бинарним записом. Хексадецимални број претвара се у одговарајући бинарни (HEX → BIN), заменом хексадацималних цифара њиховим четвороцифреним бинарним еквивалентом. У табели **1** дати су бинарни кодови, окталних, хексадецималних и декадних цифара. Поступак конверзије ћемо приказати на примеру окталног броја 157, и хексадецималног броја ABCD:

$$(157)_8 = (001)_2\,(101)_2\,(111)_2 = 1101111_2,$$

$$ABCD_{(16)} = 1010_{(2)}\ 1011_{(2)}\ \ 1100_{(2)}\ \ 1101_{(2)} = 1010101111001101_2.$$

број	*октална цифра*	*бинарни број*	*хекса цифра*	*бинарни број*	*декадна цифра*	*бинарни број*
0	0	000	0	0000	0	0000
1	1	001	1	0001	1	0001
2	2	010	2	0010	2	0010
3	3	011	3	0011	3	0011
4	4	100	4	0100	4	0100
5	5	101	5	0101	5	0101
6	6	110	6	0110	6	0110
7	7	111	7	0111	7	0111
8	10	1000	8	1000	8	1000
9			9	1001	9	1001
10			A	1010	10	
11			B	1011		
12			C	1100		
13			D	1101		
14			E	1110		
15			F	1111		
16			10			

Табела **1.** Табела бинарних кодова хексадецималних, декадних и окталних цифара

2.2.2.3 Конверзија окталних бројева у хексадецимални и обратно

Конверзија окталних бројева у хексадецималне и обратно врши се преко бинарног система (OCT → BIN → HEX; HEX → BIN → OCT). Најпре се свака октална цифра замени са три бинарне, а затим се почев од децималне тачке улево и удесно групишу по четири бинарне цифре. Свакој групи одговара по једна хексадецимална цифра. При конвертовању хексадецималних бројева у окталне, најпре се свака хексацифра замени са четири бинарне (као у табели), а затим се почев од децималне тачке групишу по три бинарне цифре. Свакој групи од три бинарне цифре одговара по једна октална. Поступак ћемо приказати на примеру конверзије окталног броја 127.15_8 у хексадецимални:

$$127.15_8 = 001\ 010\ 111.001\ 101_2 = 0101\ 0111.0011\ 0100_2 = 57.34_{16}\ .$$

2.3. ПОЈАМ КОМПЛЕМЕНТА

Комплемент је појам који се често користи када се говори о бројним системима. Практични смисао има код приказивања негативних бројева и код операције одузимања, тачније за реализацију одузимања помоћу сабирања. Комплементи позитивних бројева су исти као и сам тај број.

Најопштија, упрошћена дефиниција комплемента би била да је комплемент допуна датог броја до неке унапред дефинисане вредности.

Иако се у сваком бројном систему може дефинисати онолико различитих комплемената колико цифара има тај бројни систем, од практичног значаја су само комплемент до броја који представља бројну основу система и комплемент до највеће цифре у систему. На пример, за декадни бројни систем то су **комплемент десетке** (јер је десет основа бројног система) и **комплемент деветке** (јер је девет највећа цифра у декадном бројном систему).

Уместо сложених дефиниција, појам ова два комплемента дефинисаћемо кроз пример. Посматрајмо један четвороцифрени декадни број X (рецимо 3704). Комплемент деветке овога броја је поново четвороцифрени број који треба додати овом броју да би се добио четвороцифрени број састављен само од деветки (9999).

Дакле, *комплемент деветке* броја **3704** је број **6295** јер је:

3704	(полазни број $\mathbf{X}$)
+ 6295	(*комплемент деветке броја* $\mathbf{X}$)
9999	(збир 9999)

Слично томе, комплемент деветке броја 37 је број 62, а комплемент деветке броја 912 је број 087.

Комплемент десетке (бројне основе система) броја X је дефинисан као број који додат броју X даје резултат који има све нуле и пренос (јединицу на месту пете цифре за четвороцифрени број у примеру). Тако је комплемент десетке броја X = **3704**, број **6296** јер је:

3704	(полазни број $\mathbf{X}$)
+ 6296	(***комплемент десетке*** *броја* $\mathbf{X}$)
1 0000	(збир 10000)

Слично томе, комплемент десетке броја 37 је број 63, а комплемент десетке броја 912 је број 088. Треба уочити да, када се четвороцифрени број X сабере са својим комплементом десетке, као четвороцифрени резултат се добија нула (ако се пета цифра сматра вишком који бива одбачен у системима који раде са највише четири цифре). Ова особина која произилази директно из дефиниције комплемента десетке искоришћена је за приказивање негативних бројева.

Слично комплементима десетке и деветке у декадном бројном систему, у октално систему су од значаја комплементи седмице и осмице. Дефинисани су аналогно дефиницијама одговарајућих комплемената у декадном бројном систему као допуна до 7777_8 (за четвороцифрени број), односно до 10000_8, респективно. У хексадецималном систему, комплемент петнаестице је допуна до $FFFF_{16}$, а комплемент шеснаестице је поново допуна до 10000_{16}.

Из примера је лако уочити да је комплемент десетке увек за 1 већи од комплемента деветке истог броја. Ово је особина која важи за све бројне

системе: *Комплемент бројне основе је за један већи од комплемента највеће цифре*, и ова особина се често користи за израчунавање комплемента бројне основе тако што се комплемент највеће цифре (који се добија лако у свим бројним системима) једноставно увећа за један. Комплемент бројне основе се добија тако што се све нуле са десне стране броја препишу, прва ненулта цифра сдесна комплементира се до основе система, а остале цифре комплементирају се до највеће цифре.

2.3.1. КОМПЛЕМЕНТИ БИНАРНОГ БРОЈА

У бинарном бројном систему могу се дефинисати само два комплемента и оба су од практичног значаја. То су **комплемент јединице** (највеће цифре у систему) и **комплемент двојке** (бројне основе система).

За четвороцифрени бинарни број $\mathbf{X}$ (на пример $\mathbf{1011}_2$), *комплемент јединице* је допуна до броја $\mathbf{1111}_2$, што за број $\mathbf{X}$ из примера износи $\mathbf{0100}_2$ јер је:

$\mathbf{1011}_2$	(полазни број $\mathbf{X}$)
$+\ \mathbf{0100}_2$	(*комплемент јединице* броја $\mathbf{X}$)
$\mathbf{1111}_2$	(збир $\mathbf{1111}_2$)

Треба приметити да се комплемент јединице бинарног броја може добити тако што се свака бинарна цифра полазног броја промени (инвертује), па од јединице постаје нула, а од нуле, јединица.

Комплемент двојке је за четвороцифрени бинарни број $\mathbf{X}$ из примера ($\mathbf{1011}_2$), допуна до броја $\mathbf{10000}_2$, што за број $\mathbf{Y}$ износи $\mathbf{0101}_2$ јер је:

$\mathbf{1011}_2$	(полазни број $\mathbf{Y}$)
$+\ \mathbf{0101}_2$	(**комплемент двојке** броја $\mathbf{Y}$)
$\mathbf{10000}_2$	(збир $\mathbf{10000}_2$)

Комплементи у бинарном бројном систему играју веома важну улогу. Вероватно је то један од разлога што постоји више термина за појам комплемента јединице и комплемента двојке.

Комплемент јединице се назива још и инверзни код, непотпуни комплемент или **први комплемент.**

Комплемент двојке се назива још и допунски код, потпуни комплемент или **други комплемент**.

Иако не одговарају у потпуности дефиницијама, термини (**први комплемент** и **други комплемент**), највише се користе у рачунарској терминологији. Због тога ће се и у овом уџбенику надаље углавном користити управо ови термини, а у табели 2. дато је неколико примера комплемената.

X	1011_2	10100011_2	00000000_2	11111111_2	01010_2	11110_2	00000001_2
1. комп.X	0100_2	01011100_2	11111111_2	00000000_2	10101_2	00001_2	11111110_2
2. комп.X	0101_2	01011101_2	00000000_2	00000001_2	10110_2	00010_2	11111111_2

Табела 2. Неколико примера бинарних бројева и њихових комплемената

Као и у другим бројним системима, и у бинарном важи да је комплемент основе бројног система, у овом случају други комплемент, за један већи од комплемента највеће цифре, у овом случају, првог комплемента, што је најједноставнији пут за израчунавање. Најпре се израчуна први комплемент броја тако што се свака бинарна цифра полазног броја промени (од нуле на јединицу и обрнуто), па се затим добијеном бинарном броју дода јединица.

Сабирање са јединицом обавља се по принципима бинарног сабирања описаном у поглављу 2.4.1. Други комплемент негативног бинарног броја добија се тако што се све нуле и прва јединица с десне стране препишу а остале цифре се инвертују.

Поновимо неке важне особине комплемената бинарних бројева:

- Први комплемент се добија инвертовањем сваког бита полазног бинарног броја.
- Други комплемент се добија додавањем јединице на први комплемент.
- Први и други комплемент имају онолико бинарних цифара колико и полазни број.
- Други комплемент негативне нуле је поново нула.

2.4. БИНАРНИ БРОЈНИ СИСТЕМ

Једна бинарна цифра **0** или **1** представља минималну количину информација, односно најмањи податак који се може обрадити у рачунару, и назива се **бит (bit)**. Када се посматра бинарни запис неког броја, први бит слева је бит највеће тежине **MSB** (**Most Significant Bit**), а први бит здесна, бит најмање тежине **LSB** (**List Significant Bit**).

У већини модерних рачунара користи се група од осам битова која се назива **бајт–byte** (**1 бајт = 8 бита**). Бајт представља две хексадецималне цифре. Један бајт се састоји од две *тетраде* (1 **nibble** = 4 **bit**–а = 1 **полубајт**), од којих свака представља једну хексадецималну цифру.

Код микрорачунара основни податак који се може сместити у унутрашњу меморију представља један бајт, односно група од 8 бита. Већи рачунари, најчешће, меморишу податке у групама од 2, 4, или више бајта, и називају

се **меморијске речи** (регистар). Постоје, такође, рачунари који уопште не користе бајтове. Основна јединица меморисаних података и код ових рачунара зове се **реч** (**word**), па неки рачунари имају реч (регистар) од 36 бита, а неки реч од 60 бита. Једна меморијска реч је највећа количина информација која се у једном циклусу (приступу) може прибавити из меморије, или у њу сместити.

Снага једног рачунара умногоме зависи од дужине меморијске речи. Са повећањем меморијске речи повећава се брзина преноса података између појединих делова рачунара.

2.4.1. БИНАРНО САБИРАЊЕ

Бинарно сабирање је врло једноставно јер постоје само четири правила. Прва три су врло проста: 0+0=0, 0+1=1 и 1+0=1. Проблем се јавља код случаја 1+1=10. Како у један бит може бити записана само једна цифра, резултат сабирања је 1+1=0 и јавља се пренос једне 1 у следећи бит (бит веће тежине). Код сабирања вишецифрених бинарних бројева, при сабирању цифара треба узети у обзир и пренос (**carry**) из претходног бита, сем код бита најмање тежине (**LSB**).

Размотримо следећи проблем, сабирање два бинарна броја: $0101_{(2)}$ + $0111_{(2)}$.

Збир цифара најмање тежине LSB (крајње десне цифре у запису броја) је $10_{(2),}$ односно нула са преносом један.

Збир следећих цифара (0 и 1) је 1, али треба додати пренос из LSB, па је збир опет $10_{(2)}$, тј. нула са преносом у следећу колону.

Збир у трећој колони је: **1+1+пренос** даје $11_{(2)}$,тј. збир је 1 и пренос **C** (**carry**) је 1.

На крају, последња цифра је резултат збира **0+0+пренос**, што даје један, па је коначни резултат сабирања $1100_{(2).}$

2.5. БРОЈЕВИ СА ЗНАКОМ, КОДИРАЊЕ НЕГАТИВНИХ БРОЈЕВА

Рачунар све податке представља помоћу бинарног записа одређене дужине. У савременим рачунарима дужина бинарног записа је најчешће умножак броја 8. Дакле, дужина записа може бити 8, 16, 32, 64... бита (мада постоје и рачунари са дужинама записа које нису умножак броја 8). Ако се користи бинарни запис дужине осам бита, за такве податке се каже да су осмобитни. Како се скуп осам бита назива **бајтом**, то се за осмобитне податке каже још и да су **једнобајтни** (заузимају један бајт). Сходно томе постоје и двобајтни (шеснаестобитни), четворобајтни (тридесетдвобитни) и тако даље.

Једнобајтни подаци су они који су записани бинарним записом дужине један бајт (осам бита).

Дужина бинарног записа директно одређује колико се различитих података може записати. Лако је показати да се са записом дужине N бита може записати 2^N различитих података. Ако би се користио само један бит, помоћу њега се могу записати два податка: Један који одговара јединици и један који одговара нули. Са два бита могу се кодовати 4 различита податка. Први одговара комбинацији бинарних бројева 00, други комбинацији 01 (бинарно), трећи комбинацији 10 (бинарно) и последњи, четврти, податак комбинацији 11 (бинарно). Са 8 бита може се записати укупно $2^8 = 256$ различитих података. Свих расположивих 256 различитих података може се искористити да се прикаже 256 позитивних бројева (од 0 до 255) и тада се за такве податке каже да су **неозначени** (*unsigned*). С друге стране, ако је потребно користити и позитивне и негативне бројеве, део расположивих 256 могућности може се искористити да се прикажу негативни бројеви. Нормално, укупан број различитих података је поново 256. Подаци који садрже и позитивне и негативне бројеве називају се **означени** (*signed*).

Треба напоменути да термин **означени подаци** не значи да су ти подаци негативни бројеви, већ да могу бити и позитивни и негативни. Позитивни бројеви се могу представљати и као означени и као неозначени, негативни, само као означени. Ова два појма **означени** подаци (*signed*) и **неозначени** подаци (*unsigned*) су врло карактеристична за рачунарску терминологију и провлаче се кроз скоро све области везане за обраду нумеричких података, па је стога од велике важности јасно их дефинисати и уочити разлику међу њима. У зависности од дужине бинарног записа и од тога да ли је потребно представљати само позитивне (или и позитивне и негативне) бројеве, могуће је дефинисати неколико типова података:

- **Означени бајт** и **неозначени бајт** (једнобајтни, осмобитни запис).
- **Означена реч** и **неозначена реч** (двобајтни, шеснаестобитни запис).
- **Означени** и **неозначени дворечни (long)** (четворобајтни, тридесетдвобитни запис).

Конкретни називи типова података варирају од језика до језика и од примене до примене, али готово сви виши програмски језици користе ових шест типова података дефинисаних на наведени начин.

Код осмобитних (или једнобајтних) података, једнобајтни податак може бити **означен бајт** или **неозначен бајт**. Притом се помоћу типа неозначени бајт може представити 256 позитивних бројева (0 до 255) а помоћу типа означен бајт може представити поново 256 различитих бројева али овај пут и позитивних и негативних (и то бројеви у опсегу од –128 до +127).

Намеће се једноставно питање: Како процесор разликује означене од неозначених података? Одговор је још једноставнији: **НИКАКО** !!! Наиме, све операције над једнобајтним подацима се изводе на идентичан начин и *процесор ни по чему не разликује податке типа бајт од података типа означен бајт*. Може се поставити и питање: Да ли је могуће на неки начин саопштити процесору да ли је податак означен или неозначен? Одговор на ово питање је негативан.

У асемблеру, као ни у другим машински–оријентисаним језицима, не постоји начин да се одређени податак декларише као означен или неозначен! Машински–оријентисани језици разликују податке једино по величини и у њима је могуће једино разликовати једнобајтне (или осмобитне), двобајтне (или шеснаестобитне) и четворобајтне (или тридесетдвобитне). Ако је већ тако (процесор не разликује означен бајт и бајт, нити је могуће декларисати их различито), **чему онда два типа података? Ко прави разлику?**

Разлику прави **програмер** и он је тај који је дужан да води рачуна о томе који једнобајтни податак треба да буде типа бајт, а који типа означен бајт. После сваке аритметичке операције, процесор дефинише (поставља или брише) индикаторе–заставице (флегове) у регистру стања, а програмер сам мора да *тумачи* резултат на основу постављених заставица. Постоје заставице које су значајне при тумачењу резултата операција над означеним подацима, а постоје и заставице значајне за рад са неозначеним. Независно од тога да ли су улазни подаци означени или не, **обе групе индикатора се постављају / бришу**, програмер је тај који треба да одлучи **које ће индикаторе користити за тумачење резултата**, а које неће узимати у обзир!

Разумевање одговора на претходна питања је од изузетне важности за правилно програмирање аритметичких операција. Претходна разматрања односе се искључиво на принцип рада самог процесора и обраду података коришћењем машински–оријентисаних језика.

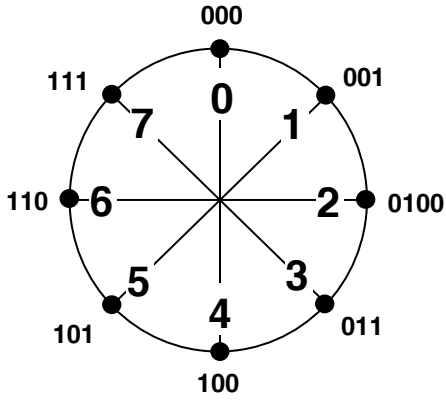
Виши програмски језици, тачније њихови преводиоци преузимају на себе поменуто тумачење резултата па је у њима могуће декларисати различито означене податке од неозначених. Треба напоменути још и да неки процесори имају различите инструкције за означене и неозначене податке (на пример, инструкције множења и дељења постоје и за означене и за неозначене податке), али је поново програмер тај који мора да одлучи коју од инструкција ће применити (зависно од тога како тумачи улазне податке).

2.5.1. КОДИРАЊЕ НЕГАТИВНИХ БРОЈЕВА – ДРУГИ КОМПЛЕМЕНТ

Претпоставимо да податке записујемо са три бита. Сви закључци ће важити и за осмобитне, шеснаестобитне и тридесетдвобитне податке. Тробитни подаци су узети као пример само због једноставности. Помоћу N бита могуће је представити 2^N различитих података. Према томе, помоћу три бита могуће је приказати укупно 2^3=8 података. То може бити осам бројева од 0 до 7, али исто тако може бити и осам различитих боја, или осам бројева од 13

до 20. Ако желимо да радимо и са позитивним и са негативним бројевима, а да при том задржимо и приказивање нуле, у обзир долазе комбинације од –4 до 3 или од –3 до 4. Погледајмо комбинације бита, означене римским бројевима:

		неозначени
0	**000**	**0**
I	**001**	**1**
II	**010**	**2**
III	**011**	**3**
IV	**100**	**4**
V	**101**	**5**
VI	**110**	**6**
VII	**111**	**7**



Када се ради искључиво са позитивним бројевима (неозначени подаци), ту углавном, нема недоумица. Комбинација означена редним бројем I одговара бинарном коду броја 1, комбинација број VI, одговара бинарном коду броја 6 и тако са сваком од осам комбинација. Тиме је трећа колона дефинисана. Корисно је замислити све бројеве распоређене у круг као на дијаграму са десне стране. Наиме, бинарним увећавањем за 1, тачка на кругу се помера за једно место удесно (у смеру кретања казаљке на сату). То је тачно и када се 7 увећа за 1, у тробитном систему се добија нула као резултат. *Сабирање са бројем K је у ствари померање удесно по кругу за K места, одузимање, померање улево.* Тако је у тробитном систему 6+3=1. Ова особина је последица чињенице да се бинарно сабирање обавља **по модулу 2^N** где је **N** број бита (дужина) бинарног записа. За систем са три бита, аритметичке операције се обављају **по модулу 8** што значи да ће сваки резултат **R** већи од 7 бити замењен остатком дељења тог броја са 8 (**R** по модулу 8). Тако се уместо 6+3=**9**, као резултат добија остатак дељења 9 са 8 (9 по модулу 8) што је **1**. Ова замена је нешто што је природни, саставни део бинарне аритметике.

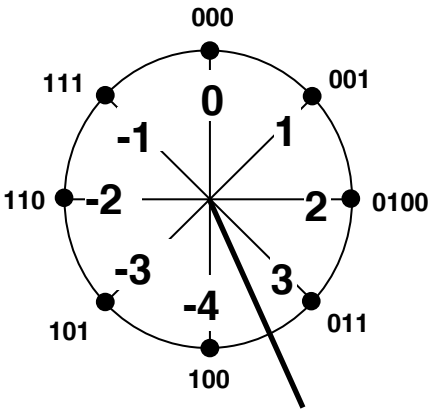
Поставља се питање како представити негативне бројеве. Негативни број је математичка фикција. На пример, –1 се може посматрати као број који сабран са бројем 1 даје нулу. Покушајмо међу овим комбинацијама да пронађемо број који у збиру са 1 даје нулу. То је комбинација VII:

```
  0 0 1      (I)
+ 1 1 1      (VII)
1 0 0 0
```

Јединца на месту најзначајнијег бита (MSB) је четврти бит и у систему са тробитним подацима ће једноставно бити одсечена. Према томе, бинарним

сабирањем комбинација I и VII, добија се нула, па је згодно комбинацију VII прогласити бројем –1. На тај начин су добијени и остали негативни бројеви у четвртој колони следеће табеле:

		Неозначени	Означени
0	**000**	**0**	**0**
I	**001**	**1**	**1**
II	**010**	**2**	**2**
III	**011**	**3**	**3**
IV	**100**	**4**	**-4**
V	**101**	**5**	**-3**
VI	**110**	**6**	**-2**
VII	**111**	**7**	**-1**



Ово представља негативне бројеве у другом комплементу. Поред ове особине, други комплемент има још пуно корисних особина које се могу искористити код аритметичких операција тако да је постао готово искључива техника за представљање негативних бројева.

На кружном дијаграму се виде места која заузимају негативни бројеви. Прелаз преко црне подебљане линије представља прекорачење код оваквог тумачења бинарног сабирања. Ако се у току аритметичких операција пређе преко ове линије, заставица која означава прекорачење се поставља.

Други комплемент се може добити додавањем јединице на први комплемент, а први комплемент се добија инвертовањем свих јединица у нуле и обрнуто. Тако број –2 можемо добити полазећи од 2 на следећи начин:

(број 2) **0 1 0** ➔ **1 0 1** (први комплемент броја 2)

(први комплемент броја 2) **1 0 1**
+ 1
1 1 0 (други комплемент броја 2=**–2**)

Општи облик записа означених бројева са N бита би био следећи:

$$- b_{N-1} \cdot 2^{N-1} + b_{N-2} \cdot 2^{N-2} + \ldots + b_1 \cdot 2^1 + b_0 \cdot 2^0$$

Ради поређења, општи облик записа *неозначених* бројева се разликује само у првом члану:

$$b_{N-1} \cdot 2^{N-1} + b_{N-2} \cdot 2^{N-2} + \ldots + b_1 \cdot 2^1 + b_0 \cdot 2^0$$

За *означене* осмобитне податке, тај општи облик постаје:

$$- b_7 \cdot 2^7 + b_6 \cdot 2^6 + \ldots + b_1 \cdot 2^1 + b_0 \cdot 2^0, \quad \text{или}$$

$$- b_7 \cdot 128 + b_6 \cdot 64 + \ldots + b_1 \cdot 2 + b_0 ,$$

где су b_7 до b_0, бинарне цифре (0 или 1) осмобитног податка $b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$. На пример, за број 9 = 0000 1001_2, цифре b_3 и b_0 су јединице, остале су нуле.

Често се о најзначајнијем биту (MSB) говори као "биту знака" што је у принципу тачно, али са таквим тумачењем треба бити јако опрезан јер остатак – седам преосталих бита *не представљају апсолутну вредност.*

Из општег облика означених бројева могу да се сагледају неке важне особине овог начина записа:

- *Сви негативни бројеви имају јединицу као најзначајнији бит*. Међутим, није довољно само ту јединицу претворити у нулу да би се добио исти позитиван број. Ако пођемо од броја –3, у тробитном систему 101, (број V), мењањем прве јединице у нулу добијамо 001 што јесте позитиван број, али 1, а не 3! Ова чињеница се често испушта из вида.

- *Уопште није свеједно колика је дужина записа означеног броја*. Наиме, само први члан општег записа је негативан па није свеједно да ли се најзначајнији бит множи са 2^7, или на пример, са 2^2. Тако се број –3 у тробитном систему записује као 101, али у осмобитном систему број 00000101_2 није –3, већ +5. Број –3 као осмобитни податак је 1111 1101_2. Значи, када се ради са означеним бројевима мора се водити рачуна о томе колика је дужина податка. То се у пракси своди на то да означени број записан као једнобајтни податак, није исти као запис истог броја у формату реч или двострука реч (long). Да би се осмобитни негативни број проширио до шеснаестобитног, виши бајт треба напунити јединицама (11111111_2), а уколико је означен број позитиван, виши бајт треба да буде нула.

- *Опсег означених осмобитних бројева је од −128 (10000000_2) до +127 (01111111_2) док је опсег неозначених осмобитних бројева од 0 (00000000_2) до +255 (11111111_2).* Очигледно је да је опсег апсолутних вредности означених бројева мањи него код неозначених. На пример, број +129 се не може представити као означен осмобитни број.

- *Применом операције другог комплемента над бинарним записом неког броја, том броју се мења знак.* Подразумева се да се ради са означеним подацима. Рецимо, 10000001_2 је бинарни запис броја -127. Применом операције рачунања другог комплемента (први комплемент +1) над овим бинарним записом добија се 01111111_2 што је бинарни запис броја +127. Исти случај би био и ако би се кренуло од неког означеног позитивног броја, применом другог комплемента добио би се тај исти број само негативан. Поново, ово важи само за означене податке.

2.5.1.1. Примери означених и неозначених бројева

1. Ако се у меморији осмобитног рачунара налази податак 10000001_2 који децимални број је представљен овим податком?

Да би се дао одговор на ово питање, мора се располагати информацијом да ли програмер жели да ово буде означен или неозначен број. Ако је тај податак за програмера неозначен, тада је то децимални број 129 ($1\bullet2^7+0\bullet2^6+0\bullet2^5+0\bullet2^4+0\bullet2^3+0\bullet2^2+0\bullet2^1+1\bullet2^0=128+1$), ако је број означен онда је то -127 ($-1\bullet2^7+0\bullet2^6+0\bullet2^5+0\bullet2^4+0\bullet2^3+0\bullet2^2+0\bullet2^1+1\bullet2^0=-128+1$). *Дакле, број 129 и број -127 записани су на потпуно исти начин и операције које процесор изводи над њима, изводи на потпуно исти начин, не знајући да ли се ради о податку 129 или -127.* Захваљујући особинама другог комплемента, могуће је да резултат буде тачан у оба случаја.

2. Којим бинарним записом треба записати број –9?

Како се овај податак може представити само као означен број, да би се дао одговор на ово питање, мора се располагати податком о жељеној дужини бинарног записа. Најмања дужина бинарног записа је 5 бита. У том случају тај бинарни запис је **10111_2** зато што је $-9 = -1\bullet2^4+0\bullet2^3+1\bullet2^2+1\bullet2^1+1\bullet2^0$. Како бинарни записи мање дужине од осам бита нису од интереса, треба пронаћи осмобитни запис. Погрешно би било када би рекли да је осмобитни запис броја овог броја **00010111_2**. Зашто? Зато што би у том случају најзначајнији бит био 0 па би број био позитиван јер једини негативни члан општег израза записа означеног броја је управо члан уз 2^7 (најзначајнији бит).

–9 у осмобитном бинарном запису је **$111\underline{10111}_2$**.

–9 у шеснаестобитном бинарном запису је **11111111 $111\underline{10111}_2$**.

До бинарног записа негативног броја може се доћи на различите начине. Можда је најједноставнији поћи од апсолутне вредности броја (9). Апсолутна вредност је позитиван број који је лако конвертовати у бинарни. Њу треба представити као бинарни број са онолико бита колика се дужина бинарног записа тражи (00001001_2). На крају треба израчунати други комплемент добијеног бинарног броја тако што се на први комплемент (11110110_2) дода јединица. Тако се добија да је осмобитни запис траженог негативног броја (–9) бинарни број 11110111_2. *Наравно, код бинарног записа негативних бројева нужно је водити рачуна о дужини записа!*

Дакле: *Осмобитни бинарни запис броја 9 је број 00001001_2. Први комплемент овог бинарног броја је 11110110_2 па је други комплемент (који је једнак првом комплементу увећаном за 1) 11110111_2. Ово је сада осмобитни бинарни запис броја –9.*

Други начин за добијање бинарног записа негативног броја је да се по кружном дијаграму какав је дат уз табелу означених бројева од нуле

померимо за 9 места улево (–9 = 0–9). За осмобитни запис кружни дијаграм има укупно 256 тачака па се то померање налево своди на рачунање разлике 256–9 што је 247. То би значило да означени број –9 има исти код као неозначени 247. Бинарни код за 247 се израчунава неком од стандардних метода. Дакле, за осмобитни запис важи:

КОД ЗА ОЗНАЧЕНИ -9 = КОД ЗА НЕОЗНАЧЕНИ 247 = 11110111_2.

Ако би се тражио шеснаестобитни запис, одговарајући неозначени број био би 65536–9.

3. Који означен број је представљен записом 10101001_2?

Тражени број се може израчунати директно из општег облика записа означеног бинарног броја као –128+32+8+1 = –87.

Други начин је да се, водећи рачуна да је број негативан, израчуна други комплемент овог броја (комплемент комплемента броја је сам тај број) а то је 01010111_2, што је бинарни код за 87 (64+16+4+2+1). Према томе, полазни број је –87.

2.5.2. ПРЕДСТАВЉАЊЕ НЕГАТИВНИХ БРОЈЕВА ДРУГИМ ТЕХНИКАМА

Негативне бројеве могуће је представити и на друге начине. У почецима развоја коришћен је такозвани **директан код** где је бит највеће тежине (**MSB**) носио информацију о знаку, а остали бити информацију о апсолутној вредности броја. Иако је овај приступ близак представи негативних бројева у декадном систему на који су људи навикли, он има неке недостатке због којих се више не користи у рачунарској техници. Основни недостатак је што бинарно сабирање какво је имплементирано у аритметичко–логчкој јединици не даје тачне резултате уколико у сабирању учествују негативни бројеви. Логичка мрежа која се мора додати да би се тај проблем решио у ствари претвара негативни број из директног кода у број у другом комплементу.

Прикажимо шта се догађа када хоћемо да израчунамо разлику бројева 9 и 4, тј. **9 - 4**. По законима аритметике овај проблем се може приказати као 9 +(–4) тј, девет плус минус четири. У следећем примеру приказано је шта се догађа у рачунару при сабирању апсолутних вредности са знаком, бројева 9 и –4. Уочимо да је резултат нетачан (–13). Проблем је, дакле: како наћи метод за приказивање негативних бројева да би резултати аритметичких операција били коректни?

```
  +9 =  00001001
(–4) =  10000100
        10001101   =  –13
```

Друга техника која је неко време коришћена је запис негативних бројева помоћу **првог комплемента**. Принцип се састоји у томе да се негативан

број представља као први комплемет бинарног записа апсолутне вредности тог броја. На пример, број –3 би био представљен тако што би се кренуло од бинарног кода за број 3 (00000011_2) па би се израчунао први комплемент тог записа. Тако би осмобитни бинарни запис за –3 коришћењем технике првог комплемента, био 11111100_2.

Употреба првог комплемента је погодна у аритметичким операцијама. Размотримо сада проблем израчунавања израза **9-4**, пошто смо га поново написали у облику **(+9)+(-4)**.

(+9) = 00001001
(–4) = 11111011
1 00000100

Број **-4** у првом комплементу добија се инверзијом бинарних цифара броја **+4**. Како је $+4_{(10)} = 00000100_{(2)}$, то након инверзије добијамо $11111011_{(2)}$, а то је (-4_{10}). Анализирајмо добијени резултат. Уочимо да је **9-4=5**, а добијени резултат није **пет**. У ствари, резултат има чак један бит више од самих основних бројева. Тај додатни бит није бит знака, већ бит преноса **C** (**carry**), и мора се третирати на посебан начин, тј. додаје се на добијени резултат:

1 00000100
+ 1
00000101

Када се у аритметици првог комплемента генерише бит преноса, да би се добио тачан резултат, он мора бити враћен са **MSB** краја, и додат на бит најмање тежине (**LSB**). То је циклично враћање преноса, ротација бита преноса (**end-around carry**), као што је приказано у претходном примеру.

У наредном примеру је показано шта се у аритметици првог комплемента догађа при одузимању већег од мањег броја, на пример, 15–20:

$+15_{(10)} = 00001111_{(2)}$ $+20_{(10)} = 00010100_{(2)}$ $-20_{(10)} = 11101011_{(2)}$

15 – 20 = (+15) + (–20)

+15 = 00001111
–20 = 11101011
C=0 11111010 Пошто је C=0 не врши се корекција

Уочимо најпре да је бит преноса нула (па неће бити корекције резултата), а да је бит знака јединица, односно резултат је негативан, као што и треба да буде. Да бисмо видели који је то број, треба најпре да извршимо конверзију негативног броја (у првом комплементу) у позитиван. Конверзија негативног броја у позитиван, такође се врши инверзијом битова.

Инверзијом цифара броја $11111010_{(2)}$ добијамо $00000101_{(2)}$, што је једнако пет и, не заборавимо, резултат је негативан, па је дакле коначно решење задатка у ствари **-5,** као што и треба да буде. Појаву враћања јединице

преноса са излаза на улаз (са **MSB** на **LSB**) треба посматрати као цену која се мора платити, да би се упростио хардвер, тј, избегла дигитална логичка мрежа за одузимање.

Дакле, када су негативни бројеви записани применом првог комплемента, могуће је користити бинарно сабирање уз једноставне корекције. Корекције се састоје управо у додавању јединице (за коју се разликују први и други комплемент) резултату у неким случајевима. Потреба те корекције је уједно и највећи недостатак ове технике.

Други недостатак је постојање две нуле *позитивне* и *негативне*. Наиме, мењањем знака нули требало би да се поново добије нула, што у систему са првим комплементом није случај. Како се мењање знака (негирање) у овој техници своди на рачунање првог комплемента, полазећи од записа нуле (00000000_2) негирањем би се добио запис 11111111_2 што би требало да поново буде нула. Ова чињеница може донекле да отежа програмирање јер тестирање да ли је број негативан ако је резултат био негативна нула даје погрешну информацију.

Због описаних недостатака практично једина техника записа негативних целих бројева која се користи је техника примене другог комплемента. Коришћењем ове технике бинарно сабирање даје тачне резулте и за позитивне и за негативне означене бројеве без икаквих корекција. Поред тога, потпуно исто бинарно сабирање даје тачне резултате и за неозначене бројеве.

Техника коришћења другог комплемента за запис негативних бројева нема ни проблем двоструке нуле. Други комплемент осмобитног записа нуле (00000000_2) је ($1\ 00000000_2$) што је поново нула у осмобитном систему. Ово одсецање бита преноса (деветог бита у осмобитном систему), које је иначе "природно" и диктирано хардвером, управо омогућава да сабирање буде јединствено и за означене (било позитивне или негативне) и за неозначене бројеве.

Треба напоменути да је информација о одсеченом деветом биту доступна програмеру кроз заставицу (флег) за пренос **C** и програмер је може користити за тумачење добијеног резултата када је то потребно. За ту намену постоје одговарајуће машинске наредбе.

2.5.3. АНАЛИЗА РЕЗУЛТАТА САБИРАЊА – ЗАСТАВИЦЕ

После сваке аритметичке операције, процесор врши анализу добијеног резултата и поставља неколико заставица[1] (флегова) које у ствари показују

[1] Заставице су обрађене детаљније у поглављу 5.1.6, а у овом поглављу биће поменуте само оне које су у непосредној вези са начином представљања означених и неозначених података као и са операцијом бинарног сабирања.

стање аритметичко–логичке јединице после обављене аритметичке операције. Тих заставица има више и разликују се од процесора до процесора, али постоји неколико које су заступљене у скоро свим процесорима и чије су дефиниције готово јединствене. Заставице су доступне програмеру и њихово тумачење чини нераздвојни део сваке аритметичке операције. Програмер (у програму) на основу стања заставица може тумачити резултат на различите начине. Постоје наредбе које гранају програм зависно од стања сваке заставице појединачно или комбинације стања заставица, тако да зависно од тога какво је стање **ALU,** процесор може наставити да извршава различите наредбе, па чак и програме (подпрограме).

Све дефиниције заставица које следе дате су **под претпоставком осмобитне аритметичко-логичке јединице**. Ако ALU може да обавља и двобајтне или четворобајтне операције, за сваку дужину података постоји аналогна дефиниција заставице и стање заставице ће се одређивати различито за различите дужине. Заставице које су тренутно од интереса су следеће:

C (од ***C****arry*) која означава пренос из бита највеће тежине. Ова заставица *после сваке аритметичке операције* добија вредност или нула (уобичајен термин – *брише се* (**Clear**)) или један (уобичајен термин – *поставља се* (**Set**)).

- **C** ће бити **нула** ако у току аритметичке операције није дошло до преноса у девети бит (непостојећи девети бит би био нула).
- **C** ће бити **јединица** ако је у току аритметичке операције дошло до преноса у девети бит (непостојећи девети бит би био јединица).

N (од ***N****egative*) која означава негативан резултат под претпоставком да су улазни подаци означени бројеви. И ова заставица после сваке аритметичке операције добија вредност или нула или један. *Уколико се улазни подаци третирају као означени бројеви, важи следеће:*

- **N** ће бити **нула** ако је аритметичка операција дала резултат који је позитиван број,
- **N** ће бити **јединица** ако је аритметичка операција дала резултат који је број негативан.

V (од *o****V****erflow*) која означава да је резултат ван опсега означеног бајта (–128 до +127) под претпоставком да су улазни подаци означени бројеви. И ова заставица после сваке аритметичке операције добија вредност или нула или један.

- **V** ће бити **нула** ако је аритметичка операција дала резултат који је у опсегу –128 до +127 *уколико се улазни подаци третирају као означени бројеви.*

- **V** ће бити **јединица** ако је аритметичка операција дала резултат који је ван опсега –128 до +127 *уколико се улазни подаци третирају као означени бројеви.*

Посматрајмо два случаја сабирања у осмобитној ALU:

- У првом случају, саберимо бројеве 1 и 253. Резултат који очекујемо је свакако 254. Ово сабирање означићемо римским бројем **I**.
- У другом случају, саберимо бројеве 1 и –3. Резултат који очекујемо је свакако –2. Ово сабирање означићемо римским бројем **II**.

Посматрајмо сада бинарне кодове ова два сабирања:

- **I** случај

0000 0001	**1**
+1111 1101	**+253**
1111 1110	**254**

- **II** случај

0000 0001	**1**
+1111 1101	**-3**
1111 1110	**-2**

Ако погледамо бинарне облике података (које процесор једино разуме), приметићемо да су оба сабирања идентична, *како по улазним подацима, тако и по резултату*. Резултат је у оба случаја 1111 1110_2.

Да ли је то 254 или –2? Одговор је у оба случаја **ДА**.

Управо тако, резултат је и 254 и –2, како ко воли. Тачније, ако програмер **жели** да улазни подаци буду означени (1 и –3), резултат треба да **тумачи** као означен број (а тада је 1111 1110_2 заиста –2), а ако **жели** да улазни подаци буду неозначени (1 и 253), резултат треба да тумачи као неозначен број (а тада је 1111 1110_2 заиста 254). О програмеровим **жељама**, процесор нити има, нити може да има појма! Значи, једини који може да направи разлику између случаја I и II је **програмер**. Наравно у вишим програмским језицима о томе брине преводилац. Поставља се питање који ће се флегови сетовати у примеру I, а који у примеру II. Посматраћемо само заставицу прекорачења (V) и заставицу који показује да је резултат негативан (N заставица), остале нису проблематичне.

- I случај: Изгледа да има прекорачења (резултат 254 је ван опсега –128 до 127) и да би V заставица требало да буде сетована. Резултат је позитиван (+254), па изгледа да N заставица не би требало да буде постављеа. **Н Е Т А Ч Н О !**
- II случај: Изгледа да нема прекорачења (резултат –2 је у опсегу –128 до 127) и да би V заставица требало да буде обрисана.

Резултат је негативан (–2), па изгледа да би N заставица требало да буде постављена. **Т А Ч Н О !**

Међутим, тешко је очекивати од процесора, ма како веровали у његове способности, да од истих улазних података, који дају исти резултат, различито поставља заставице у зависности од жеље програмера. Због тога ће заставице V и N бити **увек** одређене уз претпоставку да програмер податке тумачи као означене, значи, **као у случају II**. Ако програмер ради са неозначеним бројевима, прекорачење може да провери тестирањем C заставице, а не тестирањем V заставице (V заставица тада нема никакав смисао). Ако ради са означеним подацима, V заставица је та који носи информацију о прекорачењу, а не C заставица. Информација о томе да је број негативан, када програмер ради само са позитивним бројевима није потребна, јер ни један податак програмер неће тумачити као негативан број. Негативан резултат после одузимања два неозначена броја, биће назначен прекорачењем (опет помоћу C заставице).

Ово разматрање важи искључиво за осмобитни запис.

Све улазне податке (као и излазни) би требало тумачити на исти начин, или као означене или као неозначене. Комбинације означених и неозначених улазних података дају резултате које је тешко интерпретирати и такво програмирање носи велику опасност од грешака које могу бити веома непријатне и проблематичне за откривање.

Сабирање ради потпуно исто и за означене и неозначене податке и даје тачан **означени** резултат када су улазни подаци **означени**, а тачан **неозначени** резултат када су улазни подаци **неозначени**. Ово важи док су и улазни и излазни подаци у дозвољеном опсегу.

2.5.4. БРОЈЕВИ СА НЕПОКРЕТНОМ ЗАПЕТОМ (СА НЕПОКРЕТНОМ ТАЧКОМ)

Снагу једног рачунара у многоме одређују и бројеви над којима се може вршити обрада. На избор врсте бројева које ћемо у програму користити утичу следећи чиниоци:

1. *врсте бројева који се могу приказати у рачунару, тј. цели, реални и комплексни бројеви,*
2. *опсег вредности које ти бројеви могу имати,*
3. *тачност приказивања бројева,*
4. *цена хардвера који је потребан за памћење и обраду бројева.*

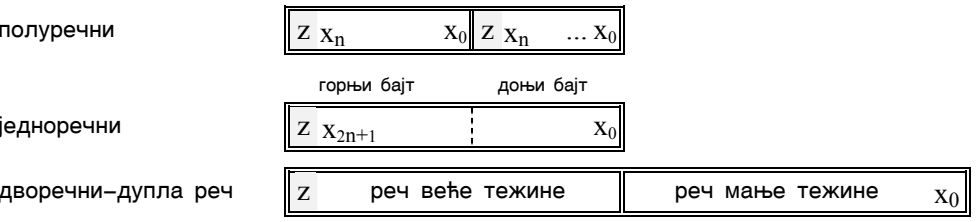
Ми смо до сада разматрали само како се у рачунару приказују цели бинарни бројеви, али и децимални (реални) бројеви се врло често користе пре свега за приказивање врло малих бројева (чија је апсолутна вредност око нуле) и врло великих бројева.

Нека је бинарни број **x** дат у облику: $x = x_p\, x_{p-1} \ldots x_1\, x_0 \bullet x_{-1}\, x_{-2} \ldots x_{-k}$, где су x_i, $i = -к....-2,-1,0,1,....., (p-1)$ бинарне цифре броја x, а x_p знак броја. Свака бинарна цифра има одређено место у машинској речи, а то место се зове **разред** или **ћелија**. Према томе, бинарни бројеви у рачунару имају коначну дужину. Број x је представљен у непокретном зарезу тако што се положај зареза (или децималне тачке) утврђује на одређеном месту у односу на разреде броја и остаје неизмењен за све бројеве приказане машинском речи.

Ако се запета утврди иза крајње левог разреда који је у машинској речи означен као позиција бита највеће тежине (иза **MSB**), онда су сви меморисани бројеви по модулу мањи од јединице.

Ако се запета фиксира иза позиције разреда који садржи бит најмање тежине онда су сви меморисани бројеви из скупа целих бројева, тј. цели бројеви су посебан случај бројева у непокретном зарезу. Све што је речено о целим бројевима односи се и на бројеве са непокретном запетом (тачком).

Зависно од тога колико машинских речи се користи за представљање нумеричких података, бројеви непокретном тачком се могу представити као:



Дужина броја непосредно одређује тачност његовог приказивања (број значајних цифара), али и потребан меморијски простор за његово записивање. Без обзира на дужину записа у бинарну позицију (*ћелију–разред*) највеће тежине (**MSB**) увек се мора меморисати и знак.

Полуречни формат омогућава смештање два бинарна броја (са знаком) у један регистар у меморији. Служи најчешће за запис целих бројева.

Једноречни формат омогућава смештање једног нумеричког податка у један регистар у меморији.

Дворечни формат омогућава запис једног нумеричког податка у два регистра при чему је у првој речи бинарна позиција највеће важности резервисана за знак читавог броја, а друга реч представља неозначен нумерички податак.

Записивање разломљених, нецелих, бројева помоћу фиксне запете је техника која се пуно примењује нарочито у рачунарима где је брзина обављања аритметичких операција од пресудне важности. Осим у рачунарима чији процесори раде директно са бројевима у покретном зарезу, или имају посе–

бан хардверски склоп (копроцесор) за подршку покретног зареза, све аритметичке функције се обављају користећи целобројну аритметику.

Основни принцип се састоји у томе да се разломљени број **замени целим бројем** тако што ће се помножити константом облика 2^N. Када се то уради, целокупна аритметика се обавља користећи бинарне аритметичке операције аритметичко–логичке јединице као да се ради о целим бројевима. Дакле, све аритметичке операције обавља хардвер и нису потребни посебни програми као што је то случај у покретном зарезу.

2.5.4.1. Неозначени бројеви представљени помоћу фиксне запете

Посматрајмо најпре само неозначене (позитивне) бројеве. Како би у бинарни облик претворили број **5,5**? То је број $\mathbf{101,1_2}$ али како овај бинарни број записати? Како би његов запис изгледао у меморији, на пример, осмобитног рачунара? Децимални зарез је, на жалост немогуће записати у меморији. Преостаје једино могућност да се запишу само бинарне цифре а да се информација о положају децималног зареза сачува на неки други начин. Када се ради о запису разломљених децималних бројева помоћу фиксног зареза, информацију о положају децималног зареза чува програмер и он је тај који води рачуна о положају зареза у резултату.

Да поновимо, у запису помоћу фиксног зареза информацију о положају зареза чува програмер, а зарез се не уписује у меморију (тачније, не уписује се његов положај).

Дужина бинарног записа (број бита) и положај децималног зареза дефинишу **формат записа**. У примеру броја 5,5 **формат записа** овог броја одређен је подацима да је запис осмобитни и да је зарез између бита 0 и бита 1. Формат је информација која нужно иде уз сваки податак појединачно када се користи запис помоћу фиксног зареза. Информацију о формату чува програмер. Дакле, у осмобитном запису број **5,5** би изгледао $\mathbf{00001011_2}$, а програмер би морао да памти да се децимални зарез налази између бита 0 и бита 1.

Приметимо на овом месту да је поменути бинарни број (00001011_2) у ствари бинарни код броја 11 и да једино програмер може да зна да ли је записани број 5,5 или 11. Број 11 је двострука вредност броја 5,5 и програмер све време мора водити рачуна о томе да ће у аритметичким операцијама учествовати број 11, а не 5,5. После обављених аритметичких операција, програмер мора да коригује резултат.

На пример ако треба број 5,5 помножити са 3, рачунар ће множити 00001011_2 (што је у ствари 11) са 3 и добити 33. Програмер мора знати да је први улазни податак у формату фиксног зареза са зарезом између бита нула и бита 1 (што практично значи да је увећан 2 пута) и да је сходно

томе и резултат у истом формату, са децималним зарезом између бита 0 и бита 1 па је и он два пута већи.

Да би се програмеру олакшало памћење формата у коме су поједини подаци записани, уведене су стандардизоване ознаке за ознаку положаја децималног зареза (формата). Такође су дефинисана формална правила која одређују у ком формату је резултат, ако се знају формати улазних података. За ознаку формата користи се облик:

$$Q_{N\text{-}R,\ R}$$

При том је N-R број бинарних цифара испред децималног зареза, а R број бинарних цифара иза децималног зараза. Број R се назива "**radix**" и тај термин је одомаћен и у литератури на српском језику. N одређује дужину бинарног записа (укупан број бита).

Када је $R = 0$, запис је целобројни, а када је $R = N$, за запис се каже да је нормализован. Помоћу таквог записа се могу приказати позитивни бројеви мањи од јединице ($0 \leq$ **број** < 1).

Правила која одређују формате резултата за сабирање и одузимање своде се на то да се могу сабирати или одузимати једино бројеви у истом формату и да је резултат који се добије поново у том формату.

Формат резултата множења и дељења је одређен следећим дефиницијама:

Број у формату $Q_{A,B}$ • број у формату $Q_{C,D}$ добија се број у формату $Q_{A+C,B+D}$

Број у формату $Q_{A,B}$ / број у формату $Q_{C,D}$ добија се број у формату $Q_{A-C,B-D}$

Проблем евентуалних негативних вредности радикса (A-C) или броја бинарних цифара пре зареза (B-D) (насталих код дељења) решава се једноставном корекцијом. На пример, ако се после дељења добио формат резултата $Q_{-1,9}$ он се може посматрати као да је број у формату $Q_{0,8}$ само га треба помножити са 2^{-1}. Детаљи везани за формате приликом дељења превазилазе ниво овог курса.

На питање који је осмобитни бинарни запис броја 5,5 из примера, прецизан и потпун одговор би био:

5,5 = 00001011$_2$ у формату $Q_{7,1}$

Из примера се види да сам осмобитни бинарни запис (без назнаке формата) није довољан јер исти бинарни запис може да одговара великом броју бројева. Већ смо видели да то може бити број 11 ако је цео број (у формату $Q_{8,0}$), а може одговарати и броју 1.375 ако је записан у формату $Q_{6,2}$. Следећи пример приказује неколико различитих вредности које могу одговарати истом запису, као и више различитих записа исте вредности:

00001011$_2$ може бити:

11	у формату $Q_{8,0}$
5,5 ($11/2^1$)	у формату $Q_{7,1}$
2,75 ($11/2^2$)	у формату $Q_{6,2}$
1,375 ($11/2^3$)	у формату $Q_{5,3}$
0,6875 ($11/2^4$)	у формату $Q_{4,4}$
0,34375 ($11/2^5$)	у формату $Q_{3,5}$

С друге стране, исти број **5,5 може се записати као**:

00001011_2	у формату $Q_{7,1}$
00010110_2	у формату $Q_{6,2}$
00101100_2	у формату $Q_{5,3}$
01011000_2	у формату $Q_{4,4}$
10110000_2	у формату $Q_{3,5}$

Из примера се може уочити да се исти број може написати у више формата а да се претварање из једног формата у други који има радикс (број бинарних цифара иза запете, односно децималне тачке) за један већи, врши тако што се бинарни запис помери за једно место улево (што одговара множењу са 2). Даље повећавање радикса после формата $Q_{3,5}$ би довело до губитка информације (јединица на месту најзначајнијег бита би отпала). Треба напоменути да говоримо искључиво о неозначеним бројевима, Формат $Q_{3,5}$ се не би смео користити да је реч о означеним бројевима (јер би у том случају 10110000_2 био негативан број).

Форматом је одређен број цифара испред зареза и број цифара иза зареза.

- Број цифара испред зареза дефинише опсег бројева који се могу тим форматом представити.
- Број цифара иза зареза дефинише тачност са којом се бројеви приказују.

Збир ова два броја (број бинарних цифара испред и иза) је једнак дужини записа која је најчешће фиксна и одређена хардвером, тако да је програмер при одређивању формата у коме ће записати неки променљиви податак принуђен да тргује између што веће тачности и што већег опсега. Приликом аритметичких операција програмер може да мења формате истог податка у жељи да повећа тачност или опсег. Операција мењања формата која се своди на множење или дељење са 2^N, тачније на померање (шифтовање) податка за N места улево или удесно, је нешто што се стално примењује готово при свакој сложенијој аритметичкој операцији.

Када су у питању константни подаци, увек постоји један оптимални формат. То је онај који има довољан број бинарних цифара испред зареза да се запише целобројни део константе. Тако иза зареза остају сви битови до задате дужине записа чиме се омогућава максимална тачност. На пример: **3,6 = 11,1001100110011001...$_2$**

Очигледно је да је за запис тачне вредности ове константе потребан бесконачан број бинарних цифара. Како је број бинарних цифара записа ограничен (рецимо на 8), треба покушати да што већи број бинарних цифара посветимо делу иза зареза како би остварили што већу тачност. Међутим, да би записали целобројни део неопходна су два бита па је број бита испред зареза минимално 2. Дакле, оптимални запис ове константе био би у формату $Q_{2,6}$. Нормално, у обзир долазе и формати $Q_{3,5}$, $Q_{4,4}$, $Q_{5,3}$... али је у њима тачност представљања све мања и мања.

3,6 = 11100110$_2$ у формату $Q_{2,6}$.

Да би били до краја прецизни, ово је најприближнији запис броју 3,6. Број који смо записали није 3,6 већ је:

3,59375 (11,100110= 2+1+0.5+0+0+0.0625+0.03125+0).

Ако би целобројни део био већи, на пример тражимо запис броја 65,6 тада би запис морао да има чак седам бита испред зареза, дакле био би нужан формат $Q_{7,1}$. Број 65,6 = 10000011$_2$ у формату $Q_{7,1}$. Уместо са 65,6 тада би рачунар располагао податком 1000001,1$_2$ што је 65,5, али то је најбоље што можемо добити у осмобитном запису.

Са константама, одређивање оптималног формата је прилично јасно. Питање је који формат усвојити за неку променљиву која се, на пример, мења у границама од 3 до 129. Очигледно је да би било нужно оставити довољно бита испред зареза за запис максималне вредности $Q_{7,1}$, али тада је тачност записа мала када променљива добије вредности блиске доњој граници. То се у пракси решава сталним мењањем формата што нас доводи до потребе за увођењем формата покретног зареза где сваки податак у меморији садржи и запис о положају зареза и све аритметичке операције се обављају узимајући у обзир обе информације.

Општи облик **осмобитног записа неозначеног броја у формату $Q_{8\text{-}R,\,R}$** је

$$2^{-R}\bullet(b_7\bullet 128 + b_6\bullet 64 + \ldots + b_1\bullet 2 + b_0)$$

дакле, разликује се од општег облика осмобитног целог броја једино по томе што је помножен константом 2^{-R}, (или подељен константом 2^R). На примеру броја 3,6 у формату $Q_{2,6}$ може се показати значење овог општег облика. У овом случају радикс (R) је 6 па је:

3,6 = (1 1 0 0 1 1 0)$_2$ у формату $Q_{2,6}$.

$$3{,}6 = (1\bullet 128 + 1\bullet 64 + 1\bullet 32 + 0\bullet 16 + 0\bullet 8 + 1\bullet 4 + 1\bullet 2 + 0\bullet 1)\bullet 2^{-6}$$

То би значило да "допринос" најзначајнијег бита више није 128 већ 2 ($128/2^6$) и тако са сваким битом.

Целокупно разматрање односило се искључиво на неозначене бројеве.

2.5.4.2. Означени бројеви представљени помоћу фиксног зареза

Представљање означених бројева у фиксном зарезу се у принципу не разликује од представљања неозначених. Бројеви представљени помоћу фиксног зареза су у ствари цели бројеви, па све што важи за означене целе бројеве важи и за означене бројеве представљене помоћу фиксног зареза. Да би се назначило да је у питању означени број, неки аутори обележавају формате означених бројева симболом Q^s. Тако ознака формата $Q^s_{3,5}$ значи да се ради о означеном броју записаном у формату $Q_{3,5}$.

Када је у питању одређивање оптималног формата константе, у случају означених бројева треба водити рачуна о томе да минимална дужина записа означеног броја захтева један бит више у односу на минималну дужину записа неозначеног броја исте апсолутне вредности. Тако је у формату $Q^s_{3,5}$, максимална целобројна вредност означених бројева је 3, а не 7 као што је то случај код неозначених. На примеру овог формата показаћемо опсег означених и неозначених бројева

$Q^s_{3,5}$	од	– 4	(10000000_2)	до	3,96875 (01111111_2)	ОЗНАЧЕНИ
$Q_{3,5}$	од	0	(00000000_2)	до	7,96875 (11111111_2)	НЕОЗНАЧЕНИ

Табела 3. показује најмањи и највећи (по апсолутној вредности) позитивни и негативни број различит од нуле. Табела се односи на формат $Q^s_{3,5}$ означених бројева:

НЕГАТИВНИ		ПОЗИТИВНИ	
најмањи по апс. вредности	највећи по апс. вредности	најмањи по апс. вредности	највећи по апс. вредности
– 0,03125 (11111111_2)	– 4 (10000000_2)	0,03125 (00000001_2)	3,96875 (01111111_2)

Табела 3. Најмањи и највећи позитивни и негативни у формату $Q^s_{3,5}$

Из ових опсега произилази да оптимални запис константе 3,6 из претходног примера неће моћи да буде у формату $Q_{2,6}$ ако хоћемо да буде записана као означен број, јер би запис у овом формату (11100110_2) у случају означених бројева представљао негативан број.

3,6 = 11100110_2 у формату $Q_{2,6}$ као НЕОЗНАЧЕНИ БРОЈ

3,6 = 01110011_2 у формату $Q^s_{3,5}$ као ОЗНАЧЕНИ БРОЈ

Да напоменемо још једном, позитивни бројеви се могу представљати и као означени (ако треба да учествују у операцијама са другим бројевима који могу бити негативни) и као неозначени (ако су сви учесници у свим аритметичким операцијама искључиво позитивни бројеви).

Општи облик **осмобитног записа означеног броја у формату** $Q^s_{8\text{-}R,\,R}$ је

$$2^{-R}\bullet(-b_7\bullet 128 + b_6\bullet 64 + \ldots + b_1\bullet 2 + b_0)$$

дакле, разликује се од општег облика означеног осмобитног целог броја једино по томе што је помножен константом $\mathbf{2^{-R}}$, (или подељен константом $\mathbf{2^{R}}$). На примеру броја 3,6 у формату $Q^{s}_{3,5}$ може се показати значење овог општег облика. У овом случају радикс (R) је 5 па је:

$$\mathbf{3,6 = 0\quad 1\quad 1\quad 1\quad 0\quad 0\quad 1\quad 1_2} \text{ у формату } Q^{s}_{3,5}$$

$$\mathbf{3,6 = (-0 \cdot 128 + 1 \cdot 64 + 1 \cdot 32 + 1 \cdot 16 + 0 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 1 \cdot 1) \cdot 2^{-5}}$$

2.5.4.3. Најчешће коришћени формати и примери

У пракси се користе различити формати за запис бројева у покретном зарезу. Осмобитни запис се за већину примена показује као недовољан па се много чешће користи двобајтни или четворобајтни запис. Како је стално конвертовање формата мукотрпан посао и како увек постоји ризик да је усвојени формат недовољан да прихвати резултат, то се често прибегава методу да се сви подаци нормализују. Тачније, бројеви се представе као величине између нуле и јединице по апсолутној вредности тако да се што више аритметичких операција обави користећи тај формат. То чини да се формат $\boldsymbol{Q_{0,16}}$ за неозначене бројеве, и формат $\boldsymbol{Q^{s}_{1,15}}$ за означене, користе нешто више од других формата.

У формату $\boldsymbol{Q_{0,16}}$ за неозначене бројеве, најмањи број различит од нуле је 1/65536 што је око $1{,}53 \cdot 10^{-5}$ док је највећи број за толико мањи од јединице (око 0,9999847). У формату $\boldsymbol{Q^{s}_{1,15}}$ за означене бројеве, најмањи позитивни број различит од нуле је 1/32768 што је око $3{,}05 \cdot 10^{-5}$ док је највећи позитивни број око 0,9999695. Најмањи негативни број по апсолутној вредности је –1/32768 (око $-3{,}05 \cdot 10^{-5}$) док је по апсолутној вредности, највећи негативни број тачно –1.

1. Како у осмобитном запису записати број π ($3{,}1416_{10}$)?

Претварањем у бинарни запис $3{,}1416_{10} = 11{,}00100100001111\ldots_2$ Пре но што одредимо који би формат записа био оптималан, морамо знати да ли желимо да овај број запишемо као означен или као неозначен број. Оптимални запис је следећи:

$3{,}1416 = 11001001_2$ у формату $Q_{2,6}$ као НЕОЗНАЧЕНИ БРОЈ

$3{,}1416 = 01100100_2$ у формату $Q^{s}_{3,5}$ као ОЗНАЧЕНИ БРОЈ

Треба приметити да је број са којим ће рачунар радити ако се користи формат $Q^{s}_{3,5}$ у ствари 3,125, али то је најприближније што може да се добије помоћу осмобитног записа. Код већих константи, грешке су у принципу још веће јер остаје мањи број бинарних цифара иза зареза.

2. Како у осмобитном запису записати број $-\pi$ ($-3{,}1416$)?

Ако пођемо од апсолутне вредности (X) негативног броја $(-X)$, и ту апсолутну вредност запишемо у формату означеног броја, други комплемент тог броја ће бити бинарни запис броја $-X$ Претварањем у бинарни запис апсолутне вредности 3,1416 = 11,00100100001111…$_2$. Негативни број се може представити само као означен па ћемо зато и ову апсолутну вредност записати у формату означених бројева:

3,1416 = 01100100$_2$ у формату $Q^s_{3,5}$ као ОЗНАЧЕНИ БРОЈ

Први комплемент овог броја је 10011011$_2$ а други комплемент се добија додавањем јединице 10011100$_2$. Према томе,

–3,1416 = 10011100$_2$ формату $Q^s_{3,5}$

Други начин за добијање бинарног записа негативног броја је да се пронађе еквивалентни неозначени број који има исти код. Тај број је за **осмобитни запис** 256–3,1416=252.8584. Код за овај неозначени број се може наћи на већ описани начин:

КОД ЗА ОЗНАЧЕНИ –3,1416=**КОД ЗА НЕОЗНАЧЕНИ** 252.8584=10011100$_2$ формату $Q^s_{3,5}$

Овај принцип је објашњен у примерима претварања негативних целих бројева, а важи и за разломљене записане помоћу фиксног зареза јер су они у ствари записани као цели бројеви. Ако би се тражио шеснаестобитни запис, одговарајући неозначени број био би 65536–3,1416.

3. *Који хексадецимални број се налази у меморији шеснаестобитног рачунара као запис константи 0.75$_{10}$ и –0.75$_{10}$ ако се зна да су записане као означени бројеви у формату $Q^s_{1,15}$?*

Претварањем у бинарни запис 0,75$_{10}$ = 0,11$_2$. Према томе,

0,75$_{10}$ = 0110 0000 0000 0000$_2$ у формату $Q^s_{1,15}$.

То значи да хексадецимални број **6000$_{16}$** представља запис броја 0,75 у формату $Q^s_{1,15}$. Запис броја –0,75 се добија рачунањем другог комплемента записа броја 0,75 у истом формату. Први комплемент је 1001 1111 1111 1111$_2$ па је други комплемент 1010 0000 0000 0000$_2$. Према томе број је:

–0,75$_{10}$ = 1010 0000 0000 0000$_2$ у формату $Q^s_{1,15}$.

То значи да хексадецимални број **A000$_{16}$** представља запис броја –0,75 у формату $Q^s_{1,15}$.

2.5.5. ПРЕДСТАВЉАЊЕ БРОЈЕВА СА ПОКРЕТНОМ ТАЧКОМ

Многе величине у природи и друштву могу имати вредности у врло широком опсегу, односно могу бити врло мале и врло велике (на пример, растојање две тачке може бити од неколико милиметара до неколико милиона километара). Такве величине најбоље је приказивати у *експоненцијалном облику*.

$$+25 = +2.5\times10^{+1};$$
$$-0.005 = -5.00\times10^{-3} = -0.5\times10^{-2}...;$$
$$567 = 5.67\times10^{2} = +0.567\times10^{+3}...$$

Број **x** је представљен у *покретној тачки* (запети), односно у *експоненцијалном облику* ако је приказан у облику:

$$x = x_m S^{x_e} \quad \text{или} \quad x = x_m S^{x_{b-a}}$$

где је: x_m мантиса броја x, тј. број са знаком чија је вредност $1 \le |x_m| < 2$,

S основа карактеристике,

$\mathbf{x_e}$ експонент карактеристике (може бити позитиван и негативан),

$\mathbf{x_b}$ експонент са вишком **a**, $\mathbf{x_b = x_e + a}$, где је **a** вишак (ниво) експонента увек такав да је $\mathbf{x_b}$ увек позитиван број (број без знака),

Основа карактеристике **S** поклапа се са основом или целим степеном основе бројног система који се користи за представљање мантисе $\mathbf{x_m}$ тј. $\mathbf{S}=2^1$, 2^3, 2^4, тј. **S** је бинарни, октални или хексадецимални бројни систем, а обично је **S**=2 или $\mathbf{S}=16=2^4$. Ако је $\mathbf{S}=2^4$, онда промена експонента за ±1 одговара померању мантисе за четири места улево (за –1) или удесно (за +1).

Зависно од жељене тачности за представљање бројних података у покретном зарезу користи се: *стандардна*, *стандардна проширена, двострука и проширена двострука тачност*. Стандардна тачност се постиже када се за регистровање нумеричког податка у покретном зарезу користе 32 ћелије тј. 4 бајта. Најчешће је знак читавог броја у разреду највеће тежине (**MSB**), потом следи 8 разреда за експонент и 23 разреда за мантису. У овом случају експонент је у границама од –128 до +127, а како се најчешће користи експонент са вишком $\mathbf{a}=2^7=128$, експонент је од 0 до 255. На пример:

$$A = \text{-}419.8195_{10} \quad \Rightarrow \quad A = 110100011.1101_2.$$

Нормализована експоненцијална форма је:

$A = \text{-}1.1010001\,11101\times2^8$, $x_e = 8$, па 8–битни експонент са вишком износи $x_b = 8 + 128 = 136 = 10001000_2$.

z	експонент x_b							мантиса X_m																		
1	1	0	0	0	1	0	0	0	1	1	0	1	0	0	0	1	1	1	1	0	1	0	0	0	0	...	0

8 бита 23 бита

Ако је **S**=16, онда су то у декадном систему бројеви у опсегу 10^{-76} до 10^{+76}. Ово је, такозвана *хексадецимална нормализација*.

Најчешће бројеви у покретном зарезу имају, такозвана *нормализовану мантису:* $1 < \mathbf{x_m} < 2$, односно децимална тачка појављује се иза *прве значајне*

цифре (прва цифра слева различита од 0). Код бинарних бројева, децимална тачка појављује се иза прве јединице с леве стране у запису броја (MSB бит), што заправо значи да су мантисе увек облика:

$$1,x_{n-1}\,x_{n-2}\;\ldots\;x_1\,x_0.$$

Примери бинарних бројева:

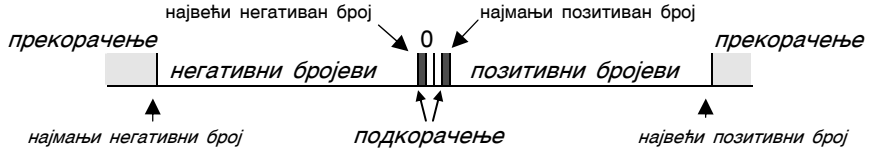
1. $-111 = -1.1100\times 2^{2}$ ⇨ $X_m = -1.1100 \quad x_e = 2$;
2. $0.001111_2 = +1.1110\times 2^{-3}$ ⇨ $X_m = +1.1110 \quad x_e = -3$;
3. $0.1_2 = +1.0000\times 2^{-1}$ ⇨ $X_m = +1.0000 \quad x_e = -1$;
4. $1010.1_2 = +1.0101\times 2^{3}$ ⇨ $X_m = +1.0101 \quad x_e = 3$.

То значи да се може изоставити прва јединица 1 па се један бит може додати за запис експонента. Како је за експонент у овом случају резервисано 8 ћелија то експонент броја у допунском коду може бити у опсегу:

$$-2^{7} < \mathbf{x}_e < 2^{7}-1.$$

Код стандардне тачности, за машинску реч дужине 16 бита за запис броја се користи дворечни формат, па мантиса садржи 6–7 тачних декадних цифара. Код проширене тачности (четвороречни формат) добија се приближно 14–16 тачних декадних цифара. По **IEEE** стандарду за запис бројева у двострукој тачности користи се 64 бита: 1 бит за знак броја, 11 битова за експонент и 52 бита за мантису.

Над бројевима у фиксном и покретном зарезу извршавају се основне аритметичке операције: сабирање, одузимање, множење и дељење, али се добијени резултати морају понекад додатно обрађивати. На пример, после операција множења и дељења треба извршити заокругљивање резултата и слично. Након извршења ових операција, као резултат се понекад могу појавити бројеви који су по апсолутној вредности већи од највећег броја (прекорачење–**overflow**), или мањи од најмањег дозвољеног броја (подкорачење–**underflow**), као што је то приказано на слици 2.1.



Слика 2.1. Скуп вредности бројева и могуће грешке

Појава прекорачења и подкорачења се аутоматски детектује у рачунару, и најчешће се прекида извршавање програма у којем се то десило.

Основне аритметичке операције се извршавају по следећим правилима:

Сабирање-одузимање:

1. *Одредити број са мањим експонентом и померити његову мантису удесно (код хексадецимално нормализованих бројева у корацима по 4 бита). Број корака једнак је разлици у експонентима бројева.*
2. *Поставити експонент резултата тако да је једнак већем експоненту.*
3. *Извршити сабирање–одузимање мантиса и одредити знак резултата.*
4. *Нормализовати резултат, по потреби, и користити 24 бита као мантису.*

Множење:

1. *Сабрати експоненте (и одузети* $\alpha = x_b$*, ако су експоненти са вишком).*
2. *Помножити мантисе и одредити знак резултата.*
3. *Нормализовати резултат, по потреби, и користити 24 бита као мантису.*

Дељење:

1. *Одузети експоненте (и додати* $\alpha = x_b$*, ако су експоненти са вишком).*
2. *Поделити мантисе и одредити знак резултата.*
3. *Нормализовати резултат, по потреби, и користити 24 бита као мантису.*

На крају поменимо и проблем представљања нуле. Наиме, нула се не може представити у покретном зарезу на уобичајен начин. Ако су мантиса и експонент једнаки нули, онда се оваква нула назива **права позитивна нула**. За случај представљања бројева у инверзном коду, постоји и **права негативна нула** која се добија инверзијом праве позитивне нуле**.** Позитиван број у покретном зарезу са мантисом једнаком нули и експонентом различитим од нуле назива се **позитивна абнормална нула.** Ако се ради о представљању негативних бројева у инверзном коду, онда је негативна нула абнормална ако се њеном инверзијом добија позитивна абнормална нула. При множењу и дељењу, абнормална нула се посматра обично као права нула. Осим наведених нула, постоје и **привидне нуле:**

$$S^{-2l-1} \quad \text{и} \quad -S^{-2l-1}$$

које представљају најмањи по модулу позитиван и негативан нормализован број различит од нуле.

2.5.6. БИНАРНО КОДИРАНИ ДЕКАДНИ БРОЈЕВИ

При претварању декадног у бинарни број може се користити метод сукце–сивних дељења (за цео део), и сукцесивних множења (за разломљени део). Тако добијени бинарни број се назива **природни бинарни код** декадног броја. При конвертовању разломљених декадних бројева у бинарне, због ограниченог броја битова увек се јављају одређене грешке, тј. непре–цизности. Додатна грешка настаје због тога што тачан рационалан декадни број најчешће нема тачан бинарни еквивалент, тј. поступак конверзије није коначан.

Ове непрецизности у приказивању бројева у бинарном бројном систему могу бити врло проблематичне у аутоматској обради података. Многе тачне методе користе се за приказивање рационалних декадних бројева, да би овај проблем био превазиђен. Заједничко за све методе је да се свака декадна цифра замењује еквивалентом од четири бинарне цифре, тј. са четири бита. Један од могућих начина кодирања дат је табели 4, а познат је као **бинарно кодирани декадни бројеви**, или **BCD** (**Binary Coded Decimal**).

Програм мора да ‘‘ зна ‘‘ или да ‘‘ запамти ‘‘ да су последње три цифре десно од децималне тачке. Коришћење кодова као што је **BCD** омогућава тачно приказивање рационалних декадних бројева, али се због њихове употребе у рачунарима јавља читав низ других проблема.

Посматрајмо бинарни број $01000111_{(2)}$. Како знати да ли он представља број $71_{(10)}$, $47_{(10)}$ или $14_{(10)}$? Први одговор подразумева да су ових осам бита нормалан бинарни број, други да се ради о бинарно кодираном декадном броју у коду ‘‘**8421**’’, а трећи да се ради о **BCD** броју у коду ‘‘**више 3**’’.

	Бинарни еквивалент	
декадна цифра	‘‘**8421**’’ **BCD** код	код ‘‘ **више 3** ‘‘
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

Табела 4. Приказ декадних цифара бинарним еквивалентом у **BCD** коду

По овом поступку број $29.375_{(10)}$ приказује се бинарним еквивалентом:

0010	1001	0011	0111	0101	код **8421**
0101	1100	0110	1010	1000	код **више 3**
2	9	3	7	5	декадна цифра

Ком декадном броју одговара бинарни низ $10010110_{(2)}$? То може бити $150_{(10)}$ (природни бинарни број без знака), $96_{(10)}$ (8421 BCD), $63_{(10)}$ (више 3 BCD код), $-105_{(10)}$ (први комплемент), или $-106_{(10)}$ (други комплемент). Многи рачунари не поседују способност да разликују ове могућности. Програмер мора да води рачуна о томе да коришћени бинарни бројеви буду коректно интерпретирани и обрађени.

BCD бројеви имају још неколико значајних проблема. Један је питање искоришћености простора бројева. **BCD** цифре користе само 10 од 16 могућих комбинација са четири бита, па је ефикасност коришћења меморије мања. Други проблем се тиче аритметике. Сабирање и одузимање **BCD**

бројева не може се извести у простој бинарној аритметици, као што је приказано у примеру где се врши сабирање бројева 2876_{10} и 6943_{10} који су представњени у **BCD** коду 8421.

```
пренос између тетрада   0    1    0    0
          2875 =    0010 1000 0111 0101
          6943 =  + 0110 1001 0100 0011
                    1001 0001 1011 1000
```

У резултату се јављају следеће тетраде $1000_{(2)}$, $1011_{(2)}$, $1000_{(2)}$, и $1001_{(2)}$ од којих друга уопште није декадна цифра. Када **BCD** аритметика генерише код већи од $9_{(10)}$, резултат мора бити коригован тако да стварно представља декадну цифру. Тетрада $1011_{(2)}$ је број $11_{(10)}$, и мора бити конвертована у број $0001_{(2)}$ ($1_{(10)}$), са преносом у следећи виши разред. Исто тако између треће и четврте тетраде се појавио пренос па и овај резултат мора бити коригован.

Ово подешавање може извести хардвер или софтвер. Неки рачунари имају посебне **BCD** аритметичке инструкције, а неки посебне инструкције за подешавање резултата. Подешавање резултата се састоји у томе да се на непостојећу комбинацију и на тетраду у којој се појавио пренос у следећу тетраду дода број 0110_2, односно 6_{10}. Не обазирући се на средства, **BCD** аритметика захтева додатне кораке и обично је спорија од бинарне аритметике. Цена којом су плаћени тачни декадни разломљени бројеви јесте мања ефикасност коришћења меморије и спорија обрада. Практично сабирање BCD бројева реализује се у више корака:

1. У првом кораку се врши сабирање BCD бројева по правилима за бинарно сабирање (бит по бит не обазирући се на тетраде);
2. У другом кораку се врши корекција над тетрадама по следећим правилима:
   а) ако нема преноса у следећу тетраду и ако је добијени број у скупу дозвољених (мањи од 1010_2) не треба вршити корекцију;
   б) ако нема преноса у следећу тетраду, а добијена тетрада није у скупу дозвољених ($\geq 1010_2$), онда овој тетради треба додати број 6_{10} тј., 0110_2.
   в) ако постоји пренос у следећу тетраду онда овој тетради треба додати број 6_{10} тј., 0110_2.
3. Ако се у другом кораку појави једна или више тетрада у којима се јављају недозвољене комбинације, корекцију треба понављати све док све цифре задовоље услов да су мање од 1010_2.

Дакле, резултат добијен у првом кораку, коригујемо према наведеним правилима и добијамо коначан (и тачан) резултат:

$$\begin{array}{cccc} {}^{0}1001^{0} & 0001^{1} & 1011^{0} & 1000 \\ \underline{0000} & \underline{0110} & \underline{0110} & \underline{0000} \\ 1001 & 1000 & 0001 & 1000 \\ 9 & 8 & 1 & 8 \end{array}$$

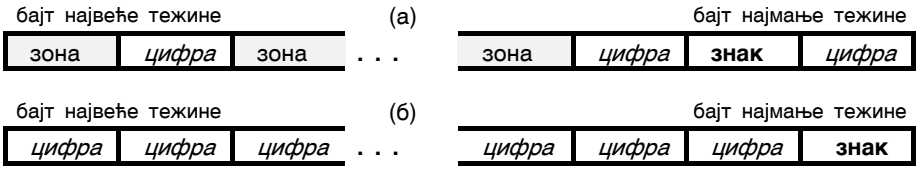
Један од често коришћених начина за записивање BCD бројева је, такозвани, код **више 3**. У овом коду се на 8421 код декадне цифре дода 0011_2, односно 3_{10}. Код овог кода резултат се увек добија након два бинарна сабирања, при чему се у првом сабирању (по правилима за бинарне бројева) одреди међурезултат, а другом кораку се врши корекција тако што се пренос између тетрада занемарује. Правила за корекцију су следећа:

1. ако код првог сабирања нема преноса у следећу тетраду, онда од те тетраде треба одузети три, односно треба јој додати 13_{10} тј., 1101_2 (ово је други комплемент цифре 3 у хексадецималном бројном систему);
2. ако се код првог сабирања појави јединица преноса из неке тетраде, онда тој тетради треба додати 3_{10}, тј., 0011_2.

Погледајмо употребу кода **више 3** на примеру сабирања декадних бројева 2875_{10} и 6943_{10}.

пренос између тетрада		0 1 1 0
2875 =		0101 1011 1010 1000
6943 =	+	1001 1100 0111 0110
међурезултат		1111 1000 0001 1110
међурезултат		1111 1000 0001 1110
корекција		1101 0011 0011 1101
коначан резултат у коду више 3		1100 1011 0100 1011
декадна вредност		9 8 1 8

BCD бројеви се смештају у меморију по принципу **једна цифра-један бајт**, и то је такозвани **распаковани облик**. При томе се у доњи полубајт записује **BCD** код цифре, а у горњи полубајт специјални код који се зове **зона**, па се овај начин памћења **BCD** бројева назива и **зонско паковање**. У последњи бајт (бајт најмање тежине) памти се цифра најмање тежине, а уместо зоне у том бајту памти се и знак броја, слика 2.2. (а). Као што се види готово половина меморијског простора остаје неискоришћена при зонском памћењу бројева.



Слика 2.2. Меморисање и обрада **BCD** бројева

Када се **BCD** бројеви обрађују у аритметичко–логичкој јединици, онда се пакују по две декадне цифре у један бајт, и то је **паковани облик**. У

пакованом облику сваки бајт садржи по две декадне цифре, а бајт најмање тежине садржи цифру и знак броја као на слици 2.2. (б). По завршетку обраде **BCD** бројеви се поново распакују и смештају у меморију.

2.6. КОДИРАЊЕ НЕНУМЕРИЧКИХ ПОДАТАКА

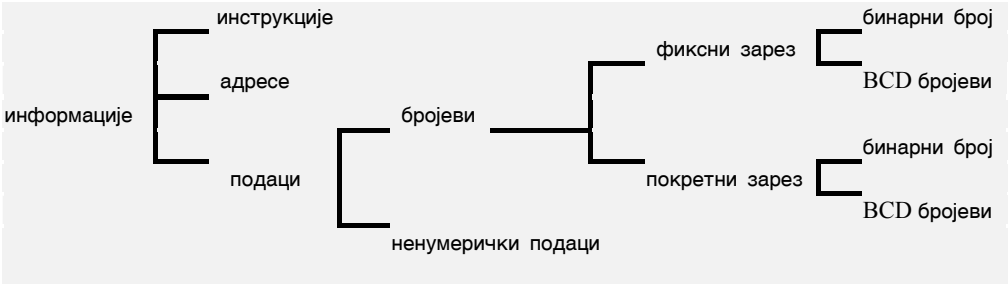
Рачунари морају поседовати могућност да складиште и обрађују како нумеричке (бројне), тако и ненумеричке тј. текстуалне податке (алфа нумерички подаци). Ненумерички подаци могу бити: **знакови** односно **карактери** (**character data**), међу којима су: *слова*, *знакови интерпункције*, *цифре* 0–9, *математички знакови*, *специјални знакови* и *контролне информације*, или **низови знакова**–*ниске* тј. *стрингови* (**string**). Подаци овог типа су меморисани на посебан начин у облику низова битова. За представљање знакова у почетку су се много користили шестобитни кодови што се касније показало недовољним, јер се са 6 битова могу приказати само 64 знака.

Два начина кодирања ненумеричких података који се најчешће користе у савременим рачунарима су: **ASCII (American Standard Code for Information Interchange)** и **EBCDIC** (**Extended Binary Coded Decimal Interchange Code**). EBCDIC код је осмобитни код који је развио **IBM**, и користе га само IBM–ови велики рачунари, и неки рачунари који су IBM компатибилни. ASCII је седмобитни код, широко распрострањен у комуникацијама између рачунара. Низ битова $1111000_{(2)}$ представља слово **x**, а низ $1111001_{(2)}$ означава **y**. Скоро сви микрорачунари користе овај код за приказ слова и симбола. Како и микрорачунари меморишу податке у осмобитним бајтовима, онда они додају један екстра бит на стандардни седмобитни ASCII код, да би се добио цео бајт. Оба кода ASCII и EBCDIC резервишу неке низове битова за контролне информације. Ови специјални кодови се користе за означавање почетка (**start**) и краја (**end**) неке поруке послате преко комуникационих линија или других средстава.

Оба ова кода имају посебне низове битова који означавају мала и велика слова, интерпункцију и друге специјалне карактере. **IBM-PC** компатибилни микрорачунари користе 8–битну варијацију **ASCII**–кода, где сваки нови низ битова (екстра код), означава неки графички карактер. Постојање ових кодних шема додатно компликује интерпретацију података који се налазе у меморији рачунара. Поменути низ битова $10010110_{(2)}$, поред већ поменутих значења ($150_{(10)}$, $96_{(10)}$, $63_{(10)}$, $-105_{(10)}$, $-106_{(10)}$), може такође представљати и мало слово **o** у рачунару који користи **EBCDIC** код.

У **ASCII** и **EBCDIC** табели се поред слова, бројева, специјалних знакова +, –, ? итд. налазе и **контролни знаци**. Ови контролни знаци не могу се видети на екрану монитора, нити се могу одштампати. Они служе за управљање радом улазно–излазних уређаја, затим при преносу информација између два рачунара итд. Табеле **ASCII** и **EBCDIC** кодова дате су у

Прилогу A. На крају можемо можемо рећи да се у рачунарима користе различити типови бинарно кодираних информација, чија се основна подела може приказати као на слици 2.3.



Слика 2.3. Основни типови информација у рачунарским системима

2.7. ЗАКЉУЧАК

Бинарни бројеви су основа за функционисање рачунара. Дигитална кола комбинују нуле и јединице, и генеришу нове нуле и јединице. Машинске инструкције и микро–програми се такође приказују као низови **0** и **1**. Сви програми написани у асем–блеру или било ком вишем језику, да би могли да раде, морају се превести у низове нула и јединица, односно у неке бинарне бројеве.

Бројеви унутар рачунара могу бити бинарни репрезенти позитивних и негативних величина, цели или разломљени бројеви, у фиксном или покретном зарезу. Они се такође могу користити као репрезенти симболичких информација и специјалних кодова. Коришћење бројева је од значаја на свим нивоима организације и рада рачунара. Сваки рачунар има свој скуп бројева које користи. Како унутар рачунара није могуће разликовати о ком типу бројева или кодова се ради, програмер мора о томе водити рачуна да не би дошло до погрешне интерпретације и грешака у обради.

2.8. ПИТАЊА

1. Која је основна разлика између римског и арапског бројног система?
2. Како основа бројног система утиче на вредност цифре?
3. Објаснити значај позиције цифре у запису броја у арапском бројном систему.
4. Како се декадни бројеви конвертују у бинарне, октални и хексадецималне?
5. Како се израчунава декадна вредност броја 10101101 ако је то: а) бинарни, б) октални или ц) хексадецимални број?
6. Које све методе постоје за представљање негативних бројева у рачунару?
7. Конвертовати бинарни број без знака –0110110 у први и други комплемент.
8. Које проблеме изазива коришћење директног кода тј. апсолутне вредности са знаком, и како се то превазилази помоћу комплемената?
9. Израчунати збир два бинарна броја без знака 10100010 и 11100111.

10. Израчунати разлику претходна два бинарна броја употребом првог и другог комплемента.
11. Како се одређује декадна вредност бинарних бројева у првом и другом комплементу?
12. Описати бројеве у непокретном зарезу.
13. Шта је прекорачење, а шта подкорачење?
14. Како се приказују бројеви у покретном зарезу?
15. Какве се све нуле могу појавити у рачунарском систему?
16. Конвертовати декадне бројеве 200 и 839 у BCD бројеве и сабрати их по правилима за сабирање природних бинарних бројева. Да ли је добијени резултат исправан?
17. Описати погодности и мане примене BCD бројева у поређењу са природним бинарним бројевима.
18. Да ли се код сабирања BCD бројева по правилима за бинарно сабирање резултат добија одмах? Шта треба урадити да би се добио тачан резултат?
19. Зашто је код **више 3** погоднији од кода **8421**?
20. Како се у рачунару записују подаци типа карактер?
21. Искористити табелу ASCII кодова за приказивање речи COMPUTER као низа бинарних бројева.

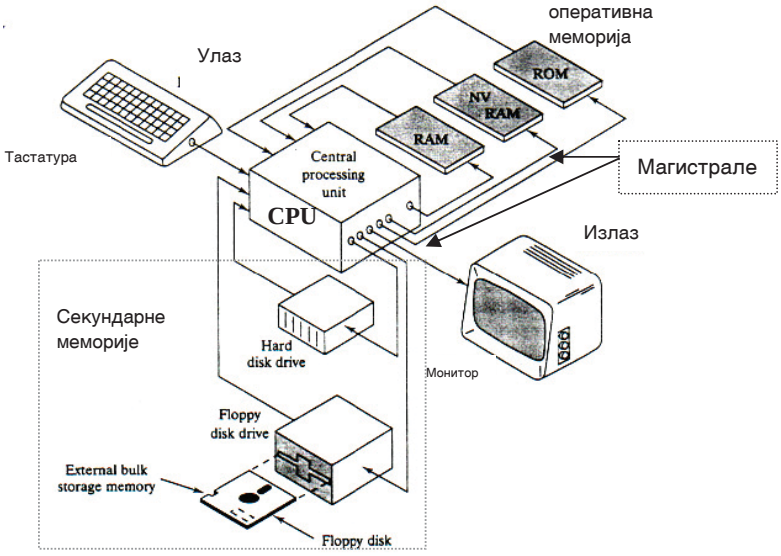
2.9. КЉУЧНЕ РЕЧИ:

- арапски бројни систем
- **ASCII** код (**American Standard Code For Information Interchange**)
- бајт (**byte**)
- база (основа) бројног система
- **BCD** бинарно кодирани декадни бројеви (**binary coded decimal**)
- бинарни бројни систем (**binary system**)
- бит (**bit**)
- децимална тачка (**decimal point**)
- декадни бројни систем (**decimal**)
- директни код
- други комплемент, допунски код, комплемент двојке (**two's complement**)
- **EBCDIC** код (**Extended Binary Coded Decimal Interchange Code**)
- хексадецимални број, бројни систем (**hexadecimal**)
- карактер, знак (**character**)
- **LSB** бит најмање тежине (**least significant bit**)
- **MSB** бит највеће тежине (**most significant bit**)
- негативни бројеви (**negative numbers**)
- неозначени бројеви (**unsigned numbers**)
- непозициони бројни систем
- нула, позитивна нула, негативна нула
- октални број, бројни систем (**octal**)
- позициони бројни систем
- пренос (**carry**)
- први комплемент, инверзни код, комплемент јединице (**one's complement**)
- реч (**word**)
- римски бројни систем
- стринг, ниска (**string**)
- тетрада, полубајт **(nibble**)
- циклично враћање преноса, ротација бита преноса (**end-around carry**

3. ЕЛЕКТРОНСКЕ ОСНОВЕ РАЧУНАРА

3.1. ДИГИТАЛНА ЛОГИКА

Хардвер рачунара се најчешће дели на неколико главних јединица као што је приказано на слици 3.1. Централна процесорска јединица (**CPU**) садржи у себи главна аритметичка, логичка и управљачка кола рачунара. **CPU** се логично дели на: управљачку јединицу која је одговорна за рад свих осталих саставних делова, аритметичко–логичку јединицу у којој се обављају све елементарне аритметичке, логичке и друге основне операције, и известан број специјалних меморијских склопова који се називају регистри.



Слика 3.1. Главне компоненте хардвера рачунара

CPU је са другим деловима рачунара спојен помоћу једне или више сабирница, тј. магистрала (**bus**), преко којих у току рада, различите врсте информација улазе или излазе из **CPU**–а. На магистрале су такође спојене: јединице главне (унутрашње) меморије, у којима се памте подаци и програми

који се управо користе, диск јединице за дуготрајно складиштење података и програма, као и велики број других улазних и излазних јединица. У срцу било ког рачунара налазе се дигитална кола која извршавају: управљачку, логичку, аритметичку и функцију памћења (складиштења). Ова кола налазе се у центру хијерархијског модела рачунарског система. Она омогућавају да се подаци и кодови пореде по једнакости или разним формама неједнакости. Она извршавају сабирање, одузимање и све логичке операције. Дигитална кола такође омогућују чување података за каснију употребу. Основа за рад дигиталних кола су логичке операције над бивалентним исказима тј. операције над исказима који могу имати само две истинитосне вредности: **тачан** (**true**) и **нетачан** (**false**). За ове две вредности има много синонима и примера у свакодневном животу: *позитивно–негативно*, *да–не*, *ниско–високо*, *истина–лаж, укључено–искључено*. Оваква стања се врло лако могу кодирати бинарним бројним системом, тј. помоћу **1** и **0**. Теоријске основе за бивалентну логику садржане су у делу математике познате под именом Булова алгебра (**George S. Boole**).

3.1.1. БУЛОВА АЛГЕБРА

Булова алгебра је дедуктивни математички систем који почива на аксиомама, на бази којих се даље развијају теореме. Заснива се на бинарним законима мишљења, где један исказ може бити или истинит (тачан) или неистинит (нетачан), а никада не може бити делимично тачан или делимично нетачан.

3.1.1.1 Аксиоме и теореме Булове алгебре

Нека је дат скуп $S = \{x, y, z, ...\}$ који садржи најмање два различита елемента, и нека су на овом скупу дефинисана два бинарна операнда са ознаком **+** (**логичко сабирање**, **ИЛИ**) и $\cdot$ (**логичко множење**, **И**), и један унарни операнд **–** (**негација**, **НЕ**). Булова алгебра садржи два специјална елемента **0** и **1**, таква да све променљиве **x, y, z, ...** узимају вредност из скупа {**0, 1**}. Да би овај скуп **S,** и операције +,– и $\cdot$ сачињавали Булову алгебру, неопходно је да буду задовољене аксиоме **Hantingtona**:

A-1 : Бинарне операције **+** и $\cdot$ су комутативне на скупу **S,** и међусобно су дистрибутивне тако да за свако **x, y, z,** који припадају скупу **S**, важи:

$$x + y = y + x \qquad x \cdot y = y \cdot x$$
$$x \cdot (y + z) = x \cdot y + x \cdot z \qquad x + (y \cdot z) = (x + y) \cdot (x + z).$$

A-2 : Бинарне операције **+** и $\cdot$ на скупу **S** поседују неутралне елементе **1** и **0,** тако да за свако **x** које припада скупу **S**, постоје елементи **1** и **0**, који такође припадају скупу **S**, тако да је:

$$x + 0 = 0 + x = x \qquad x \cdot 1 = 1 \cdot x = x.$$

A-3 : На скупу **S**, за свако **x** које припада скупу **S**, постоји јединствен инверзни елемент $\overline{x}$, који такође припада скупу **S**, такав да је :

	$x + \overline{x} = 1$ $\quad x \cdot \overline{x} = 0.$
T-1 Теорема идемпотентности:	$x + x = x$ $x \cdot x = x.$
T-2 Теорема о нултим елементима:	$x + 1 = 1$ $x \cdot 0 = 0.$
T-3 Теорема о инволуцији:	$\overline{(\overline{x})} = x$
T-4 Теорема о апсорпцији:	$x + x \cdot y = x$ $x \cdot (x + y) = x.$
T-5 Теорема о асоцијативности:	$x + (y + z) = (x + y) + z$
T-6 Де–Морганови закони:	$\overline{(x + y)} = \overline{x} \cdot \overline{y}$ $\overline{(x \cdot y)} = \overline{x} + \overline{y}.$

Напомена: Овде је важно уочити да у Буловој алгебри важи принцип дуалности, тј. све аксиоме и теореме су дате у пару, па све што важи за логичко множење важи и за логичко сабирање, само се **+** замени са **·** и **0** са **1**. Теореме **T-1**, **T-2,** а нарочито **T-4** указују на једну врло битну особину логичких функција да се сложене логичке функције могу минимизирати, тј. трансформисати у еквивалентне логичке функције исте истинитосне вредности, али знатно једноставнијег облика са мање саставних делова (мање променљивих и мање операција).

Напомена: Де–Морганови закони нам казују да се сложени логички искази негирају тако што се негира сваки исказ понаособ, али се негира и операција. На пример, негација исказа: *"Ако **сам** слободан **и** ако **је** лепо време, **ићи** ћу у шетњу*" је исказ: *"Ако **нисам** слободан **или** ако **није** лепо време, **нећу** ићи у шетњу*".

3.1.2 ОСНОВНЕ ЛОГИЧКЕ ОПЕРАЦИЈЕ НАД БИНАРНИМ ЦИФРАМА

Дигитална кола су пројектована тако да имплементирају принципе бинарне аритметике, Булове алгебре и бивалентне логике. Наиме, ова кола се могу наћи у једном од два стабилна стања, тако да се на њиховом излазу јавља или висок напонски сигнал (1) или низак (0). Логичка кола користе бинарне цифре **0** и **1** за представљање истинитосних вредности **нетачан** и **тачан**. Уобичајено је да се вредност **тачан** кодира као бинарна јединица, а **нетачан** као бинарна нула. Постоје две врсте логичких операција, зависно од броја операнада које у њима учествују, и то су:

– *унарне, логичке операције над једним операндом (негација),*

– *бинарне, логичке операције над два операнда (све друге операције).*

Операнди који учествују у логичким операцијама у рачунару могу бити:

– *логички подаци, где се вредности тачан и нетачан замењују специјалним низом бинарних цифара, тј. имају специјалан код,*
– *бинарни бројеви (вишецифрени) где се жељена логичка операција примењује на сваки бит одвојено, а резултат зависи само од садржаја то параг ботова (или пара ботова исте тежине – на истом месту у запису броја),и не утиче на резултат операције над бинарним цифрама на било ком другом месту у запису броја.*

Комбиновањем елементарних логичких операција, и њиховом применом на логичке променљиве, добија се велики број различитих логичких израза и функција. Ако имамо само две логичке променљиве (n=2), можемо направити 2^p ($p=2^n$), односно 16 функција.

3.1.2.1 Негација (NOT)

Најпростија логичка операција која се обавља над једном операндом зове се негација или **НЕ** операција (инверзија или комплементирање). Негација узима вредност тачан (1), и конвертује је у вредност нетачан (0) и обрнуто. На слици 3.2 је показана табела негације. X је улазна величина (операнд), а Z је излазна величина (резултат).

X	Z
0	**1**
1	**0**

$z = \overline{x}$

Слика 3.2. Табела истинитосних вредности негације

За израчунавање резултата логичких операција обично се користе табеле (као на слици 3.2.), које се зову табеле истинитости, комбинационе табеле или табеле стања. Ове табеле садрже све могуће вредности за улазне величине (X, Y, A, B,...), и одговарајући резултат, односно излазну вредност (Z, X, Y, ...). Број могућих комбинација улазних величина једнак је 2^n где је **n**–број улазних величина. Ако имамо једну улазну величину (негација), онда је број комбинација 2 (2^1), ако имамо две улазне величине број комбинација је 4 (2^2), за три је 8 (2^3) итд.

3.1.2.2 ИЛИ операција (OR)

Ова операција се врши над две или више улазних вредности, а назива се још и **логичко сабирање**, **дисјункција**. Да би резултат операције имао вредност **1** (тачан) мора бар једна улазна величина имати вредност **1** (**тачан**). На слици 3.3 је приказана таблица истинитости за **ИЛИ** операцију над две улазне вредности X и Y, као и таблица истинитости за **n** улазних вредности $X_1,..., X_n$. Уочавамо да комбинације X=1, Y=0 и X=0, Y=1 нису исте, али је резултат операције исти, тј. Z=1. Резултат Z=1 добија се када су једна или више улазних вредности једновремено једнаке 1.

X	Y	**Z**
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

z = x+y

X_1	X_2	...	X_{n-1}	X_n	**Z**
0	0	...	0	0	**0**
0	0	...	0	1	**1**
0	0	...	1	0	**1**
0	0	...	1	1	**1**
...	...	...	...	...	**...**
1	1	...	1	0	**1**
1	1	...	1	1	**1**

$$\mathbf{Z = X_1 + X_2 + ... + X_n}$$

воз са леве стране	**воз са десне стране**	**рампа спуштена**
не	*не*	***не***
не	*да*	***да***
да	*не*	***да***
да	да	**да**

Слика 3.3. Табела истинитости логичке операције **ИЛИ**

3.1.2.3 Операција И (AND)

Резултат ове операције је истинит (**1**), само ако су све улазне вредности такође истините. Другим речима, резултат операције **И** (**AND**) је једнак нули, ако је бар једна улазна вредност једнака нули. Операција **И** се још назива **логичко множење** или **конјункција**. Табела истинитости за две вредности X и Y, и за низ **n** улазних вредности $X_1,...X_n$ дата је на слици 3.4. Логичко множење даје резултат **тачан** само ако ни један улазни сигнал није једнак нули, тј. да би резултат био Z=1, морају сви улазни сигнали истовремено бити једнаки јединици: $X = Y = \mathbf{1}$ тј. $X_1 = X_2 = ... = X_n = \mathbf{1}$.

X	Y	**Z**
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

z = x · y

X_1	X_2	···	X_{n-1}	X_n	**Z**
0	0	...	0	0	**0**
0	0	...	0	1	**0**
0	0	...	1	0	**0**
0	0	...	1	1	**0**
.	.	.	.	.	**0**
1	1	...	0	1	**0**
1	1	..	1	0	**0**
1	1	...	1	1	**1**

$$\mathbf{Z = X_1 \cdot X_2 \cdots X_N}$$

имамо слободно време	***леп дан***	***идем у шетњу***
не	не	**не**
не	да	**не**
да	не	**не**
да	да	**да**

Слика 3.4. Табела истинитости логичке операције **И**

3.1.2.4 Ексклузивно ИЛИ (XOR)

Ова операција се назива још и **искључиво ИЛИ**, а даје истинит резултат (тачан, 1), ако је **једна и само једна** од улазних величина истинита. Табела истинитости операције **ексклузивно ИЛИ** дата је на слици 3.5. Ако пажљивије погледамо резултат ове операције, уочићемо да он одговара

збиру бинарних цифара (не узимајући у обзир пренос), па се зато ова операција назива и **сабирање по модулу два.**

x	y	z
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

$$z = x \oplus y$$

Слика 3.5. Таблица истинитости **ексклузивног ИЛИ** (**XOR**)

3.1.3 ЕЛЕМЕНТАРНА ЛОГИЧКА КОЛА

Основне логичке операције су: **НЕ, ИЛИ, И** и **ексклузивно ИЛИ**. Ове операције, да би генерисале резултат, следе правила математичке логике са само две вредности: **тачан** и **нетачан** (**1** и **0**). Електронске компоненте које извршавају логичке операције, изразе и функције називају се логичка кола. Стандардни симболи ових кола дати су на слици 3.6.

КОЛО	И	ИЛИ	НЕ	НИ	НИЛИ	ЕКСКЛУЗ. ИЛИ	компаратор
IEC							
DIN							
АМЕРИЧКИ.							
ФУНКЦИЈА	$X = AB$	$X = A + B$	$X = A$	$X = \overline{AB}$	$X = \overline{A + B}$	$X = \overline{A}B + A\overline{B}$	$X = AB + \overline{A}\overline{B}$

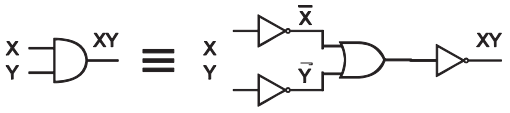
Слика 3.6. Основна логичка кола

Комбиновањем ових кола могу се реализовати произвољне логичке функције, као и све друге елементарне логичке операције. На слици 3.7 приказана је табела могућих логичких функција са две улазне величине, и све оне се могу приказати помоћу елементарних логичких операција **И**, **ИЛИ** и **НЕ**. Скуп операција помоћу којих се може реализовати свака друга логичка операција, и помоћу којих се може представити свака логичка функција, назива се **база** логичког система. У случају Булове алгебре скуп операција {**И**, **ИЛИ**, **НЕ**} представља базу. Но, операција **И** се може реализовати помоћу операције **НЕ** и **ИЛИ** (као што се то види на слици 3.8.), па скуп {**НЕ**, **ИЛИ**} такође чини базу логичког система. Такође се и операција **ИЛИ** може реализовати помоћу операција **И** и **НЕ** (слика 3.9.), па и скуп {**НЕ**, **И**} чини базу.

X	0	1	0	1	аналитички облик	НАЗИВ ФУНКЦИЈЕ
Y	0	0	1	1	функције	
F_0	0	0	0	0	0	константа нула
F_1	0	0	0	1	$X\,Y$	операција **И**, конјункција
F_2	0	0	1	0	$\overline{X}Y$	забрана по X
F_3	0	0	1	1	Y	променљива Y
F_4	0	1	0	0	$X\overline{Y}$	забрана по Y
F_5	0	1	0	1	X	променљива X
F_6	0	1	1	0	$X\overline{Y} + \overline{X}\cdot Y$	искључиво **ИЛИ**
F_7	0	1	1	1	$X + Y$	операција **ИЛИ**, дисјункција
F_8	1	0	0	0	$\overline{X}\cdot\overline{Y}$	Пирсова функција, операција **НИЛИ**
F_9	1	0	0	1	$\overline{X}\cdot\overline{Y} + X\cdot Y$	еквиваленција, компаратор
F_{10}	1	0	1	0	$\overline{X}$	негација X
F_{11}	1	0	1	1	$\overline{X} + Y$	импликација од X према Y
F_{12}	1	1	0	0	Y	негација Y
F_{13}	1	1	0	1	$X + \overline{Y}$	импликација од Y према X
F_{14}	1	1	1	0	$\overline{X} + \overline{Y}$	Шеферова функција, операција **НИ**
F_{15}	1	1	1	1	1	константа један

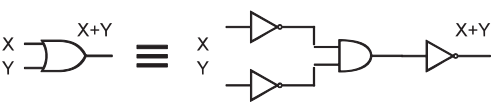
Слика 3.7. Табела функција са две променљиве

X	Y	$\overline{X}$	$\overline{Y}$	$\overline{X}+\overline{Y}$	$\overline{\overline{X}+\overline{Y}}$	XY
0	0	1	1	1	0	**0**
0	1	1	0	1	0	**0**
1	0	0	1	1	0	**0**
1	1	0	0	0	1	**1**



Слика 3.8. Реализација операције **И** помоћу операција **НЕ** и **ИЛИ**

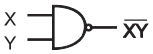
X	Y	$\overline{X}$	$\overline{Y}$	$\overline{X}\cdot\overline{Y}$	$\overline{\overline{X}\cdot\overline{Y}}$	X+Y
0	0	1	1	1	0	**0**
0	1	1	0	0	1	**1**
1	0	0	1	0	1	**1**
1	1	0	0	0	1	**1**



Слика 3.9. Реализација операције **ИЛИ** помоћу операција **И** и **НЕ**

Посматрајући табелу на слици 3.7. уочавамо две функције **НИ** и **НИЛИ**, које представљају негацију двеју основних логичких операција. Може се показати да је свака од ових операција (и **НИ** и **НИЛИ**), уједно база логичког система, тј. све друге операције се могу реализовати преко само једне од ове две функције. Употреба ових логичких кола пружа низ погодности у реализацији дигиталних мрежа у рачунарским склоповима и уређајима. На слици 3.10. дата је табела истинитости за **НИ** коло, као и њихов симбол, а на слици 3.11. дата је табела истинитости и симбол (који се користи у електричним шемама) за коло **НИЛИ**.

X	Y	X Y	**НИ**
0	0	0	**1**
0	1	0	**1**
1	0	0	**1**
1	1	1	**0**

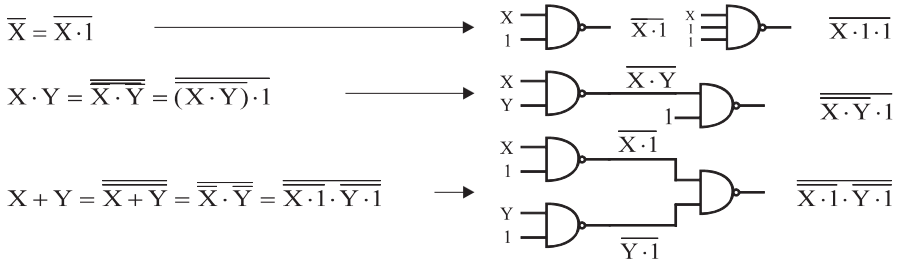


Слика 3.10. Табела истинитости и симбол **НИ** кола

X	Y	X+Y	**НИЛИ**
0	0	0	**1**
0	1	1	**0**
1	0	1	**0**
1	1	1	**0**

Слика 3.11. Табела истинитости и симбол **НИЛИ** кола

На слици 3.12. показано је како се помоћу **НИ** кола могу реализовати основне логичке операције **НЕ**, **ИЛИ** и **И**. На слици 3.13. приказана је реализација **НИ**, **ИЛИ** и **И** операција помоћу **НИЛИ** операције.



Слика 3.12. Реализација операција **НЕ, ИЛИ** и **И** помоћу **НИ** кола

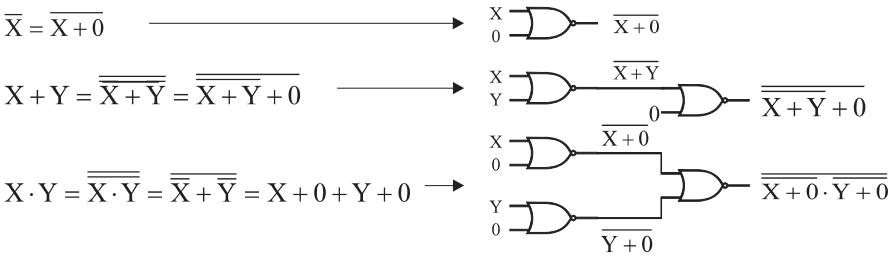
Са изузетком **НЕ** кола, сва поменута логичка кола су двоулазна. У рачунарској техници врло често се јавља потреба за применом **И** и **ИЛИ** операција над три, четири, осам, шеснаест и више улаза истовремено. Овај проблем се може решити двојако:

1. *употребом вишеулазних логичких кола,*
2. *повезивањем више двоулазних кола.*

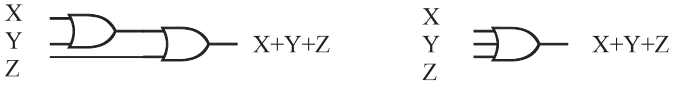
На слици 3.14. приказана је реализација **ИЛИ** кола са три улаза (троулазно коло), а на слици 3.15. реализација **И** кола за четири улаза.

Употреба стандардних логичких кола, а нарочито **НИ** и **НИЛИ** кола, умногоме поједностављује и појефтињује израду сложених логичких мрежа. Наиме логичка кола се никада не праве појединачно, већ се у једном интегрисаном колу (чипу, **chip**) обично налази неколико логичких кола исте

врсте. На пример, у једном чипу са 14 извода обично се налазе четири двоулазна логичка кола **И**, **ИЛИ, НИ**, или **НИЛИ**.



Слика 3.13. Реализација операција **НЕ, И** и **ИЛИ** помоћу **НИЛИ** кола



Слика 3.14. Реализација троулазног **ИЛИ** кола

Слика 3.15. Реализација четвороулазног **И** кола

Одређене логичке функције могу наћи и практичну примену у рачунарској техници. Реализација склопова који подржавају рад оваквих функција захтева низ електронских елемената. Што је већи број оператора у изразу за логичку функцију то је и њено извођење сложеније. Но, обзиром на то да се у изради једног рачунара користи од неколико стотина до неколико хиљада логичких функција, јасно је да је свака уштеда у том погледу од значаја. Да би се остариле уштеде у потрошњи логичких кола, неопходно је минимизирати све логичке функције које се јављају у једном рачунарском систему. Минимизација је неопходна да би се упростила електрична мрежа, и смањио број логичких кола потребних за реализацију. У овом случају то значи да функције треба приказати са што мање оператора и променљивих, а да при томе функција има исто значење, односно исти скуп вредности.

3.1.4 МИНИМИЗАЦИЈА ЛОГИЧКИХ ФУНКЦИЈА

Минимизација је поступак трансформације сложене логичке функције у функцију која има исту истинитосну вредност, али мањи број елемената, мање операнада и операција које над њима треба извршити.

Логичке функције могу се приказати на различите начине, а ми смо у досадашњем излагању већ користили три основна:

– *шематско приказивање помоћу логичких кола,*
– *табеларно, помоћу таблица истинитости,*
– *аналитичко, помоћу основних логичких операција.*

Све логичке функције аналитички се могу приказати у два облика:

– *дисјунктивна форма, која представља логичку суму логичких производа,*
– *конјунктивна форма, која представља логички производ логичких сума.*

Ако је функција приказана табеларно, може се одредити њен аналитички облик као дисјунктивна или конјунктивна форма, при чему се за одређивање дисјунктивне форме групишу елементи који одговарају јединици, тј. за које је вредност функције једнака логичкој јединици, док се за конјунктивну форму групишу елементи који одговарају нулама функције, као што је показано на слици 3.16. Мада постоје два стандардна начина овог превођења, овде ћемо размотрити само један од њих, а други се реализује по аналогији. Тај алгоритам дефинише логички израз функције у облику дисјунктивне нормалне форме (**DNF**), а састоји се од следећих корака:

а) *прво се за све вредности функције* $F=1$ *прави конјункција логичких променљивих и то самих променљивих уколико је њихова вредност у том случају једнака 1, односно негација променљивих ако је њихова вредност једнака 0. На пример, за број 3 је* $X_1=0$, $X_2=1$, $X_3=1$, *па је њој одговарајућа конјункција:* $\overline{X}_1 \cdot X_2 \cdot X_3$, *односно* $\overline{X}_1 X_2 X_3$.

б) *када су формиране све конјункције за вредности* $F=1$, *онда се оне повезују оператором дисјункције* (**+**) *и на тај начин се добија дисјунктивна нормална форма дате функције* F.

Очито је да дисјунктивна нормална форма даје веома сложен облик функције, па ћемо посебну пажњу посветити минимизацији функција датих у овом облику. Постоји више начина на које се једна логичка функција може минимизирати. Као прво, наведимо примену аксиома и теорема Булове алгебре. Тако нпр. ако се у једној функцији налази израз $\mathbf{x+0}$, он се на основу аксиоме **A-2** ($x+0=x$) може заменити само са променљивом $\mathbf{x}$, чиме се сложеност функције смањује за један оператор. Исто тако, ако је у функцији дат израз облика $\mathbf{x\cdot(x+y)}$, користећи **теорему о апсорпцији**, овај израз се може заменити са $\mathbf{x}$, па се целокупна функција поједностављује за два оператора (конјункцију и дисјункцију) и има једну променљиву мање.

дец.бр.	X_1	X_2	X_3	F
0	0	0	0	**0**
1	0	0	1	**0**
2	0	1	0	**0**
3	0	1	1	**1**
4	1	0	0	**1**
5	1	0	1	**0**
6	1	1	0	**1**
7	1	1	1	**1**

конјунктивна форма групише нуле (0, 1, 2, 5)

$$F = (X_1+X_2+X_3)(X_1+X_2+\overline{X}_3)(X_1+\overline{X}_2+X_3)(\overline{X}_1+X_2+\overline{X}_3)$$

(0) (1) (2) (5)

дисјунктивна форма групише јединице (3, 4, 6, 7)

$$F = \overline{X}_1X_2X_3+X_1\overline{X}_2\overline{X}_3+X_1X_2\overline{X}_3+X_1X_2X_3$$

(3) (4) (6) (7)

Слика 3.16. Одређивање нормалне форме на бази табеле

Примена наведених закона не даје одређени алгоритамски пут како да се функција минимизира. Она омогућује пројектантима да се на основу свог искуства и сагледавања појединих елемената функције снађу и смање број оператора у функцији. Зато су у оквиру Булове алгебре развијене посебне методе минимизације, а једна од најпознатијих заснива се на примени Карноових мапа (**Karnaugh**).

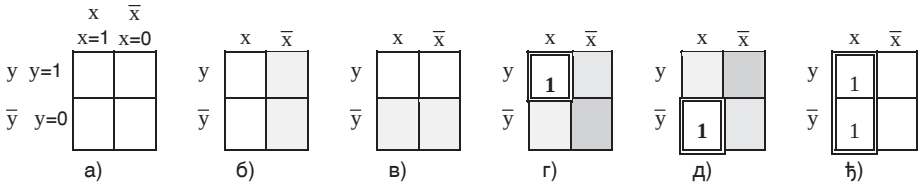
3.1.4.1 Метода Карноових мапа

По овој методи минимизација се изводи графичким путем. Она је једноставна и практична, а заснива се на уписивању функције у специјалну табелу, Карноову мапу односно дијаграм. У следећем кораку врши се минимизација, и коначно, функција се може представити у свом минималном облику. Карноове мапе су различите и зависе од броја променљивих у функцији. Обично се овај метод примењује за минимизацију функција са 2, 3 и 4 променљиве, а за функције које имају више променљивих користи се нека од других познатих метода, на пример таблична минимизација.

Погледајмо сада како изгледа Карноова мапа за две променљиве **x** и **y** слика 3.17 а). Она се састоји од 4 поља. Свако од тих поља резервисано је за једну могућу конјункцију променљивих или њихових негација. Уколико се одређена конјункција појављује у функцији, онда се у дато поље записује 1, а ако се не појављује, не записује се ништа. На тај начин се пресликава функција две променљиве на одговарајућу Карноову мапу. На пример, ако је дата функција $F(x,y) = (x \cdot y) + (x \cdot \overline{y})$, у њој се појављују две конјункције: **x·y** и $x \cdot \overline{y}$. Место конјункције **x·y** у мапи одређује се на следећи начин:

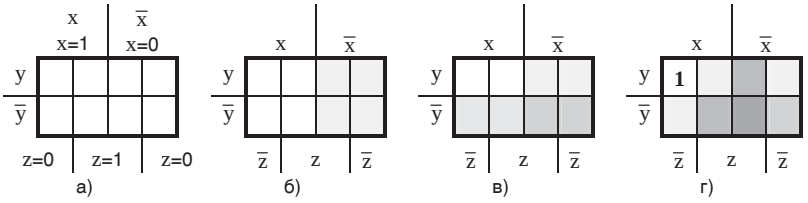
– *пошто конјункција **x·y** садржи променљиву **x** (без негације) онда се она мора налазити у области где је **x =1** (а осенчимо област **x =0**, слика 3.17. б)), а како садржи и променљиву **y** то мора бити и у области где*

је ***y=1*** *(неосенчена област на слици 3.17. в)). Дакле, конјункција* ***x·y*** *се налази на пољу које је у пресеку ове две области, у које уписујемо 1 (слика 3.17. г)).*



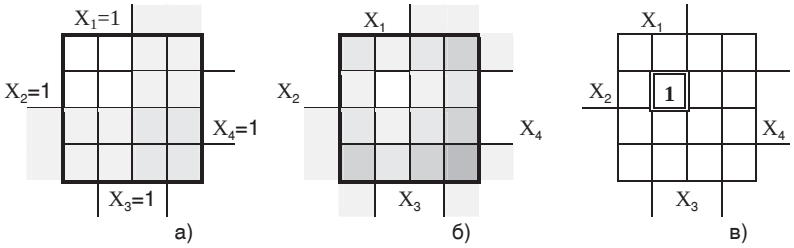
Слика 3.17. Карноова мапа за функцију са две променљиве

На исти начин се одређује поље које садржи конјункцију $x \cdot \bar{y}$, слика 3.17. д), а Карноова мапа читаве функције **F** је дата на слици 3.17. ђ). Проблем је нешто сложенији код функција за 3 променљиве. Нека су то променљиве **x, y** и **z**, Одговарајућа Карноова мапа има изглед приказан на слици 3.18 а). И у овом случају свако поље мапе резервисано је за једну тачно одређену конјункцију променљивих или њихових негација. Како се за задату конјункцију проналази одговарајуће поље? Проналажење одговарајућег поља иде постепено, елиминацијом поља која не задовољавају. Нека је дата конјункција $x \cdot y \cdot \bar{z}$. Пошто се у задатом изразу променљива **x** јавља у афирмативном облику, у обзир долазе 4 лева поља (десна половина мапе се осенчи јер та поља не долазе у обзир, слика 3.18. б)). Друга променљива је **y**, па се 4 лева поља смањују само на 2, и то у горњој врсти, а доња врста се осенчи (слика 3.18. в)). Коначно, трећи елемент је $\bar{z}$, па треба осенчити поља у средини у којима је **z=1** (слика 3.18. г)). Преостало, неосенчено поље одређује поље датог израза, и у њега уписујемо **1**, слика 3.18. г). Аналогним поступком може се наћи поље за било коју конјункцију 3 променљиве.



Слика 3.18. Карноова мапа за три променљиве

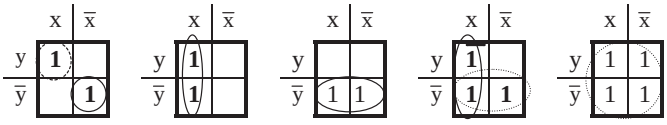
На крају ћемо приказати како се попуњава Крноова мапа (слика 3.19. а)) за функције са 4 променљиве $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_3$ и $\mathbf{X}_4$. Истим поступком елиминације одредићемо положај конјункције: $\mathbf{X}_1 \cdot \mathbf{X}_2 \cdot \mathbf{X}_3 \cdot \mathbf{X}_4$. Пошто су све променљиве без негације, треба осенчити поља у којима променљиве узимају вредност **0** (слика 3.19. а) и б)), а преостало неосенчено поље је поље у које се уписује 1 која одговара овој конјункцији, слика 3.19. в).



Слика 3.19. Карноова мапа за четири променљиве

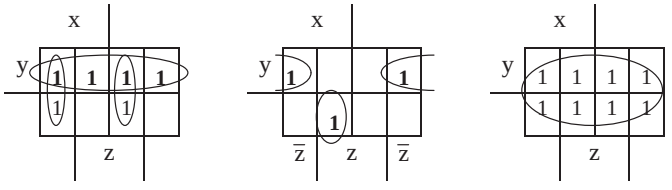
До сада смо видели како се функција, која је задата дисјунктивном нормалном формом, пресликава на одговарајућу Карноову мапу. Следећи корак је управо процес минимизације. Он се састоји у груписању јединица у оквиру мапе у веће целине. При томе се настоји да ове целине буду што веће, али оне не смеју да садрже поља која немају **1**. Може се груписати: 1 поље самостално, 2 поља, 4 поља, 8 поља или 16 поља. Пошто свака Карноова мапа има неке своје специфичности, размотримо могућност груписања код сваке врсте.

Код мапе са 2 променљиве могу се груписати: само 1 поље, два суседна поља или сва 4 поља. Када се у једној мапи прави више група, настоји се да те групе буду што веће, макар и по цену заједничког поља у групама. Примери правилног груписања су дати на слици 3.20.:



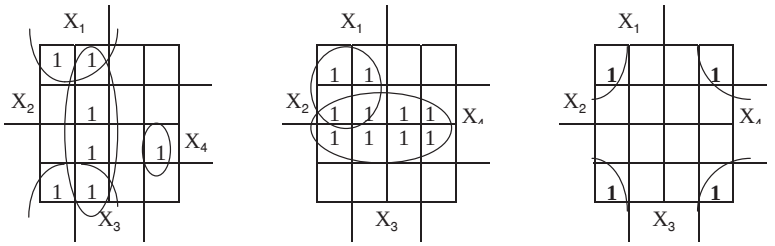
Слика 3.20. Груписање јединица у Карноовој мапи са две променљиве

Код мапе са 3 променљиве претходним принципима груписања потребно је додати још два: могу се груписати осам јединица и елементи са стране, као што је показано на слици 3.21. Ако се пажљивије посматра Карноова мапа, може се уочити да се подручје независно променљиве $\bar{z}$ протеже директно слева на десну страну табеле. То се може замислити као да је табела нацртана на ваљку, чији је омотач по висини развијен у раван.



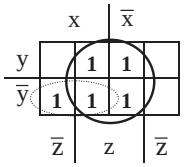
Слика 3.21. Груписање јединица у Карноовој мапи са три променљиве

Код мапе са 4 променљиве, сва претходна правила важе, а могу се гру–писати и крајње горња са крајњим доњим пољима, као и поља у угловима. Примери правилног груписања у мапама са 4 променљиве су дати на слици 3.22. Основно правило при формирању група је да треба формирати мини–малан број група тако да буду обухваћене све јединице у Карноовој мапи.



Слика 3.22. Груписање јединица у Карноовој мапи са четири променљиве

Након прве и друге фазе минимизације, које се састоје од пресликавања логичке функције на Карноову мапу и груписања јединица, приступа се пос–ледњој фази: одређивању аналитичког облика минималне форме логичке функције на основу формираних група. Ова фаза састоји се у следећем: за сваку формирану групу дефинише се конјункција променљивих или њихових негација, али само оних које су за целу групу непромењене. Када су фор–миране конјункције за све групе, коначна функција добија се као дисјункција ових појединачних чланова. Размотримо то на следећем примеру мапе са 3 променљиве где су већ формиране групе као на слици 3.23:



Слика 3.23. Одређивање аналитичког израза за групу јединица

Прво дефинишемо одговарајућу конјункцију за групу од 4 средње јединице. У овом случају група има јединице у пољима $\mathbf{x}$ ($\mathbf{x=1}$) и $\overline{\mathbf{x}}$ ($\mathbf{x=0}$) односно прелази из афирмације у негацију, па та варијабла неће фигурисати у резултујућем изразу. У датој групи има поља и са $\mathbf{y=1}$ и са $\mathbf{y=0}$, па ни ова варијабла не утиче на резултујућу конјункцију. Група је дефинисана у свим пољима са $\mathbf{z=1}$ (област $\mathbf{z}$), тако да варијабла $\mathbf{z}$ улази у конјункцију. Коначно, први фактор минималне логичке функције је $\mathbf{z}$ (није везан конјункцијом јер је само једна променљива).

Друга група састоји се од два поља у доњем левом углу мапе. Вредност варијабле $\mathbf{x}$ је јединствена у овој групи, па улази у конјункцију. Затим, у

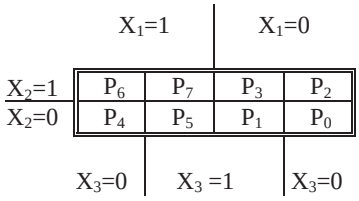
групи постоји поље са варијаблом **z**, али и са $\overline{\mathbf{z}}$, па, према томе, ова варијабла не улази у коначни израз за ову групу. Како је за ову групу и варијабла $\overline{\mathbf{y}}$ јединствена, то и она улази у израз, па је коначан облик конјункције за ову групу: $\mathbf{x}\cdot\overline{\mathbf{y}}$. Минимални облик ове функције добија се повезивањем конјункција за поједине групе оператором дисјункције, па је у нашем случају:

$$\mathbf{F = z + (x \cdot \overline{y})}$$

Приликом минимизације логичких функција могуће је поћи и од таблице истинитости, а не само од дисјунктивне нормалне форме, слика 3.24. Да би се таблица истинитости пресликала на Карноову мапу, посматрају се сви редови где је функција једнака **1** и одговарајућа комбинација 1 и 0 променљивих дефинише поље у мапи. Наиме, када променљива има вредност 1, посматра се у Карноовој мапи област у којој је та променљива једнака јединици, а ако има вредност 0, посматра се њена негација.

дек. бр.	X_1	X_2	X_3	Z
0	0	0	0	P_0
1	0	0	1	P_1
2	0	1	0	P_2
3	0	1	1	P_3
4	1	0	0	P_4
5	1	0	1	P_5
6	1	1	0	P_6
7	1	1	1	P_7

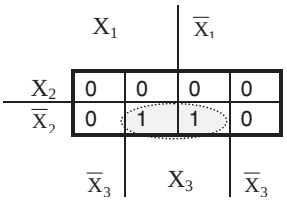
а)



б)

декад. бр.	X_1	X_2	X_3	Z
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0

в)



г)

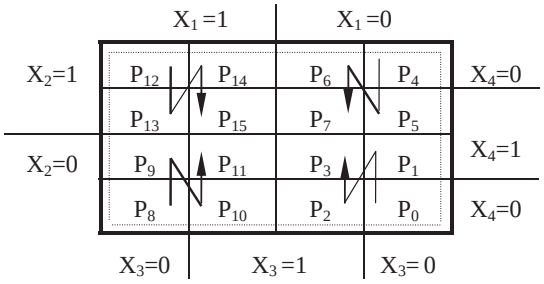
Слика 3.24. Попуњавање Карноове мапе и минимизација функције **z**

Међутим, ако се променљиве X_1, X_2, X_3 распореде као на слици 3.24. а), онда се Карноова мапа може попунити по шаблону јер свакој врсти из табеле увек одговара једно те исто место у Карноовој мапи, као на слици 3.24. б). Како је положај конјункција фиксиран, то се само уместо P_0, P_1, ...

P_7 упишу јединице или нуле (слика 3.24. г)), сагласно табели истинитости, слика 3.24. в). На страницама дијаграма д) назначене су области ненегираних променљивих (афирмација) и њихових комплемената (негација). Графички се минимизација изводи тако што се у табели заокруже две суседне јединице, као у табели на слици 3.24. г), за функцију задату табелом в). Посматра се која од променљивих у оквиру групе јединица не прелази из афирмације у негацију. То су у датом примеру $\overline{X}_2$ и X_3 . Њихов логички производ ће представљати тражену минималну дисјунктивну форму функције:

$$\mathbf{F} = \overline{\mathbf{X}}_2 \cdot \mathbf{X}_3$$

За функцију са четири променљиве, при фиксном распореду променљивих, редослед ређања у Карноовој мапи елемената дат је на слици 3.25. Ову табелу треба замислити као да је нацртана на торусу, па развијена у раван. Другим речима, њена лева ивица се директно наслања на десну, а доња ивица на горњу. Ово треба имати у виду приликом заокруживања суседних чланова.



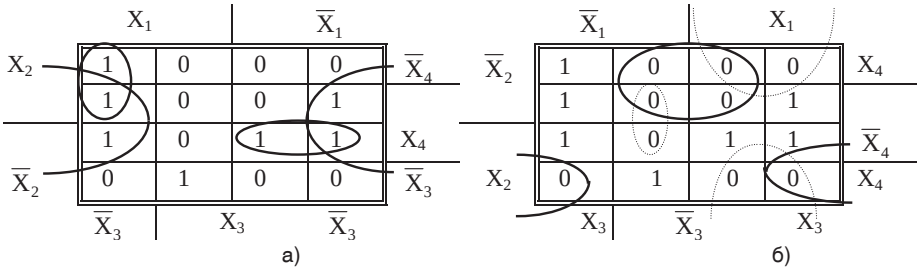
Слика 3.25. Карноова мапа за четири променљиве

За налажење минималне конјунктивне форме (**MKF**) функције, треба најпре попунити Карноову мапу и то на исти начин као код дисјунктивне форме. Затим треба груписати појединачна поља, парове, квартете и октете нула и то на исти начин као што се групишу јединице код дисјунктивне форме. На крају треба одредити аналитички израз за минималне дисјункције, с тим што афирмације и негације променљивих на страницама Карноове мапе међусобно замењују места. Другим речима, поља у области где је променљива једнака нули узимају се као афирмација, нпр. област $\mathbf{x=0}$ је област $\mathbf{x}$, а област $\mathbf{x=1}$ је област $\overline{\mathbf{x}}$. Табела на слици 3.26. а) показује Карноову мапу за добијање дисјунктивне, а табела б) показује Карноову мапу за добијање конјунктивне форме функције. Као што се види, оне су идентичне, само су области негације и афирмације променљивих укрштене. Минимална дисјунктивна форма (**MDF**) је:

$$F = X_1X_2\overline{X}_3 + \overline{X}_3X_4 + \overline{X}_1\overline{X}_2X_4 + X_1\overline{X}_2X_3\overline{X}_4 .$$

MKF има онолико логичких сума (дисјункција) колико у табели постоји заокружених поља, парова, квартета и октета нула. У свакој од тих сума учествују оне промењљиве које у интервалу заокружених група не прелазе из афирмације у негацију. Квартет нула у средини даје суму $\overline{X}_2 + \overline{X}_3$, а квартет заокружен испрекиданом линијом даје суму X_1+X_4. Од пара нула заокружених испрекиданом линијом добија се члан $\overline{X}_1 + \overline{X}_3 + \overline{X}_4$, а пар заокружен пуном линијом даје дисјункцију $X_2 + X_3 + X_4$. Финални облик MKF добија се када се направи логички производ (конјункција) ових дисјункција:

$$F = (\overline{X}_2 + \overline{X}_3) \cdot (\overline{X}_1 + \overline{X}_3 + \overline{X}_4) \cdot (X_1 + X_3) \cdot (X_2 + X_3 + X_4)$$



Слика 3.26. Пример Карноових мапа за функцију са четири промењљиве

У практичним колима може се десити да се одређене комбинације независно промењљивих никада не појављују. Тада се место у табели које одговара тој комбинацији промењљивих може третирати и као логичка нула и као логичка јединица, зависно од тога како се добија оптималније груписање са суседним јединицама (код MDF), односно нулама (код MKF).

3.2. ЕЛЕМЕНТАРНА МЕМОРИЈСКА КОЛА

Логичка стања **0** и **1** у дигиталним логичким колима представљена су ниским и високим напоном. Током рада рачунара (у поступку обраде), напонски сигнали пролазе кроз електронска кола и при томе се брзо и непрекидно мењају. Подаци долазе на улазе логичких кола у виду електричних импулса преко једне или више улазних линије везе. Логичка кола обрађују ове сигнале и генеришу излазни напонски сигнал, који се такође води на неке улазне линије других логичких кола. Чим се промени ниво напона на улазу, мења се и ниво напона на излазу (сагласно закону обраде у тој логичкој мрежи). Логичке мреже, код којих излазна стања зависе само од тренутне вредности улазних величина називају се **комбинационе мреже.**

Другу врсту логичких мрежа тзв. **секвенцијалне мреже**, чине мреже код којих логичко стање на излазу зависи не само од тренутне вредности сигнала на улазу, већ и од претходног стања у ком се та логичка мрежа

налазила. Да би била могућа реализација оваквих мрежа, морају постојати такви електронски склопови способни да **запамте** претходно стање. Таква електронска кола називају се меморијски елементи. Минимална количина података која се може **запамтити** је једна **0** или **1**, тј. један бит. Логичка мрежа која може да запамти један бит (једну бинарну цифру), зове се **флип–флоп**.

3.2.1 ФЛИП–ФЛОП

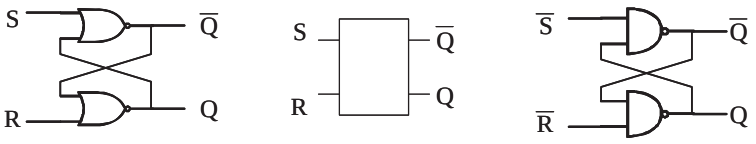
Флип–флоп је једно од најпростијих кола са два стабилна стања која се користе за складиштење, односно меморисање података у бинарном облику, а истовремено и једно од основних кола дигиталне технике.

Бит информације се кодира присуством или одсуством импулса, или логичког нивоа 1 или 0, па самим тим један флип–флоп може да памти у одређеном времену само једну бинарну цифру, тј. један бит информације. Подаци већи од једног бита, памте се у уређеном скупу флип–флопова који се назива **регистар**. Скуп више регистара, организованих на одређени начин, чини **меморију**. Рад флип–флопа као меморијског елемента може бити приказан табелом истинитосних вредности или помоћу одговарајућих логичких функција приказаних у аналитичком облику.

3.2.1.1 R/S флип–флоп

R/S флип–флоп се састоји од укрштене везе два **НИЛИ** (или два **НИ**) кола, тако да је излаз првог спојен на улаз другог, а излаз другог на улаз првог. На тај начин је остварена позитивна повратна спрега потребна за кумулативни процес при промени стабилних стања.

Логичка структура R/S флип–флопа, реализованог укрштањем два двоулазна **НИЛИ** кола, приказана је на слици 3.27 а). На слици 3.27 б) приказан је графички симбол флип–флопа.



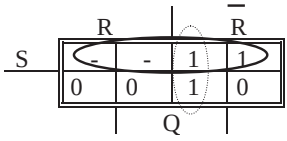
Слика 3.27. **R/S** флип–флоп реализован **НИЛИ** и **НИ** колима

Конвенцијом је усвојено да се стање флип–флопа изражава и интерпретира логичком вредношћу напона на једном излазу, који се назива главни излаз

флип–флопа и најчешће означава са **Q**. Други излаз је увек комплементаран $\overline{Q}$, и уколико се користи, елиминише употребу једног **НЕ** кола. За почетно стање флип–флопа усвојено је стање логичке 0 на главном излазу, тј **Q = 0**. Улаз **S** (**Set**) служи за постављање излаза флип–флопа на стање логичке јединице (сетовање), тј. **Q = 1**, $\overline{Q}$ **= 0**. Други улаз **R** (**Reset**) служи за довођење излаза у стање логичке нуле (ресетовање), тј. **Q = 0**, $\overline{Q}$ **= 1**.

Стање излаза у које ће флип–флоп прећи у наредном тренутку $(t+1)$, означава се као $Q(t+1)$, и не зависи само од улазних сигнала $S(t)$ и $R(t)$, већ и од стања флип–флопа $Q(t)$, у посматраном тренутку t. Табела стања, на слици 3.28, у потпуности одређује рад R/S флип–флопа.

R(t)	S(t)	Q(t)	**Q(t+1)**
0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**1**
1	0	0	**0**
1	0	1	**0**
1	1	0	-
1	1	1	-



Слика 3.28. Табела стања R/S флип флопа и Карноова мапа

Минимизацијом на основу Карноове мапе са слике 3.28, добијамо минималну дисјунктивну форму функције R/S флип–флопа:

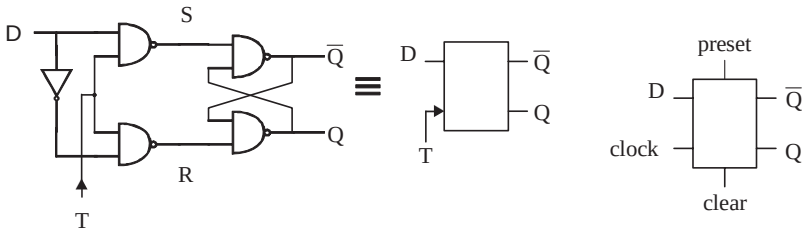
$$Q(t+1) = S(t) + \overline{R}(t)Q(t),$$

уз услов да $R(t)S(t) = 0$, којим се забрањује да оба улаза истовремено буду једнака јединици (не може истовремено бити $S(t) = 1$ и $R(t) = 1$), јер тада стање излаза није дефинисано. Наиме, пошто је $S = 1$ онда излаз Q мора бити јединица, али због $R = 1$ излаз би морао да буде нула. Дакле, излаз би у истом тренутку времена требао да има две различите вредности, што је немогуће.

R/S флип–флоп се може реализовати и помоћу два **НИ** кола, слика 3.27 ц). Код овог флип–флопа постоји неодређеност излаза у случају да на оба улаза и S и R дођу нуле ($S = 0$ и $R = 0$, односно када је $\overline{R}(t) \cdot \overline{S}(t) = 1$). Због појаве ових неодређености излаза, праве се и друге врсте флип–флопова, као на пример **R/S** флип–флопови са једним доминантним улазом, затим **D** флип–флоп, **JK** флип–флоп, **T** флип–флоп, **JKT** флип–флоп, **MS** флип–флоп, итд.

3.2.1.2 D флип–флоп

R/S флип–флоп није погодно коло за померање података у рачунару, из простог разлога што је потребно остварити везу на оба улаза. Уписивање јединице се врши довођењем сигнала S = 1, док се уписивање нуле обавља довођењем сигнала R = 1. Знатно боље коло за прихватање података и њихов пренос, представља **D** флип–флоп (**data flip-flop**). Овај флип–флоп осигурава да не дође до истовремене побуде на оба улаза. Логичка структура, табела истинитости и симболичка ознака овог флип–флопа, дати су на слици 3.29.



Слика 3.29. **D** флип–флоп

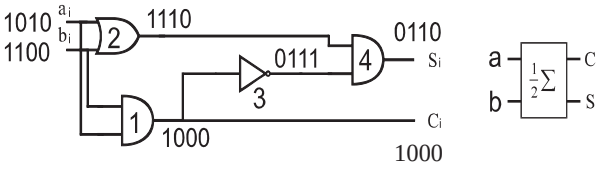
Улаз са ознаком **T** назива се синхронизациони улаз, или такт сигнал (clock). Треба напоменути да сва дигитална логичка кола у једном рачунару најчешће раде под дејством једног јединственог временског сигнала, који обезбеђује једновременост, синхронизацију у читавом рачунару. Учестаност такт сигнала се мери јединицом која се зове мегахерц (**MHz**), а то је милион импулса у секунди. Ако је такт сигнал **50 MHz**, то значи да има 50 милиона импулса у секунди, или један импулс сваких **20** нано секунди. Такт сигнали се генеришу помоћу посебних електронских склопова који се зову **осцилатори** или **такт генератори**. Такт сигнал се уз помоћ кола за кашњење може поделити на мање временске интервале. Исто тако, уз помоћ специјалних дигиталних кола, такозваних делитеља, такт интервали времена се могу повећавати. На тај начин се добијају импулси различитог трајања и учестаности, а користе се у управљању свим елементарним операцијама у централном процесору и осталим деловима рачунара.

3.3. ЕЛЕМЕНТАРНА АРИТМЕТИЧКА КОЛА

Помћење података је једна од функција коју обављају дигитална кола у рачунару. Она такође могу бити намењена и обављању бинарних аритметичких операција. На слици 3.30 приказана је логичка мрежа позната под називом полусабирач (**half adder**), његова симболичка ознака и табела

истинитости. Полусабирач је реализован уз помоћ **И**, **ИЛИ** и **НЕ** логичких кола. Ова логичка мрежа врши сабирање два бита, две бинарне цифре. Резултат сабирања бинарних бројева садржи две компоненте: збир $\mathbf{S_i}$ и пренос у следећи разред $\mathbf{C_i}$.

улази		пренос	збир
$\mathbf{a_i}$	$\mathbf{b_i}$	$\mathbf{C_i}$	$\mathbf{S_i}$
0	0	**0**	**0**
1	0	**0**	**1**
0	1	**0**	**1**
1	1	**1**	**0**



Слика 3.30. Табела истинитости, логичка мрежа полусабирача, блок дијаграм

Улазни подаци $\mathbf{a_i}$, $\mathbf{b_i}$ ступају на улазе логичких кола, која на основу њихових тренутних вредности генеришу сигнале на својим излазима. Када су оба улазна сигнала нула, на излазу првог **И** кола (1) генерише се пренос: $\mathbf{C_i = 0}$. Иста два улазна сигнала долазе и на улаз **ИЛИ** кола (2), које такође на свом излазу даје нулу. **НЕ** коло (3) прихвата бит преноса и претвара **нулу** са свог улаза у **јединицу** на свом излазу. Ова јединица за–једно са нулом из **ИЛИ** кола (2) долази на улазе другог **И** кола (4) које на свом излазу генерише нулу, сигнал на излазу **И** кола 4 представља збир $\mathbf{S_i}$ бинарних цифара $\mathbf{a_i}$ и $\mathbf{b_i}$ са улаза.

Када је $\mathbf{a_i = 1}$ а $\mathbf{b_i = 0}$, прво **И** коло конвертује улазне податке у нула бит преноса на свом излазу ($\mathbf{C_i = 0}$). **ИЛИ** коло конвертује те исте улазне по–датке у јединицу на свом излазу. Инвертор **НЕ** прихвата бит преноса и претвара га у јединицу, која заједно са јединицом из **ИЛИ** кола долази на улазе другог **И** кола. Друго **И** коло на основу две јединице на својим улазима прави јединицу на свом излазу, која представља збир $\mathbf{S_i = 1.}$

У трећем случају за улазне податке $\mathbf{a_i = 0}$ и $\mathbf{b_i = 1}$, поступак обраде и резултат су исти, тј.: $\mathbf{S_i = 1}$ и $\mathbf{C_i = 0}$.

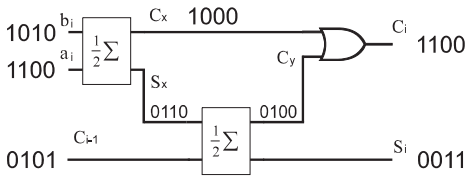
У четвртом случају, треба сабрати две бинарне јединице, $\mathbf{a_i = 1}$ и $\mathbf{b_i = 1}$. Када на улаз првог **И** кола дођу две јединице, ово коло на свом излазу даје бит преноса који је такође једнак јединици, $\mathbf{C_i = 1}$. **ИЛИ** коло прихвата две јединице на својим улазима, и на излазу даје такође јединицу. Бит преноса ступа на улаз инвертора **НЕ**, који од јединице прави нулу на свом излазу. Ова нула заједно са јединицом са излаза **ИЛИ** кола долази на улазе другог **И** кола, које на излазу даје збир $\mathbf{S_i = 0}$. Значи, резултат је:

$$\mathbf{S_i = 0} \quad \text{и} \quad \mathbf{C_i = 1}.$$

Полусабирач је погодан једино за сабирање само два бита, али не може сабрати и пренос из претходног разреда. Но, код многих рачунара се и за

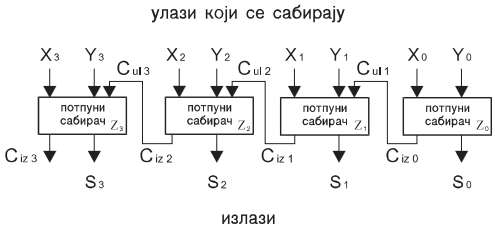
оваква сабирања користи другачије решење. Пренос из претходног разреда се може јавити и код сабирања вишецифрених бинарних бројева. Сабирање бројева са учешћем бита преноса из претходног разреда обавља, се уз помоћ дигиталне мреже која се зове потпуни сабирач, или суматор (**full adder**). Шема потпуног сабирача и његова таблица истинитости дати су на слици 3.31. Као што се види са слике потпуни сабирач се састоји од два полусабирача и једног логичког **ИЛИ** кола. На улазе једног полусабирача долазе бинарне цифре $\mathbf{a_i}$ и $\mathbf{b_i}$, и он на својим излазима даје делимичан збир $\mathbf{S_x}$ и пренос $\mathbf{C_x}$. На улазе другог полусабирача долазе делимични збир $\mathbf{S_x}$ и пренос из претходног разреда $\mathbf{C_{i-1}}$. Други полусабирач на свом излазу даје потпуни збир **Si** бинарних цифара $\mathbf{a_i}$ и $\mathbf{b_i}$ и преноса из претходног разреда $\mathbf{C_{i-1}}$, и делимични пренос $\mathbf{C_y}$. Коначни пренос у следећи разред $\mathbf{C_i}$, добија се на излазу из **ИЛИ** кола.

$\mathbf{a_i}$	0	**0**	**0**	0	1	**1**	**1**	1
$\mathbf{b_i}$	0	**0**	**1**	1	0	**0**	**1**	1
$\mathbf{C_{i-1}}$	0	**1**	**0**	1	0	**1**	**0**	1
$\mathbf{S_i}$	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**1**
$\mathbf{C_i}$	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**1**



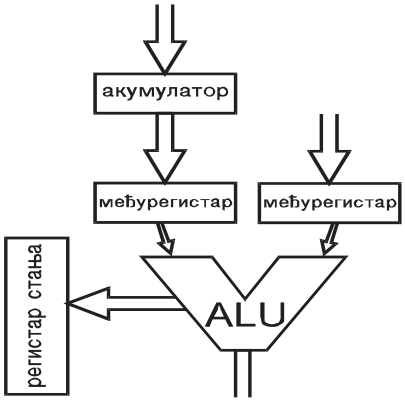
Слика 3.31. Табела итинитости и блок дијаграм потпуног сабирача

Овакав процес израчунавања понавља се за све разреде бинарног броја. Некада су се због високе цене хардвера правили и рачунари који су имали само један сабирач на чије улазе су се доводиле бинарне цифре и пренос из претходног разреда један за другим, тј серијски. Овакав сабирач се звао серијски или редни сабирач, а сабирање осмобитног броја се одвијало у осам тактова. Данас се за сваки бинарни разред користи посебан сабирач. Тако за сабирање 32–битних бројева постоји један полусабирач за сабирање **LSB**, и низ од 31–ог потпуног сабирача за сабирање преосталих ботова, или низ од 32 потпуна сабирача, на чије улазе истовремено долазе свих 32 бита бројева који се сабирају. Овакво сабирање се зове паралелно сабирање, а део аритметичко–логичке јединице која то обавља зове се паралелни сабирач, слика 3.32.



Слика 3.32. Блок дијаграм паралелног четворобитног сабирача

Аритметичко–логичка јединица (**ALU**) обавља аритметичке и логичке операције над бинарним бројевима и ротирање. То могу бити операције сабирања, одузимања, множења и дељења бинарних и BCD бројева, логичке операције, разне врсте померања бинарних бројева улево и удесно, и можда још понека једноставна операција, али обично не много више од набројеног. Помоћу тих основних операција решавају се задаци постављени рачунару. Аритметичко–логичка јединица се обично шематски представља у облику слова **V**, као што је приказано на слици 3.33.



Слика 3.33. Шематски приказ ALU са припадајућим регистрима

Поред сабирача, у аритметичко–логичкој јединици се налазе и многи други склопови. Ту су логичке мреже које обављају операције **И, ИЛИ**, **искључиво ИЛИ** и **НЕ** над свим цифрама бинарних бројева истовремено, затим померачки регистри, кола за комплементирање итд. Код неких великих рачунара у састав ALU улазе и склопови за BCD аритметичке операције, склопови за рачунање у покретном зарезу. Многи аутори у састав ALU укључују и неке регистре специјалне намене у којима се налазе операнди и резултат операције, то су привремени регистри и акумулатори, као и један специјални регистар у који се аутоматски уписују одређени подаци након сваке операције коју изврши централни процесор. Тај регистар у разним рачунарским системима има другачије име, а ми ћемо га звати регистар стања.

3.4. РЕГИСТРИ

Најчешће коришћени елементи дигиталних уређаја су различите врсте регистара. Регистар је елемент који служи за складиштење произвољног бинарног броја ограничене дужине. Свака бинарна цифра датог броја чува се

у засебном меморијском елементу, па је за бинарни број од **n** цифара потребно **n** бинарних меморијских ћелија. Регистри се примењују углавном за привремено меморисање или прихватање делимичних или коначних резултата у процесу обраде података. Неопходни су на свим местима где треба остварити везу између блокова са различитим брзинама, затим при реализовању аритметичких операција, за претварање серијског у паралелни код и обратно, итд.

Бинарни број који се налази у регистру назива се садржај регистра. За улаз бинарног броја у регистар користи се термин уписивање, док се за излаз броја из регистра користи термин читање. Уписивање и читање садржаја може да се уради на два начина:

1. паралелно, када све цифре бинарног броја улазе у регистар или излазе из њега истовремено,
2. серијски, када број улази, односно излази из регистра цифра по цифра (**бит по бит**), почевши од најмађег или најстаријег разреда.

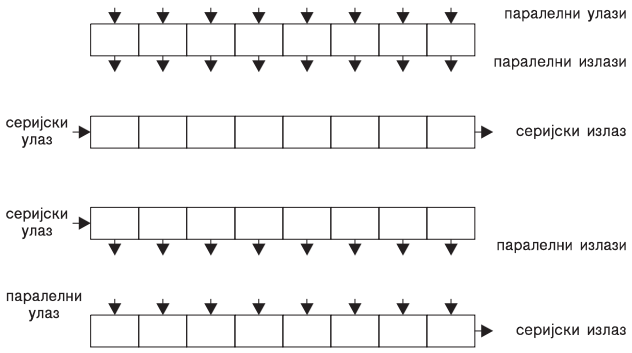
Комбинацијом описаних начина улаза и излаза информација разликујемо различите типове регистара, слика 3.34:

1. регистри са паралелним улазом и паралелним излазом,

2. регистри са серијским улазом и серијским излазом,

3. регистри са серијским улазом и паралелним излазом,

4. регистри са паралелним улазом и серијским излазом.



Слика 3.34. Основне врсте регистара

Код првог типа регистара уписана информација остаје стално у њему, и они се зову стационарни регистри. Код остала три типа регистара садржај регистра се помера како пристижу нови битови, и уколико се не очита на време, он може бити и изгубљен. Због тога се ови регистри називају динамички, померачки или шифт–регистри (**shift registers**).

У састав централног процесора улазе поред ALU и контролно–управљачка јединица и већи број регистара. Ови регистри се деле на две групе сагласно њиховој намени:

– *регистри опште намене,*

– *регистри специјалне намене (акумулатори, међурегистри, регистар стања (статус регистар), програмски бројач, регистар инструкција, регистар података, адресни регистри и низ других регистара). На овом месту ћемо поменути само неке који су тренутно од интереса а улога осталих ће бити објашњена у оквиру обрада у којима учествују.*

Акумулатор (**accumulator**) је регистар у којем се обично добијају (акумулирају) резултати различитих операција с бинарним бројевима. Подаци у акумулатор долазе са магистрале за податке, а из акумулатора се преносе на међурегистар. У саставу процесора може бити један или више акумулатора.

Међурегистри (**buffers**) привремено "памте"–прихватају податке, и то један (на слици 3.33 лево) податке из акумулатора, а други (десни) податке који долазе право са магистрале. Такво привремено памћење података потребно је да би се по два податка могла довести на улаз аритметичко–логичке јединице, која их обрађује у прикладном моменту. Поред ових међурегистара у процесору се налазе и меморијски адресни регистар и прихватни регистар података, чију улогу ћемо објаснити у оквиру меморије. Према томе, међурегистри имају само помоћну улогу, па се понекад при разматрању преноса података могу изоставити.

Регистри опште намене (R_0-R_N) постоје у већини рачунара, сем код неких микрорачунара. Број ових регистара варира од рачунара до рачунара, па их могу имати од једног или два до неколико десетина. На тим регистрима привремено се записују различити подаци, и зато су то регистри опште намене. Они су регистри "при руци", па је приступ до њих једноставнији и бржи него до регистра у меморији. Време приступа подацима унутар ових регистара је вишеструко краће него када се приступа подацима у меморији. Стога се њиховом употребом убрзава рад процесора. Но ови регистри су истовремено врло скупи јер се израђују у специјалним полупроводничким технологијама. Подаци на магистралу долазе или из прихватног регистра података, или из регистара опште намене или из аритметичко–логичке јединице. Подаци са магистрале иду на регистре опште намене, на акумулатор, или на неки од међурегистара.

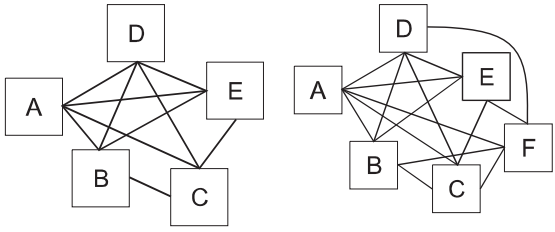
Регистар стања (регистар статуса, регистар услова), служи за то да се на њему прикажу различита стања која могу настати током обраде података. У ствари, тај се регистар састоји од низа међусобно независних флип–флопова такозваних заставица (**flags**). Свака од тих заставица сигнализира неко стање које је настало током обраде података. То могу бити, на пример ова

стања: *резултат је једнак нули, омогућен прекид програма, дошло је до препуњења, постоји бит преноса* итд. Таква стања могу утицати на даље одвијање рада. Те заставице омогућују кориснику (програмеру) да провером стања одређене заставице усмери одвијање програма.

Адресни регистри (**address registers**) служе за чување адреса меморијских локација у којима се налазе инструкције (бројач инструкција), или адреса локација у којима се налазе подаци (бројач података, показивач стека, индекс регистри), или пак садрже додатне информације помоћу којих се израчунавају адресе и података и инструкција (базни и сегментни регистри).

3.5. МАГИСТРАЛЕ

Да би рачунар могао да ради, потребна је комуникација између логичких кола, флип–флопова, померачких регистара и других компоненти, које улазе у састав централног процесора. Комуникација мора постојати и између централног процесора, меморија, улазних и излазних јединица, и других уређаја који улазе у састав рачунарског система. Неке од компоненти рачунара остварују везе са само неколико других склопова, док неке друге захтевају повезивање са великим бројем других делова рачунарског система. Неки рачунарски системи допуштају директно повезивање свих компоненти, такозвано повезивање **од тачке до тачке** (**point to point**), приказано на слици 3.35.

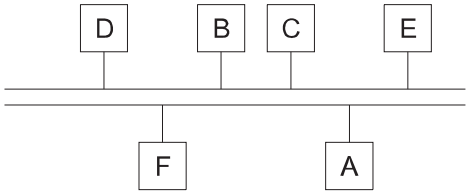


Слика 3.35. Повезивање од тачке до тачке и додавање нове компоненте

Ако би компоненте које треба повезати били регистри са **n** ботова, онда је за повезивање **p** регистра потребно: (**p-1**). **p.n/2** линија везе. При додавању макар и само једне нове компоненте треба остварити велики број нових веза, па је реконфигурација система практично врло тешко изводива. Предност ове структуре је велика брзина преноса, јер се истовремено може остварити већи број међусобних веза једновремено (повезивање више регистара) и поузданост, јер се у случају отказа директне везе, веза између два регистра може успоставити преко других регистара.

Решење проблема повезивања више компоненти јесте у коришћењу заједничких комуникационих линија које се зову сабирнице или магистрале (**bus**), као што је приказано на слици 3.36.

Група линија преко којих се информације у бинарном облику преносе између регистара у саставу централног процесора, назива се интерна магистрала. У оваквој организацији CPU дуж једне магистрале у јединици времена, могуће је остварити само један пренос, али је зато број линија једнак **n,** и не зависи од броја регистара (и других дигиталних склопова) који су на њу повезани. Да би се остварио пренос између два регистра посредством магистрале, мора се путем једне групе линија – **адресне магистрале** (**address bus**), послати бинарни број који означава адресу регистра који у том тренутку суделује у преносу података, а чији подаци се налазе (или ће се наћи) на **магистрали података** (**data bus**). Да би се све одвијало како треба, брине нека управљачка јединица (најчешће у саставу **CPU**) која преко **управљачке магистрале** (**control bus**), шаље управљачке сигнале који одређују смер преноса података и сигнале који усклађују догађаје у времену. Иако су ово по функцији три различите магистрале, често се на цртежима приказују као једна.



Слика 3.36. Генералисана структура магистрале

Многи рачунари имају неколико разних магистрала, то су такозвани **multiple–bus** рачунари. Код већине таквих рачунара постоје два основна система магистрала. Један су интерне, тј. **унутрашње магистрале** (слика 1.4), које служе за пренос података између главне меморије и различитих делова централног процесора, (ALU, разних регистара и сл.). Други чине **спољашње магистрале** које повезују централни процесор са осталим деловима рачунарског система. Коришћење две магистрале повећава брзину рада и омогућава паралелно одвијање меморијских и улазно–излазних операција. Али унутрашње и спољне магистрале су међусобно повезане, јер само на тај начин може се усклађено одвијати рад процесора и осталих јединица које чине рачунарски систем. Повезивање ових магистрала обавља се помоћу улазно–излазних међусклопова (**interfaces**).

Друга врста рачунара има само једну магистралу на коју су повезани сви делови рачунарског система. То су такозвани **single–bus** рачунари (слика

1.3). Предности овакве архитектуре јесу мала цена и једноставно додавање нових периферијских уређаја. Нормално, то је плаћено смањењем брзине рада рачунара. На основу ових разматрања можемо закључити да се једна магистрала користи обично код малих рачунара који се зову **мини** и **микрорачунари**, а да велики рачунари обично користе системе са више магистрала.

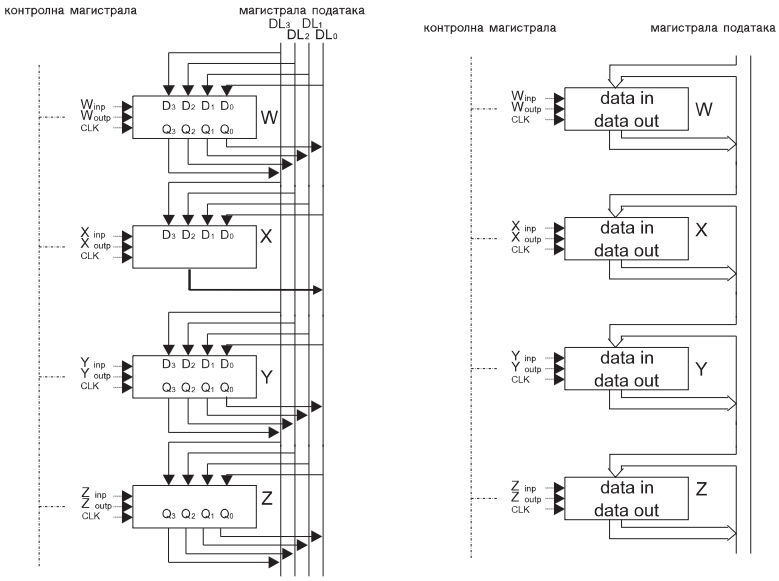
Иако је на магистралу повезано мноштво компоненти, у једном тренутку времена, у преносу података учествују само две. Остали склопови не учествују у преносу, у датом тренутку, али и не смеју ометати пренос између активних компоненти. Ово је могуће остварити захваљујући примени **buffer** регистара и специјалне класе логичких кола у изради рачунарских склопова. То су логичка кола са три стања, тј. ова кола могу бити у стању логичке нуле, јединице и бесконачне импедансе. Када су дигитална кола у стању бесконачне импедансе, она као да нису повезана на магистралу, тј. као да не постоје. Пренос података дуж магистрала додатно компликује чињеница да сви делови рачунарског система који су повезани на магистралу немају исту брзину рада. Неки електромеханички уређаји су релативно спори, на пример: принтери, терминали, плотери, тастатуре и сл. Дискови и траке су знатно бржи од њих, али кудикамо спорији од оперативне меморије или процесора. Како сви ови уређаји морају да комуницирају преко магистрале, неопходно је обезбедити ефикасан механизам којим ће се премостити разлике у њиховој брзини рада.

Решење је нађено у употреби међурегистара (**buffer registers**) који се уграђују у ове уређаје да би привремено чували податке за време преноса. Погледајмо то на примеру преноса једног карактера (знака) из процесора у принтер где ће бити одштампан. Процесор започиње пренос слањем тог знака преко магистрале на међурегистар (**buffer**) за принтер. Пошто је **buffer** електронски склоп овај пренос захтева мало времена. Када је међурегистар напуњен, принтер може да почне са штампањем без додатних интервенција од стране процесора. Дакле, процесор и магистрала више нису потребни принтеру и могу се користити за друге активности. Принтер наставља са штампањем карактера који је записан у његовом **buffer**–у, и није на располагању за друге захтеве за штампањем док не заврши штампање оног знака који се налази у његовом **buffer**–у.

Да закључимо, међурегистар елиминише разлике у брзини рада процесора и принтера и онемогућава да спори уређаји блокирају рад брзих уређаја, односно да процесор чека док принтер не заврши са штампањем. Ово омогућава процесору да се брзо пребацује са једног уређаја на други, и да практично истовремено управља преносом података са и у више разних уређаја.

Број магистрала битно утиче на организацију централног процесора, а карактеристична су три типа организације: око једне, око две и око три магистрале. Пре тога покажимо како се посредством једне четворобитне

магистрале $\mathbf{DL_0}$ - $\mathbf{DL_3}$, на коју су повезана четири регистра **X, Y, Z, W**, врши пренос података између два регистра, слика 3.37 а).



а) б)

Слика 3.37. Пример а) четворобитне магистрале и б) и хипотетичке рачунарске магистрале

Шта се догађа на магистрали и који су управљачки импулси потребни да се изврши пренос податка из регистра **Z** у регистар **X** (који има паралелни улаз и серијски излаз)**.** Свака линија података садржи по један бит. Нека је у регистру **Z** запамћен број 0110. Поступак преноса је следећи:

1. *послати* $\mathbf{Z}_{outp}$ *управљачки сигнал регистру* Z*,*
2. *у следећем циклусу такт сигнала, чита се податак из регистра* Z *и шаље се на магистралу,*
3. *податак је на магистрали и доступан је свим регистрима* X*,* Y*,* Z *и* W*,*
4. *послати* $\boldsymbol{X}_{inp}$ *управљачки сигнал на регистар* X*,*
5. *у следећем циклусу такта* **CLK***, регистар* X *прихвата податке са магистрале.*

Такт сигнал, долази на улаз свих регистара, односно сва дигитална кола у рачунару су тактована. Мада су сви регистри повезани на магистралу активан је само онај који је добио одговарајући управљачки сигнал **OUTP**, а сва остала кола су у стању бесконачне импедансе, тј. њихови излази као да нису повезани на магистралу. Сигнал **OUTP** траје све до завршетка

преноса података. Иако су улази свих регистара повезани на магистралу, податак ће се пренети само у онај регистар који добије управљачки импулс INP. Понекад је пре уписа новог податка у регистар потребно прво извршити брисање старог садржаја, тј. уписати 0 у све ћелије регистра. На слици 3.37 (б) приказана је хипотетичка магистрала података у рачунару, где се због прегледности не уцртавају линије сваког бита понаособ, јер би то било крајње непрактично (и неразумљиво) у случају већег броја ботова (нпр. 16 или 32 и више).

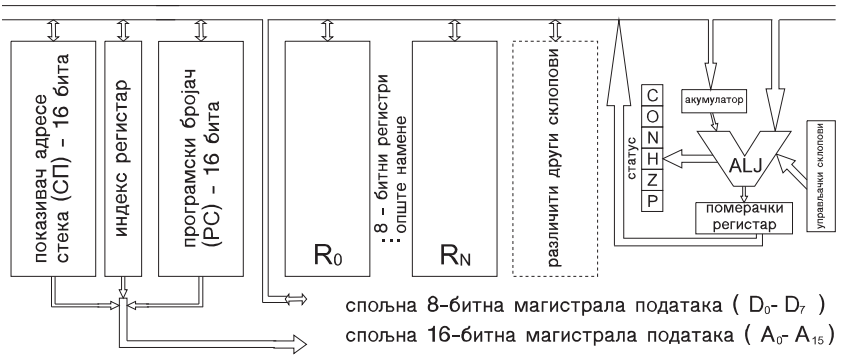
Без обзира на то о којој се магистрали ради, на њу је прикључен већи број регистара, при чему сваки од њих у принципу може бити или извор или одредиште података. При сваком преносу учествују само два регистра. Остали, иако прикључени на магистралу, не учествују у преносу. У сваком процесору може се налазити једна или више магистрала. Свака магистрала садржи три групе линија: једносмерне линије за пренос сигнала адресе – **адресна магистрала** (адресе меморијских локација или улазних и излазних уређаја), и двосмерне линије за пренос података и управљачких сигнала – **магистрала података** и **управљачка магистрала**. Будући да број магистрала знатно утиче на организацију процесора и рачунара, приказаћемо карактеристичне типове организације процесора око једне, две или три магистрале.

3.5.1. ОРГАНИЗАЦИЈА ПРОЦЕСОРА ОКО ЈЕДНЕ МАГИСТРАЛЕ

При градњи рачунарских система ради се искључиво са спољним магистралама, тј. оним које се налазе ван процесора. Међутим, при разматрању интерне организације процесора одлучујућу улогу имају унутрашње магистрале, тј. оне које се налазе у централном процесору, и повезују различите елементе унутар процесора. Размотримо најпре повезаност архитектуре рачунара са бројем интерних магистрала.

Постоје процесори који имају једну интерну магистралу. Такви процесори имају најједноставнију архитектуру. Пример рачунара, односно његовог процесора организованог око једне магистрале приказан је на слици 3.38. На слици је, ради једноставности, приказана само магистрала за податке, иако је очигледно да она може радити само заједно са интерним адресним и управљачким сигналима, који постоје, али нису нацртани јер нас у овом часу занима само ток података. Према томе једном магистралом се временски мултиплексирано (један за другим) преносе сви подаци. Са магистралама је повезан одређени број различитих регистара, као што су акумулатор, регистри опште намене $\mathbf{R_0}$ до $\mathbf{R_N}$, међурегистри, регистар стања, па аритметичко–логичка јединица. Када у процесору постоји само једна магистрала, у једном тренутку времена се може обављати само један пренос, јер

се на магистрали не може истовремено налазити више различитих по–датака.



Слика 3.38. Организација процесора око једне магистрале, типична организација микропроцесора

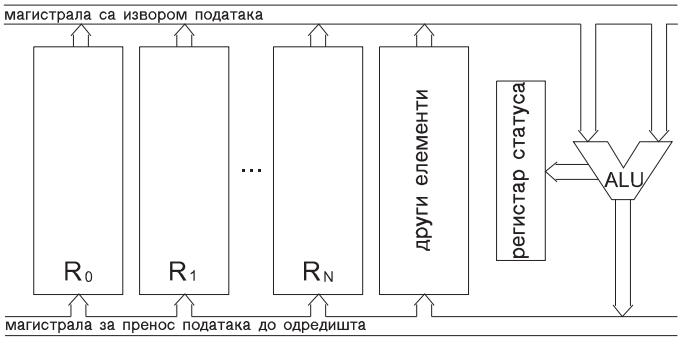
Процесор који има само једну магистралу података ради нешто спорије у поређењу са системима који имају две или више магистрала које могу, између различитих елемената рачунара, истовремено преносити по два, па чак и више података.

3.5.2. ОРГАНИЗАЦИЈА ПРОЦЕСОРА ОКО ДВЕ И ТРИ МАГИСТРАЛЕ

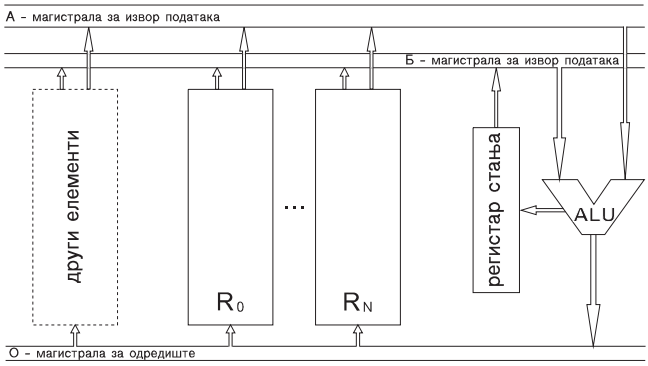
Пример процесора организованог око две магистрале дат је на слици 3.39. Једна магистрала служи за податке који долазе на улаз у **ALU** из регистра опште намене и других елемената. Резултати обраде из **ALU** долазе на другу магистралу, преко које се могу упутити на било које одредиште. Кад се употребе две магистрале, подаци се могу истовремено преносити по обе магистрале, што убрзава рад рачунарског система.

Пример организације процесора око три магистрале приказан је на слици 3.40. Сваки улаз у аритметичко–логичку јединицу има своју магистралу за улазне податке. Због тога овде, у принципу, не би ни били потребни међурегистри на улазу у **ALU**, јер би се улазни подаци могли тој јединици истовремено проследити директно са магистрале.

Већина произвођача ипак ставља међурегистре, јер су потребни ради памћења улазних података, али и из других разлога. Излазни подаци преносе се на одредиште посебном магистралом. Очито да таква органи–зација омогућује још бржи рад рачунара, јер се истовремено може преносити више података, свака магистрала преноси своје податке.



Слика 3.39. Организација процесора око две магистрале



Слика 3.40. Организација процесора око три магистрале

Овде треба напоменути да се у микрорачунарима и микропроцесорима најчешће користи једносабирничка архитектура са слике 3.38.

3.6. МЕМОРИЈЕ

Меморија је специјални хардвер намењен за смештање бинарних података (тј. упис), с циљем чувања података до момента узимања (тј. читања) података ради даље обраде. Радње уписа и читања података представљају активност која се зове **приступ меморији** (**memory access**). Што је краће време приступа то су меморије брже. Меморије нису ништа друго до одређени, обично врло велики број регистара, повезаних у једну целину. У сваки од тих регистара може се записати један бинарни податак (број), који има онолико бинарних цифара, колико је дуга реч рачунара.

За разумевање рада рачунара од основног значаја је разумевање начина меморисања програма и података, и начина проналажења жељених података и инструкција у унутрашњој меморији рачунара. По већ описаној аналогији аутоматске и ручне обраде података, меморију треба схватити као свеску, где је свака страница нумерисана. На свакој страници уцртана је табела са великим бројем врста и само две колоне, као на слици 3.41.

Прва колона садржи редни број врсте, а у другу колону се уписују подаци. При томе, у једној врсти, у одређеном тренутку, може бити уписан само један податак. Када у неко поље упишемо нови податак, претходно меморисани податак се неповратно губи. Наиме, да би уписали нови податак у табелу морамо најпре обрисати стари податак.

Слика 3.41. Коришћење свеске за складиштење података.

Ако све странице поређамо у непрекидан низ (једну испод друге), добијамо меморију у облику целовитог блока као на слици 3.42. У једну меморијску локацију уписује се једна бинарна реч. Број ботова у једној речи представља дужину меморијске речи.

Редни број врсте и странице служе за проналажење података и називају се адреса податка, или адреса локације податка. Врста у којој се налази записан податак назива се локација. Сам број који је записан на некој локацији представља податак.

Адреса локације податка, односно редни број врсте у свесци (меморији), представља физичку – ефективну адресу локације, и састоји се од две компоненте:

- *редног броја странице на којој се налази податак и*
- *редног броја врсте на страници (тј. редног броја врсте у ужем смислу), што представља неку врсту привидне – виртуелне адресе локације.*

Укупан број врста на свим страницама заједно, представља максимални број података који се уопште могу у једном тренутку записати, и назива се **капацитет меморије**.

Када је адреса локације података приказана у облику редног броја те врсте, онда је то **физичка адреса** података. Да би смо пронашли неки податак морамо знати (или некако израчунати) његову физичку адресу.

0000	1	1	1	0	0	1	0	1	0	0	0	0	0	1	1	1
0001	0	0	0	0	0	1	1	0	1	0	1	0	1	0	0	0
0002	1	0	1	0	1	0	1	0	1	0	0	1	1	1	0	1
...	...															
00FE	0	0	1	0	1	0	1	0	1	1	0	0	1	1	1	1
00FF																
0100																
0101	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0102	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
...																
01FE																
01FF	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0200																
0201																
0202																
...																
FFFE	1	1	0	0	0	1	0	1	0	0	1	1	1	1	0	0
FFFF	0	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0

Слика 3.42 Унутрашња, главна меморија рачунара

Да би себи олакшао и убрзао рад, човек при меморисању и обради великог броја података примењује различите шаблоне. Тако рецимо, податке одређеног типа, смешта увек у тачно одређене врсте, па уместо да у програм обраде уписујемо сам податак, уписујемо редни број врсте (тј. адресу), где се тај податак налази. Има још неколико разлога због којих се у програм обраде не уписује сам податак већ његова адреса.

Први разлог је, што се неки подаци током времена често мењају, (на пример, плата радника се сваког месеца мења), док су поступци обраде у једној апликацији релативно непроменљиви или се ретко мењају (на пример, увек нам треба да израчунамо просечну плату запослених и слично). Али у општем случају подаци су знатно мање променљиви него поступци обраде, па се најчешће једни те исти подаци обрађују на више разних начина у циљу добијања различитих информација.

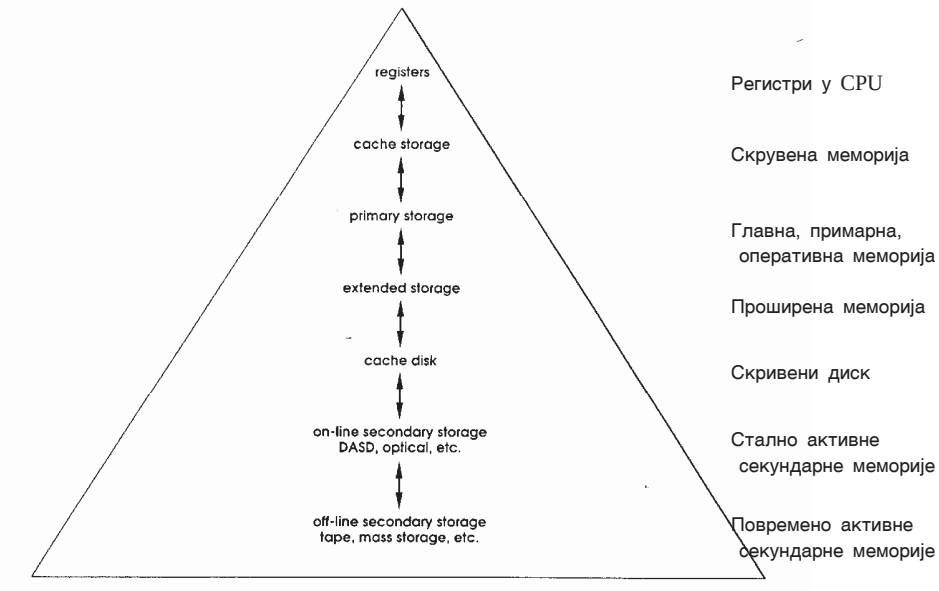
Други разлог је што, у поступцима обраде читаве групе података (велики број података истог типа), бивају обрађене на исти начин (запослених може бити од неколико појединаца до неколико хиљада, и сви се обрађују на исти начин).

Трећи разлог је што различити подаци могу имати исту бројну вредност, па би директно писање података у програм обраде, тај програм учинило: мање разумљивим, мање прегледним, мање универзалним и програм би био више подложан грешкама.

Коначно, самом човеку који је неизоставни саставни део рачунарског система, врло је тешко да дуже време памти велики број адреса података, односно много вишецифрених бројева. Човеку је зато знатно лакше да податке систематисује по неком критеријуму, и да им придружи нека **симболичка имена**, која на неки начин асоцирају на њихову природу и начин употребе (тј. обраде која се над тим податком врши).

Употреба симболичких имена података у програму обраде, уместо самих података или њихових физичких адреса, назива се симболичко адресирање. Напоменимо да сви програмски језици, изузев машинског језика, омогућавају програмерима употребу симболичких имена.

Сваки рачунарски систем поседује различите врсте меморијских уређаја. Неки су врло брзи, а други су спори. Уређаји као што су магнетни и оптички дискови имају врло велики капацитет, док нпр. регистри садрже само један бајт или реч. Капацитет и врсте меморија варирају од система до система. На слици 3.43 је дата општа хијерархијска шема расположивих уређаја за меморисање. Шема се назива хијерархијска јер су уређаји поређани у растућем редоследу одоздо на горе са аспекта брзине рада.



Слика 3.43. Хијерархија уређаја за складиштење података

Суседни меморијски уређаји на хијерархијској шеми у нормалним условима рада директно комуницирају, тј. пренос података остварује се само између два суседна нивоа. Брзина преноса података је увек у обрнутој сразмери са капацитетом, што значи да је капацитет утолико већи што је уређај даље

од врха. Видимо да у рачунарским системима постоји мноштво различитих уређаја чија је основна, али не и једина намена памћење и чување података за каснију употребу. Међу њима постоји следећа хијерархија:

- *регистри (**registers**),*
- *скривена меморија (**cache storage**),*
- *примарна, главна, оперативна меморија (**primary storage**),*
- *проширена меморија (**extended storage**),*
- *скривени диск (**cache disk**)*
- *стално активне секундарне меморије (**on-line secondary storage**), магнетни и оптички дискови који су укључени у обраду података и саставни су део рачунарског система,*
- *повремено активне секундарне меморије (**off-line secondary storage**, траке, масовне меморије).*

Регистри се налазе у CPU и самим тим су на највишем нивоу хијерархије. Они су најбржа и најскупља меморија у рачунарском систему, но уједно и најмањег капацитета. Имају различиту намену, и могу служити за памћење података чији су намена и поступак обраде унапред дефинисани у рачунарском систему (регистри специјалне намене), док неки други регистри памте податке чија употреба није унапред одређена (регистри опште намене). Израђени су у најбржим полупроводничким технологијама.

Испод регистара, јефтиније и нешто мање брзе и већег капацитета, су скривене меморије, које служе као спрега међумеморија (**buffer**) између регистара и примарне меморије. У њој се држе или копије најчешће коришћених података (нпр.VAX11/780), или блока инструкција које следе иза текуће инструкције (нпр. MC 68020). У централном процесору постоји посебан хардвер који води рачуна који подаци и инструкције се налазе у овој меморији, и када треба прибавити нове. Такође, систем мора да обезбеди да све измене података у овој меморији буду пресликане и на оригинале у примарној меморији.

Примарне меморије, још јефтиније и спорије, али још већег капацитета, садрже инструкције и податке у текућој обради, а које могу припадати или оперативном систему или неком од корисничких програма. То је главна, односно, оперативна меморија, о којој ћемо у овом поглављу говорити, и она је као и све претходне меморије изведена у полупроводничкој технологији.

У многим великим рачунарима, примарној меморији се додаје мало спорија, проширена меморија, огромног капацитета, која служи за држање података, и програма који су привремено избачени из примарне меморије. Бригу о додељивању (алоцирању) примарне и проширене меморије води део оперативног система који се зове управљање меморијом. Сличну улогу, као проширена меморија, има и скривени диск, са још већим капацитетом и још

мањом брзином рада, а задатак му је још да чува резервне копије података.

On-line секундарне меморије чине тврди дискови, измењиви дискови и оптички дискови, док **off-line** секундарне меморије, чине магнетне траке и специјалне масовне меморије које служе за дуготрајно памћење података који су на посебан начин организовани.

3.6.1. ГЛАВНА МЕМОРИЈА РАЧУНАРА

Главна, оперативна меморија се може посматрати као одређени број локација, које имају своју дужину (тј. број ботова) и своју адресу (тј. редни број). Помоћу адресе приступа се одређеној локацији са циљем уписа или читања њеног садржаја. Читање података из меморије, значи заправо копирање податка из једне локације у меморији, у неки регистар изван меморије (најчешће у неки регистар CPU). Укупни број локација које има меморија (капацитет), дужина меморијске речи и брзина приступа, важни су фактори у одређивању величине и снаге рачунарског система.

Капацитет оперативне меморије је реда 1,2,4,8,16,32 милиона меморијских речи, мада су већ у употреби и оперативне меморије са више од 100 мегабајта (Megabyte). Дужина меморијске речи зависи од врсте рачунара и варира, од 8, 16 или 32 бита до више од 100 бита, а најчешће представља умножак од 8 ботова тј. 1 бајта. Једна меморијска реч обично садржи 4 бајта, а дупла реч 8 бајта тј. 64 бита, мада то зависи од типа и произвођача.

Када број локација помножимо са дужином речи, добијамо укупан број меморијских елемената које садржи оперативна меморија. Ове меморије су се некада правиле од магнетних језгара, а данас се искључиво праве у полупроводничкој технологији, која је знатно бржа и није више тако скупа.

Брзина рада меморије је један од одлучујућих фактора брзине рачунара. Она се може окарактерисати на више начина, а један је време приступа, тј. време потребно да се добије садржај неке локације након постављања њене адресе.

По овом критеријуму разликујемо две врсте меморија: секвенцијалне меморије и меморије са директним приступом.

Код секвенцијалних меморија време приступа зависи од локације где је податак меморисан. Главни представници ових меморија су: магнетне траке, магнетни мехурићи и меморије са електричним набојем (CCD), и углавном се користе за реализацију секундарних меморија.

Време приступа код меморија са директним приступом је увек исто, и не зависи од локације податка. У ову групу меморија спадају **RAM** (**Random Access Memory**) и **ROM** (**Read Only Memory**) меморије, и данас су то по

правилу полупроводничке меморије. RAM меморије губе свој садржај након искључења напајања, али се у њих подаци могу и уписивати и из њих читати. У ROM меморије се не могу поново уписивати подаци (већ се само читају), али при искључењу напајања не губе свој садржај. Негде између ове две групе, налазе се разне врсте дискова, који се у литератури сматрају, или меморијама са директним приступом, или меморијама са полудиректним приступом. Код неких типова меморије, приликом читања, податак се избрише из меморије, па га је потребно изнова записати. Овај процес се назива меморијски циклус (**memory cycle**), а време потребно за ту операцију зове се временски циклус (**cycle time**). Овакве меморије се називају деструктивне меморије.

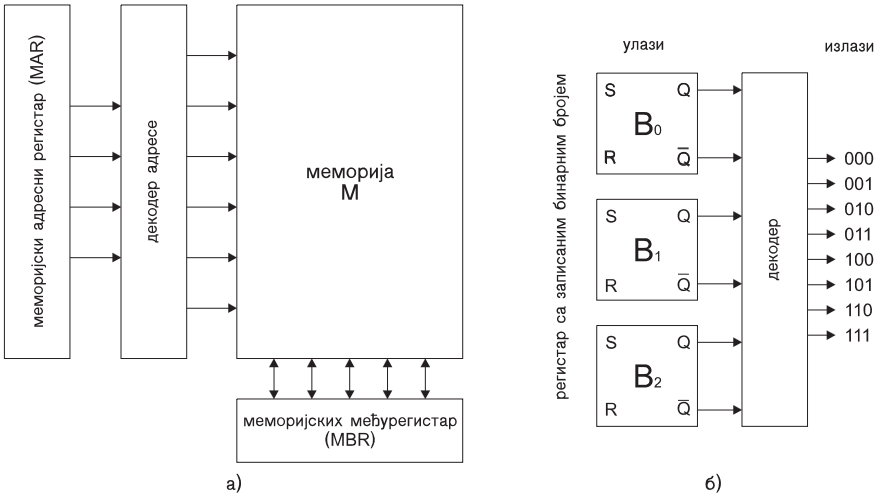
Такође постоје две врсте RAM меморија. Статичке полупроводничке RAM меморије, које се праве помоћу флип–флопова, које задржавају свој садржај и после читања (недеструктивне), све до поновног записа, или искључења напајања. За разлику од њих, динамичке RAM меморије (**DRAM, Dynamic RAM**) се реализују као капацитивност **MOS** транзистора, и као и сви кондензатори и ови временом губе набој, па се повремено (на пример, сваке мили секунде) морају освежавати, тј. поново се уписују стари садржаји. Динамичке меморије су деструктивне, па се и после читања, мора вршити поновни упис прочитаног податка. Данас динамичке RAM меморије имају огроман капацитет и брзину. Наиме, већ постоје DRAM меморије са 600 **MBps** (шесто мега бајта у секунди), а очекују се **nDRAM** меморије са 1,6 **GBps** (гигабајта у секунди). Захваљујући развоју нових технологија RAM меморије више нису тако скупе (3–10 **$/MB**), па данашњи рачунари користе оперативне меморије врло великог капацитета. Данас чак и микрорачунари имају преко 500 **MB** RAM–а, а код **mainframe** рачунара она се креће и преко 100 **GB**.

На слици 3.44. а) приказана је упрошћена блок шема меморије. Да би било могуће читање и упис података, у меморији морају постојати још неки склопови, који са самом меморијом чине јединствену функционалну целину. Један од њих је регистар у који се записује адреса локације којој се приступа. Он се обично назива меморијски адресни регистар (**MAR, memory adress register**).

Други регистар у који се привремено смешта податак који се уписује, или је прочитан из локације која је адресирана, је прихватни регистар података тзв. меморијски међурегистар (**MBR, memory buffer register**).

Трећи склоп је декодер адресе. Овај склоп, на бази бинарног кода адресе меморијске локације, врши селекцију те меморијске локације. То је декодер типа један од 2^n, где је **n** број ботова у адреси. На слици 3.44. б) је приказан декодер један од 2^3, којим се на бази тробитног улаза изабира један од 8 излаза. Четврти саставни део сваке оперативне меморије је управљачки меморијски склоп, којим се бира врста приступа (упис–читање).

Овај склоп није приказан на слици 3.44. јер његова веза са осталим деловима није тако очигледна. Склоп утиче на смер преноса података између меморијског елемента и прихватног регистра података.



Слика 3.44. Основни саставни делови меморије

Код меморије великог капацитета декодер, један од 2^n, може бити исувише сложен, па су могуће и друкчије организоване меморије. Пре разматрања разних типова организације RAM и ROM меморија, још једном морамо истаћи да се подаци у рачунару памте у облику низа бинарних цифара **0** и **1**, односно као бинарни бројеви. Меморија служи само за памћење података и ови се могу читати и евентуално уписивати, тј. у меморији се никада не врши никаква обрада. Обрада се врши само у централном процесору.

Програми, тј. низови инструкција имају исти облик, као и подаци, када се налазе у рачунару. Наиме, и инструкције се такође кодирају помоћу цифара бинарног бројног система, тако да су у рачунару такође представљене као бинарни бројеви. **John von Neumann** је дошао на идеју да се подаци и програми смештају у исту оперативну меморију. Ово је тзв. Нојманова кон–фигурација рачунара. Када су инструкције уписане у оперативну меморију рачунара, оне се могу третирати на исти начин као подаци. На овај начин је створена могућност да сам рачунар изврши превођење програма, написаног у програмском језику који рачунар не разуме (на пример, у **FORTRAN**–у), у програм еквивалентан по функцији, али написан у језику који рачунар ра–зуме (нпр. у машинском језику). Но, третирање инструкција као података је и опасно, па оперативни систем рачунара мора строго водити рачуна о томе где су у меморији записани подаци, а где инструкције, јер централни

процесор врши обраду инструкција сасвим друкчије од обраде података. То је део оперативног система који се зове механизам заштите меморије.

3.6.2. РАЗЛИЧИТИ ТИПОВИ ОРГАНИЗАЦИЈА МЕМОРИЈЕ

Памћење података у меморијама увек се своди на памћење нула и јединице. То се може остварити на различите начине. Основни елемент меморије, назовимо меморијска ћелија, бележи и памти једну нулу или јединицу, односно један бинарни бит. Меморије морају бити у стању да бележе велики број битова повезаних у бајтове и речи. Да би се меморија могла употребљавати, мора постојати могућност да се подаци уписују у меморију и да се читају из ње. То значи да елементарне ћелије за памћење појединачних битова треба на одговарајући начин повезати тако да могу испуњавати задатке који се од њих траже.

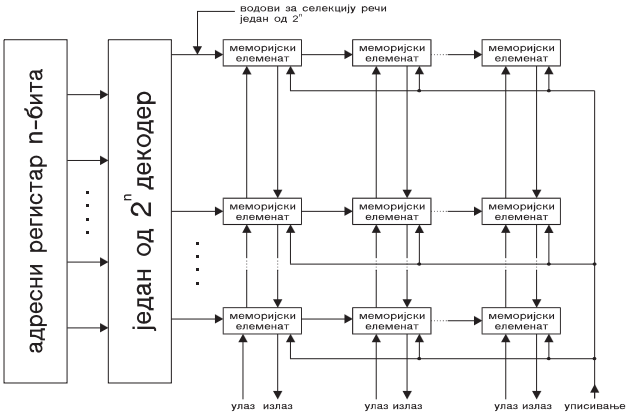
Елементарне меморијске ћелије могу бити повезане на различите начине или, како се друкчије каже, меморије могу бити организоване на различите начине. Тако постоје дводимензионалне и тродимензионалне меморије, и меморије организоване као стек.

Осим таквих паралелних организација меморија код којих се одједном може уписивати или читати цела једна реч или бајт и то са случајним приступом, постоје и серијске меморије код којих се подаци крећу један иза другог, па се битовима може приступити само одређеним редоследом, а не случајно. Размотрићемо карактеристичне типове организација меморија, и то прво дводиманзионалне, а затим тродиманзионалне меморије, стек–меморије и серијске померачке меморије.

3.6.2.1 Дводимензионалне меморије

Како изгледа организација дводимензионалних меморија приказано је на слици 3.45. Меморијски елементи (ћелије) поређани су један до другог тако да низ таквих хоризонтално поређаних елемената (један ред) може да значи једну реч. Меморијски елементи сврстани у вертикалне колоне чине битове исте тежине у различитим речима. Тако прва колона слева може записивати податке 2^0, друга колона 2^1, трећа 2^2 и последња 2^{m-1}, где је **m** дужина употребљене речи.

Адресни регистар од **n**–битова може имати 2^n различитих стања. Свако такво стање декодер претвара у стање **1** само на једној од 2^n излазних линија које служе за селекцију адресиране меморијске локације са које се чита или у коју се уписује.



Слика 3.45. Организација дводимензионалне меморије

У једном тренутку времена, линијама за селекцију може се активирати само један хоризонтални ред меморијских елемената који чине једну реч. Остали хоризонтални редови нису активни и не учествују у операцији читања или уписивања. Када се генерише одговарајући сигнал на линији за уписивање, податак доведен на улазне линије уписују се у одговарајуће ћелије селектоване речи. При читању стања одговарајућих меморијских елемената селектована реч се појављују на излазним линијама. Сигнал за читање обично је комплемент сигнала за уписивање, тј. стање 1 на воду за уписивање омогућује упис података у меморијски елемент, а при стању 0 на тој линији појављују се подаци на излазним линијама.

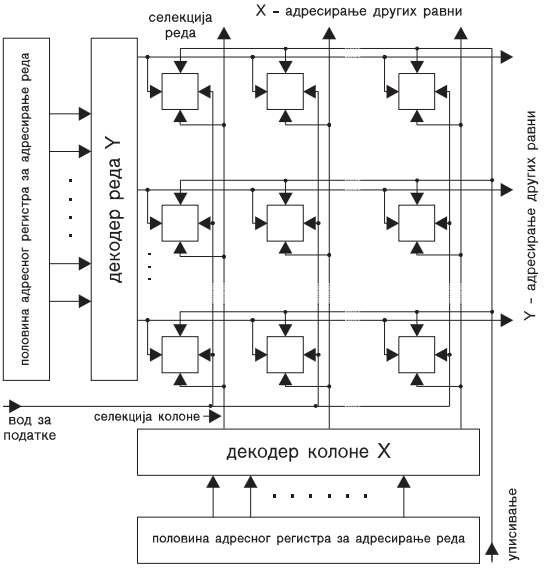
Једну димензију такве меморије чине адресе локација (на слици 3.45 вертикално), а другу димензију дужина речи (на слици 3.45 хоризонтално). Због тога је називамо дводимензионалном меморијом (а неки то називају и једнодимензионалним, линеарним адресирањем). Међутим, треба истаћи да је таква организација прилично непрактична, јер треба имати онолико селекционих линија колико меморија има речи. Колико је то линија, довољно је рећи да за LSI–меморије (меморије високог степена интеграције, **Large Scale Integration**) од 64**K** бита уз 8–битне речи, треба више од 8000 линија. Треба рећи још и то да се капацитети LSI–меморија обично изражавају у броју битова, а не бајтова. Тако LSI–меморија од 8**K** бита може приказати 1**K** бајт, а она од 64**K** бита представља 8**K** бајта.

3.6.2.2 Тродимензионалне меморије

Дводимензионална меморија мора имати велики број линија за селекцију речи, односно онолико линија колико има речи. То значи да уз 256_{10}, 1024_{10} или 4096_{10} употребљених речи, мора имати исто толико селекционих линија.

Толики број селекционих линија представља проблем, па се настоји да се њихов број смањи употребом тродимензионалне меморије. То се постиже тако да се селекција поједине меморијске ћелије не изврши до краја у посебном спољњем декодеру, већ се део селекције изводи и у самој меморијској ћелији. Под спољном селекцијом подразумева се све оно што се налази изван меморијских ћелија, иако може бити на истом чипу.

Најједноставнији начин таквог адресирања појединих ћелија меморије јесте кад се адресни регистар и декодер поделе на два дела, обично на две половине, једна за хоризонтално адресирање, а друга за вертикално. Свака половина одабере по једну адресну линију у складу са податком који је записан на одговарајућој хоризонталној, односно вертикалној половини адресног регистра. Адресира се само она меморијска ћелија која се налази на пресеку одабране хоризонталне и вертикалне линије. То се изводи тако да се у сваку меморијску ћелију дода **И** коло које одабере меморијску ћелију само онда кад су одабране и хоризонталне и вертикалне линије које долазе из спољњег декодера. Како се то може извести показано је за једну раван тродимензионалне меморије на слици 3.46.



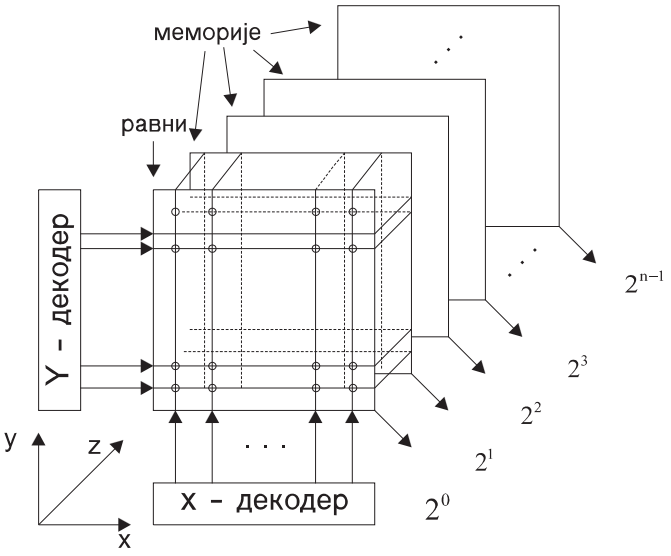
Слика 3.46. Приказ једне равни тродимензионалне меморије

Половина адресе намењена је адресирању реда ($\mathbf{Y=n/2}$), а половина адресирању колоне ($\mathbf{X=n/2}$). Одабере се ћелија која се налази на пресеку одабраних линија $\mathbf{X}$ и $\mathbf{Y}$. Само се у ту меморијску ћелију може уписати податак

кад се он при постојању сигнала за уписивање налази на линија за податке. Исто се тако из те ћелије може прочитати податак, и он се такође појављује на линији за податке. Размотримо сада колико декодерских линија треба за дводимензионалну, односно тродимензионалну меморију. Код дводимензионалне меморије, за **n** адресних битова на адресном регистру потребно је 2^n излазних адресних линија из јединственог декодера. Код тродимензионалне сваки декодер има $2^{n/2}$ линија, а оба заједно имају $2 \cdot 2^{n/2} = 2^{n/2+1}$, тј. приближно двоструко мање.

Меморију која је овде названа тродимензионалном неки називају и меморијом са дводимензионалним адресирањем, односно меморијом са коинцидентним адресирањем. Код тродимензионалне меморије адресе чине две димензије, а дужина речи трећу димензију. Да би се организовала таква тродимензионална меморија треба онолико равни меморије (као што је она на слици 3.46), колико је битова дуга реч која се памти.

У равни тродимензионалне меморије приказаној на слици 3.46, адресира се само једна меморијска ћелија. Она представља један бит (нпр. нулти бит) адресиране речи, а све остале меморијске ћелије у тој равни представљају по један бит различитих речи. То значи да при таквој организацији меморије у једној равни има онолико једнобитних речи колико има битова у тој равни. Према томе, да би се добило више битова тј. меморијска реч, треба употребљавати онолико равни меморије колико су дуге речи које треба памтити. Организација такве меморије приказана је на слици 3.47.



Слика 3.47. Организација тродимензионалне меморије

Дужина речи представља трећу димензију меморије, а битови једне меморијске речи налазе се један испод другог у разним равнима меморије. При томе декодиране адресне линије из прве равни прелазе у другу, затим трећу, итд., па је, без обзира на број равни, потребан увек исти број линија за адресирање. Подаци се читају и уписују на линијама за податке у свакој равни. Тако се у нултој равни уписује 2^0, у првој 2^1, у другој 2^2, трећој 2^3, итд. На слици 3.47, због једноставности и прегледности, приказане су само неке важније линије, а остале су изостављене. Малим кружићем означена је основна меморијска ћелија, која је осим памћења бинарног податка обавља и додатну логичку **И**–операцију, јер је адресирана само онда када постоји логичка јединица и по линији (X) и по линији (Y) које пролазе кроз кружић.

3.6.2.3 **Меморије организоване у стек**

Дводимензионална и тродимензионална организација меморија омогућују случајан приступ до сваког податка. То значи да се било који податак може читати или уписивати независно од тог који се податак читао или уписивао пре тога. Због тога се такве меморије и називају меморијима са **случајним приступом**, без обзира на то да ли се ради о меморијама **RAM** или **ROM**.

Но, постоје и такве организације меморија гда се подаци могу уписивати или читати само по неком реду, па то нису меморије са случајним приступом. Једна од таквих је и стог или стек меморија (**stack**).

Меморија организована у стек састоји се од низа регистара, од којих се сваки састоји од одређеног броја меморијских елемената (ћелија). Такав низ регистара, сложених један на други као стог, за разлику од меморија са случајним приступом, има доступан само онај регистар који је на врху стога. Подаци се умећу на врх стога, а са врха се и ваде. На слици 3.48 а) приказано је једно могуће стање стека.

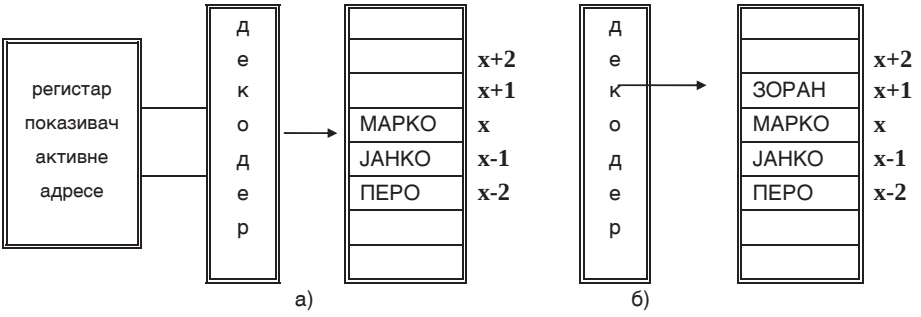
Слика 3.48. Стек меморија и операције са стеком

Функционисање стека може се упоредити са ситуацијом кад се на радном столу на гомилу слажу папири које треба решити (обрадити). Нови папири увек се стављају на врх гомиле, а онај ко решава те папире увек узима папир само са врха гомиле. Кад узме један папир и реши га, премести га на друго место, или баци. На врху стека тада се нађе папир који је био испод претходног, итд. Такав начин рада зове се "**последњи ушао први излази**", или **LIFO**, (**Last In First Out**).

Ако се уметне нова реч, онда се све речи које су пре биле у стеку помакну за један корак наниже, а нова уђе на врх. На слици 3.48. б) види се стање након уметања нове речи ЗОРАН. Ако се извади једна реч (од почетног стања приказаног на слици 3.48 а)), добије се ситуација као на слици 3.48. ц). При уписивању речи (слика 3.48. д)) брише се пређашња реч МАРКО и уместо ње се може уписати, рецимо., реч СОЊА. При уписивању се подаци у осталим регистрима не мењају, тј. реч ЈАНКО и ПЕРО остају на својим пређашњим местима. При читању се прочита реч с врха стека, дакле, реч МАРКО, слика 3.48. е). Читање и уписивање податка исто је као у обичној RAM меморији рачунара, док су операције уметања и вађења податка својствене само стеку и не постоје код обичне меморије. Обзиром да су подаци записани у бинарном облику и да је у питању померање података, најједноставнији начин реализације стека, јесте употреба померачких регистара (**shift register**).

Бит најмање тежине у свим регистрима заједно представља један померачки регистар, а следећи бит је други померачки регистар, итд. При реализацији стека на такав начин подаци се померају по регистрима. Такав начин реализације у начелу је сличан стеку приказаном на сликама 3.48.

Постоји и друга могућност реализације стека, тако да се не помера податак, него се мења адреса која одговара врху стека. Такав начин реализације стека приказан је на слици 3.49. При тој реализације активна адреса стека, тј. адреса која одговара претходном врху стека, помера се при уметању, односно вађењу податка.



Слика 3.49. Реализација стека померањем адресе (софтверски стек)

Она у појмовном смислу и даље чини врх стека, мада је боље да се зове активна адреса, јер врх стека у том случају може бити и на доњој страни. Ако, дакле, садашња активна адреса показује локацију **x** у којој је записана реч МАРКО, а на нижим положајима су речи ЈАНКО и ПЕРО, као на слици 3.49 а), онда ће се при стављању нове речи ЗОРАН (слика 3.49 б)), активна адреса најпре померити на положај **x+1**, па ће се затим у њу записати уметнута реч. Ако се жели извадити реч МАРКО, онда се она прочита и избрише, а затим се активна адреса помери на положај **x-1**.

3.6.3. МЕМОРИЈА ROM

Меморија **ROM** или **ROS** (**Read-Only Store**) испуњава два захтева:

– *неизбрисивост (**non-volatibility**),*

– *недеструктибилност–неуништивост садржаја (**non-destructive readout**).*

Неуништивост значи да се ROM појављује као меморијско поље чији је садржај, једном уписан, сталан и не може се променити под утицајем процесора (операцијом уписивања). Због својих карактеристика ROM се обично употребљава за меморисање сталних програма.

У групу исписних меморија, тј. меморија које се могу само читати, спадају:

а) ROM – искључиво исписна меморија,

б) PROM – програмабилна исписна меморија,

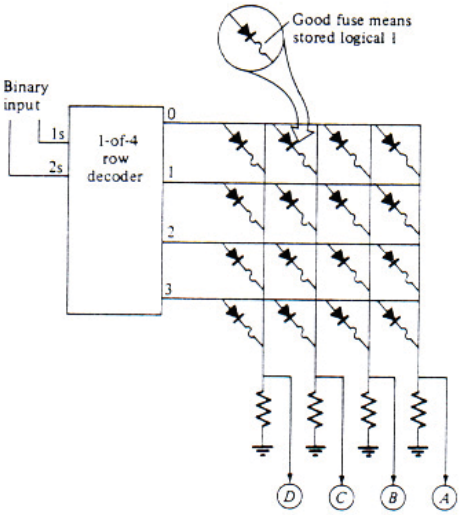
ц) EPROM – променљива програмабилна исписна меморија,

д) EEPROM (EAROM)-електрично програмабилна исписна меморија.

а) У **ROM** је уписан одређени садржај већ за време израде меморијског чипа. Бит–узорке који одговарају жељеном програму корисник доставља произвођачу у стандардном облику (нпр. MOTOROLA захтева папирну траку као резултат употребе **MC6800** програмског пакета или хекса–декадни код на бушеној картици **IBM**). Произвођач израђује одговарајућу маску (према приложеном програму корисника) и као задњи корак у производњи чипа ROM врши метализацију, односно успоставља везе између редова и колона. Минимални број **ROM**–ова који се на тај начин могу произвести одређује произвођач, а креће се око броја 1000. Време приступа за **ROM** у технологији **MOS** је 500 до 850 ns.

б) **PROM** (**user-programmable read only memory**) је тип исписне меморије коју може испрограмирати сам корисник уз помоћ уређаја за про–грамирање PROM–ова. У меморије PROM спадају два типа програма-билних ROM–ова: биполарно поље диода и биполарна транзисторска конфигурација. Оба ова типа примењују тенику "прегорљивих веза" (**fusi-ble links**), тј. метализованих веза између базе и емитера или у PN споју диода, слика 3.50. За време програмирања уређај генерише низ импулса

којим ће се изабране "прегорљиве везе" растопити и тиме прекинути везу у матрици између колоне и реда. PROM има кратко време приступа – мање од 100 ns и употребљава се код већине микрорачунара израђених у малим серијама за смештање програма. Недостатак PROM–а сличан је недостатку ROM–а – када је једном испрограмиран, не може се мењати његов садржај.



Слика 3.50. Диодни PROM

ц) **EPROM** је меморија коју може програмирати корисник уз помоћ (E)PROM–програмера, али се њен садржај може избрисати и затим поново испрограмирати. EPROM се брише осветљавањем чипа ултраљубичастим зрацима у трајању од пет до десет минута. Садржај свих меморијских локација се брише. Након брисања, EPROM може бити поново испрограмиран. Цена EPROM–а је релативно висока. Време приступа је између 150 до 1200 наносекунди. За израду меморија EPROM употребљава се технологија MOS. Један тип вишеструко програмибилног PROM–а је RPROM, који се брише електрички.

д) **EEPROM** или EAROM (**electrically alterable ROM**). Као алтернативно решење проблема избрисивости наведен је и EEPROM, обично сврстан у разред меморија које се углавном читају (**read-mostly memory, RMM**); међутим, EEPROM омогућује и уписивања. Операција уписивања тражи време реда величине милисекунде, док су операције читања реда микросекунде. Очито је да се такве меморије не могу употребљавати као класичне меморије са директним приступом. Меморија EAROM се употребљава тамо где је нужно смештање малог броја података или параметара. Има мали капацитет, неколико десетина **KB** (кило бајта).

3.6.4. МЕМОРИЈЕ СА ВИШЕ МОДУЛА И ПРЕКЛАПАЊЕ

Главна меморија рачунара најчешће представља колекцију већег броја физички одвојених модула. Сваки од њих има свој меморијски адресни регистар (**MAR**) и меморијски бафер регистар (**MBR**) тако да је могуће постићи да два и више модула истовремено извршава операцију уписа (**write**) или читања (**read**). На тај начин се може постићи знатно скраћење средњег времена приступа меморијским локацијама.. Да би се то постигло потребно је имати додатне управљачке уређаје што поскупљује хардвер, па се ови системи обично користе у великим рачунарима. Могућа су два приступа:

- **први**, када се узастопне локације (речи) налазе у истом меморијском модулу. Овај метод омогућава, рецимо, да једном модулу приступа централни процесор ради уписа или читања податка (или инструкције), док је други модул под контролом, рецимо, **DMA** контролера и извршава пренос података у или из улазно–излазних јединица. За селекцију меморијског модула користи се **k** битова највеће тежине (у ефективној меморијској адреси).
- **други**, када се узастопне локације (речи) налазе у узастопним (разним) меморијским модулима. Обај метод се назива преплитање меморија (**memory interleaving**), и омогућава централном процесору (или **DMA** контролеру) да истовремено приступи већем броју меморијских локација. Да би систем био врло ефикасан процесор мора имати могућност да "предвиди" који ће му податак бити ускоро потребан и да унапред пошаље захтев меморији. (**prefetch**). То је случај када се обрађују низови података (у узастопним локацијама). Сем тога, потребно је да број меморијских модула буде $\mathbf{r}=2^{\mathbf{k}}$. За селекцију меморијског модула у коме је жељени податак користи се **k** битова најмање тежине. Ово може теоријски скратити време прибављања података за 1/**r** пута (где је **r** број меморијских модула). Ове погодности се губе при извршавању наредби условног или безусловног скока, јер се тада управљање преноси на неку другу инструкцију којој су најчешће потребни подаци из неког другог дела меморије.

3.7. ЗАКЉУЧАК

У овом поглављу проучене су логичке основе рачунара као и елементарна логичка кола која се користе у реализацији рачунара. Комбиновањем основних логичких кола могу се реализовати различите функционалне логичке мреже као што су: кодери, декодери, мултиплексери, демултиплексери, полисабирачи и сабирачи.

Елементарна логичка кола се такође користе у реализацији основних меморијских склопова за меморисање једног бита, као што су разне врсте флип–флопова. За меморисање података дужине речи, користе се регистри који имају онолико флип–флопова колико реч има ботова. За упис, и читање података из регистара користе се одговарајући управљачки сигнали које генеришу управљачке логичке мреже. Комбиновањем више флип–флопова праве се и друге дигиталне мреже као што су бројачи и померачки регистри.

За меморисање већег броја бинарних података користе се хардверски склопови који се зову меморије, за чију реализацију се могу користити различити медији и које се могу организовати на разне начине.

ALU и регистри унутар централне процесорске јединице, повезани су међусобом посредством специјалних линија које допуштају истовремено присуство већег броја дигиталних кола, и која се зове интерна магистрала. Веза CPU са меморисјким јединицама и улазно–излазним уређајима, такође је остварена посредством магистрала, за чије функционисање је такође потребно имати одређене управљачке сигнале.

3.8. ПИТАЊА

1. Који су основни саставни делови централног процесора?
2. Шта је логичка основа рада рачунара?
3. Навести аксиоме Булове алгебре.
4. На коју особину логичких функција указују основне теореме Булове алгебре?
5. Које су основне логичке операције?
6. Шта је то таблица истинитости?
7. Нацртати шему којом се реализују **И**, **ИЛИ** и **НЕ** операција помоћу **НИЛИ** кола.
8. Нацртати шему којом се реализују **И**, **ИЛИ** и **НЕ** операција помоћу **НИ** кола.
9. Објаснити минимизацију логичких функција помоћу Карноових мапа.
10. Да ли стање на излазу R/S флип–флопа зависи само од тренутне вредности сигнала на S и R улазу?
11. У чему је разлика у начину рада D и R/S флип–флопа?
12. Шта на свом излазу даје бинарни полусабирач?
13. У чему је разлика између полусабирача и потпуног бинарног сабирача?
14. Који склопови улазе у састав аритметичко–логичке јединице?
15. Које су основне врсте регистара и која им је намена?
16. Која је намена основних регистара у саставу CPU?
17. Шта је предност повезивања регистара преко магистрала?
18. Како све могу бити организовани процесори?
19. У чему је разлика између адресе локације и њеног садржаја?
20. Зашто се у програмима обраде користе симболичке адресе тј. симболичка имена података, а не сами подаци?
21. Описати хијерархију уређаја за меморисање.
22. Шта је секвенцијални а шта директни приступ меморијским локацијама?
23. Који су основни саставни делови главне меморије рачунара?
24. Описати разлике у организацији дводимензионалних и тродимензионалних меморија.
25. Шта је стек меморија и како се може реализовати?
26. Шта су ROM меморије, и које врсте постоје?

3.9. КЉУЧНЕ РЕЧИ

- **ALU**, аритметичко–логичка јединица (**Arithmetic and Logic Unit**)
- Булова алгебра
- **D** флип–флоп (**data flip-flop**)
- динамичка RAM меморија (**Dinamic RAM, DRAM**)
- декодер (**decoder**)
- деструктивна RAM меморија (**DRO, destructive readout**)
- дигитално коло (**digital circuit**)
- директни, случајни приступ (**direct access, random access**)
- дводимензионална меморија
- **ексклузивно**, **искључиво ИЛИ**, сабирање по модулу два (**XOR**)
- физичка адреса, ефективна адреса (**efective address**)
- флип–флоп (**flip-flop**)
- **И** операција, конјункција, логичко множење (**AND**)
- **ИЛИ** операција, дисјункција, логичко сабирање (**OR**)
- интегрисано коло (**IC integrated circuit**)
- кодер (**coder**)
- коинцидентно адресирање
- линеарно адресирање
- логичка операција
- логички податак (**logical data**)
- **LSI,** технологија високог степена интеграције (**large-scale integration, LSI**)
- магистрала, сабирница (**bus**)
- меморија (**memory, storage**)
- меморијска адреса (**memory address**)
- **НЕ** операција, комплементирање, инвертовање, негација (**NOT**)
- **НИ** операција, Шеферова функција **(NAND**)
- **НИЛИ** операција, Пирсова функција (**NOR**)
- полусабирач (**half adder**)
- померачки регистар (**shift register, shifter**)
- преплитање меморија (**memory interleaving**)
- примарна, главна, оперативна меморија (**primary storage, operating memory**)
- проширена меморија (**extended memory**)
- **R/S** флип–флоп (**set-reset flip-flop**)
- RAM меморија (**Random Access Memory**)
- регистар (**register**)
- релативна адреса (**relative address**)
- ROM меморија (**read only memory**)
- **ROS** меморија (**Read Only Store**)
- сабирач, потпуни сабирач (**full adder**)
- секундарна меморија, масовна меморија (**secondary storage, mass memory**)
- секвенцијални приступ (**sequential access**)
- симболичко адресирање (**simbolic addressing**)
- скривена меморија, скривени диск (**cache memory, cache disk**)
- статичка RAM меморија (**Static RAM**)
- тродимензионална меморија
- заставица (**flag**)

4. АРХИТЕКТУРА РАЧУНАРА
ПРИБАВЉАЊЕ ИНСТРУКЦИЈЕ И ИЗВРШАВАЊЕ

Међусобно повезивање и координација рада дигиталних кола, магистрала, регистара, аритметичко–логичке јединице и других склопова у саставу рачунара, остварени су помоћу програма на микропрограмском нивоу и на нивоу машинског језика, као што је то приказано на хијерархијском моделу слика 1.5. Ова два нивоа садрже основне програме који су способни да управљају радом дигиталних мрежа тако да ове мреже обаве одређени користан рад.

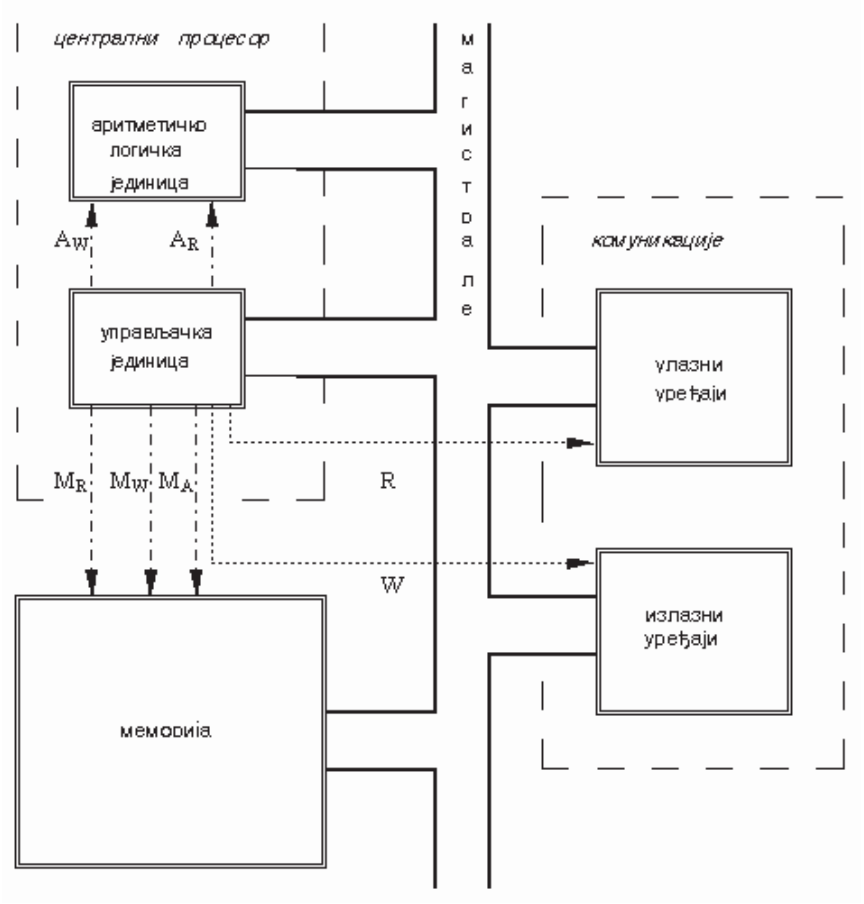
Програми на машинском језику и микропрограми, који координирају рад дигиталних логичких кола, компоновани су као низови инструкција од којих је свака појединачно приказана као неки бинарни број. Током извршавања неке инструкције, њени битови служе као улазни сигнали у логичка кола која генеришу контролне и управљачке сигнале, извршавају аритметичке или логичке операције, указују на локације у меморији итд. На овом нивоу постоји низ специфичних детаља који варирају од рачунара до рачунара и од произвођача до произвођача. Но, генерална идеја је мање више иста. Ради лакшег разумевања у разматрању ових нивоа, кренућемо обрнутим редоследом, тј. прво ћемо посматрати шта се догађа у рачунару на нивоу машинског језика, а потом на нивоу микропрограма.

4.1. ПОЈЕДНОСТАВЉЕНА АРХИТЕКТУРА РАЧУНАРА

Појам архитектуре рачунара означава главне саставне делове рачунара и њихову повезаност у једну функционалну целину. На слици 4.1 приказ ане су главне компоненте архитектуре типичног рачунара које чине: процесор, меморија и комуникациони систем.

Улазне и излазне јединице омогућавају повезивање рачунара са спољашњим светом из којег рачунар добија податке и захтеве за обраду, и коме рачунар шаље резултате обраде и информације. Ових јединица може бити више у једном рачунарском систему.

Меморију представљају различити електронски склопови који се налазе унутар рачунара, а у којима се налазе програми и подаци који се управо обрађују (текући, актуелни програм и подаци). Ова меморија се зове оперативна или главна меморија рачунара. Додатна меморија, коју чине разни регистри, налази се унутар централне процесорске јединице.



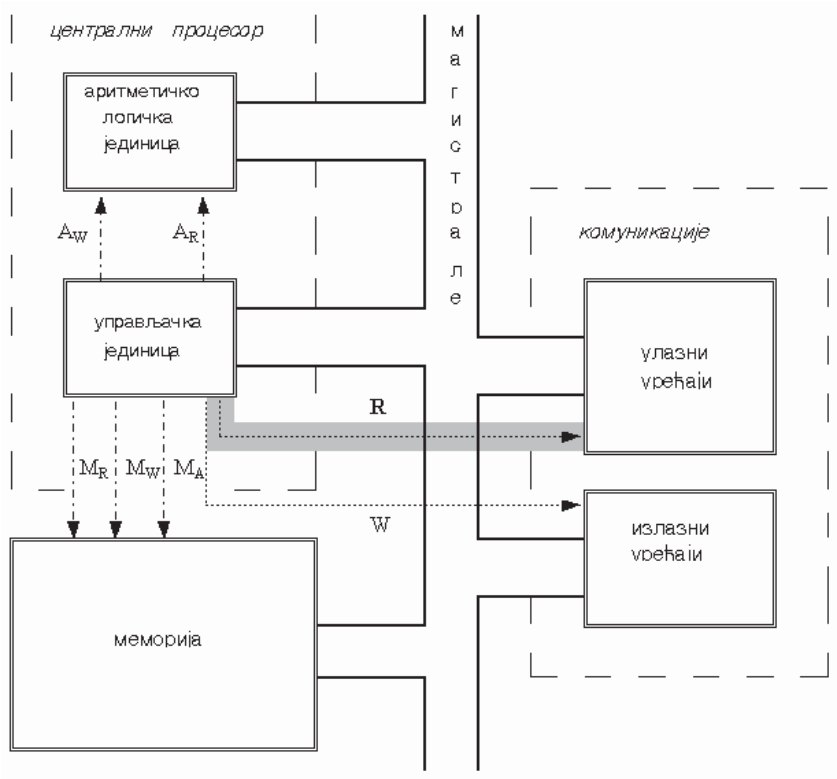
Слика 4.1. Генералисана архитектура рачунара

Главна компонента архитектуре рачунара је централна процесорска јединица (CPU), која се састоји од управљачке јединице (**control unit**), и аритметичко логичке јединице (ALU). У састав ове јединице улази и одређени број регистара специјалне и опште намене које, за сада, нећемо помињати, нити користити у почетним разматрањима. Пуним линијама на овој слици су представљене магистрале података и адреса, а управљачки сигнали су приказани испрекиданим линијама.

Централна процесорска јединица садржи дигитална кола која прихватају и извршавају програмске инструкције. Извршавање било које инструкције има за последицу генерисање (од стране управљачке јединице) читавог низа управљачких сигнала. Ови сигнали контролишу и синхронизују рад свих делова рачунара.

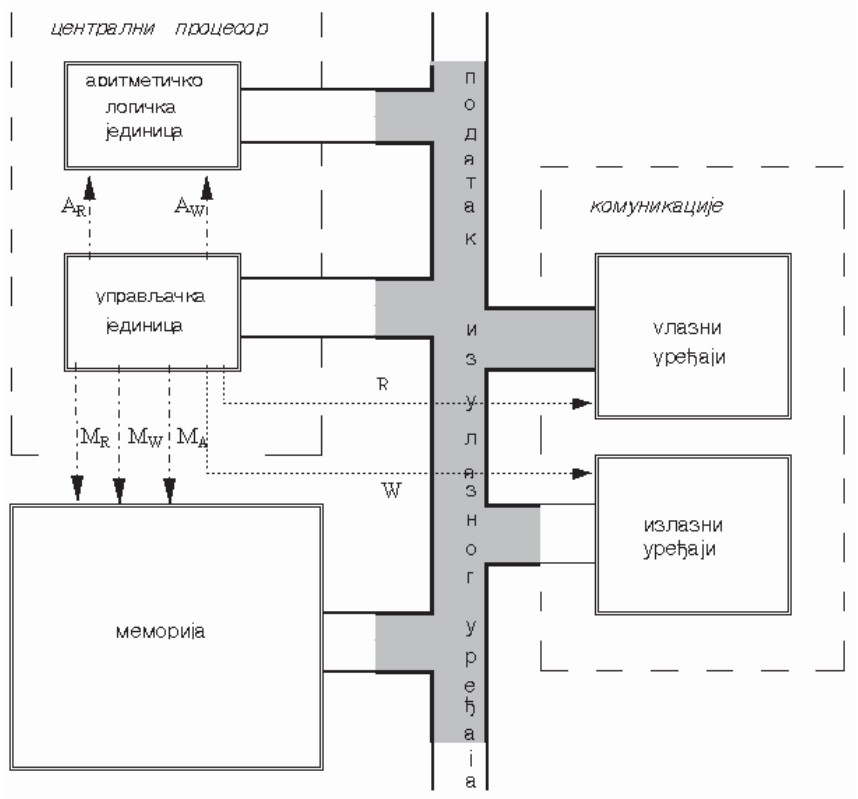
4.1.1. УЛАЗ ПОДАТАКА

Када централном процесору треба податак из спољашњег света, он улазној јединици пошаље захтев за унос податка, односно за његово записивање на магистралу. Овај захтев се иницијализује постављањем управљачког сигнала на линију **R** (слика 4.2), и након тога централни процесор чека да та улазна јединица одговори, односно изврши постављени задатак.



Слика 4.2. Улазно коло је активирано управљачким сигналом **R** из управљачке јединице

Под дејством управљачког сигнала R, улазна јединица копира потребан податак из неког свог регистра на магистралу података, слика 4.3.

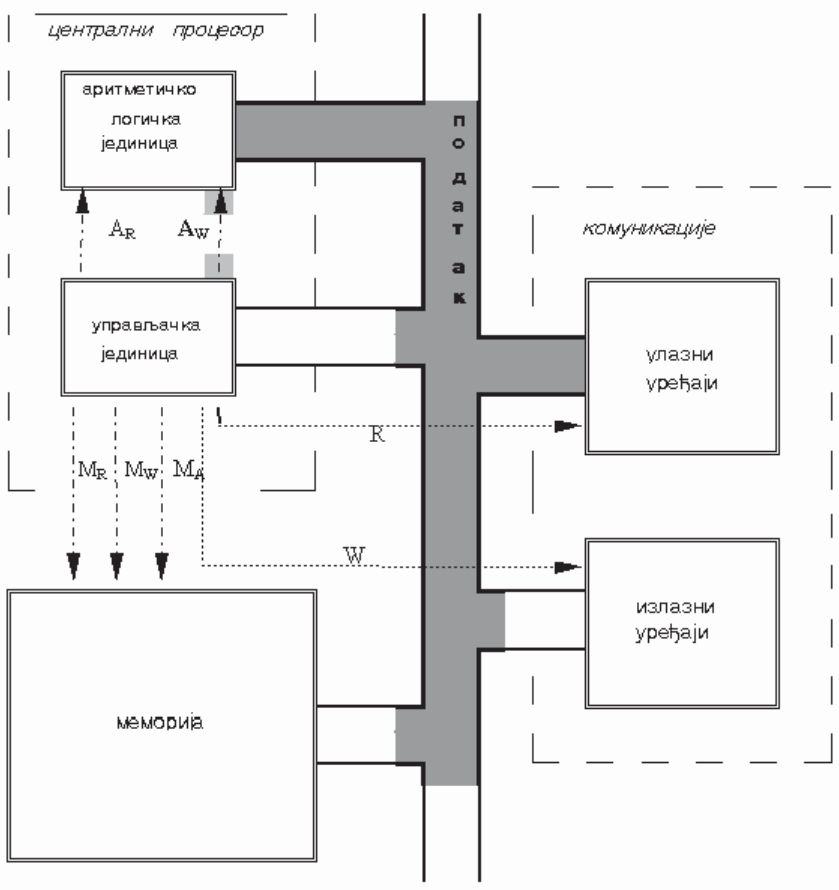


Слика 4.3. Након што је улазно коло добило управљачки сигнал, оно поставља податак на магистралу

Када је податак постављен на магистралу (види слику 4.3.), он је на располагању свим компонентама рачунара, тј. доступан је и меморији, и управљачкој јединици, и аритметичко–логичкој јединици, и било којој излазној јединици, али само једна од њих може стварно да га преузме. **Која?** То зависи од тога који управљачки сигнал ће генерисати управљачка јединица ($\mathbf{M_W}$, $\mathbf{A_W}$ или $\mathbf{R}$).

Ако податак са магистрале преузима аритметичко–логичка јединица, онда ће бити генерисан сигнал $\mathbf{A_W}$ (слика 4.4), док ће меморија и излазна јединица остати неактивне. Слово (и индекс) **W**, представља скраћеницу од речи **write**, а означава операцију уписивања податка. Слово **R** (и индекс)

представља скраћеницу од речи **read**, што означава читање, док индекс **A** означава да су у питању сигнали адресе.

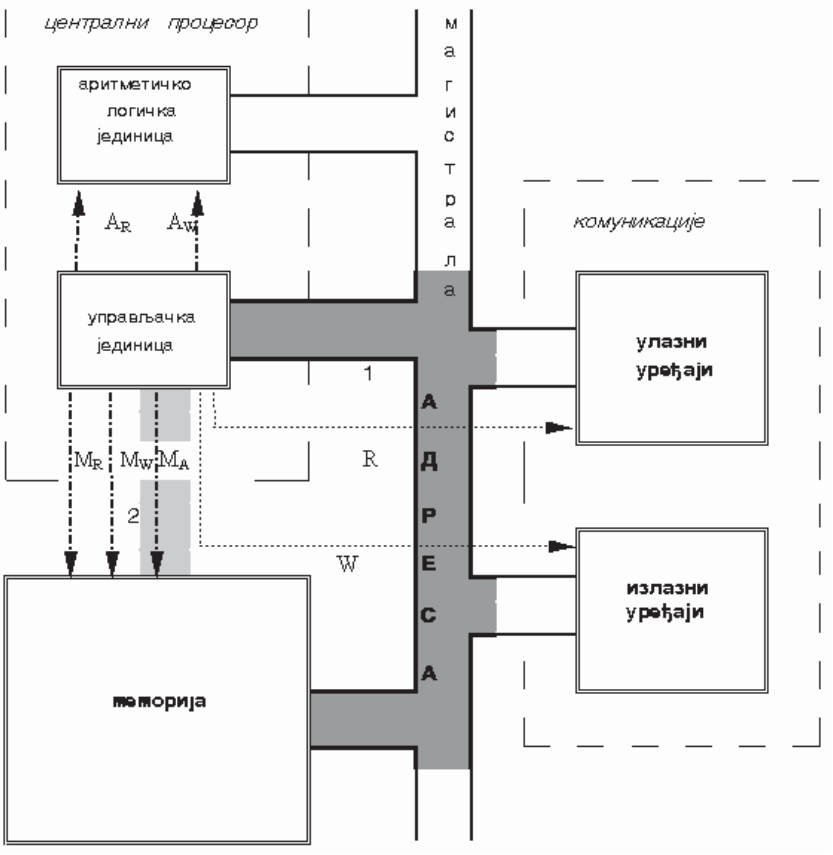


Слика 4.4. Управљачка јединица одређује одредиште податка, и шаље управљачки сигнал одговарајућој јединици, у овом случају је то аритметичко–логичка јединица

4.1.2. ПРИСТУП МЕМОРИЈИ

Смештање података у меморију је знатно сложеније од смештања података у аритметичко–логичку јединицу. Меморија садржи велики број меморијских локација (неколико милиона), због чега морамо тачно одредити коју ћемо локацију користити за смештање податка.

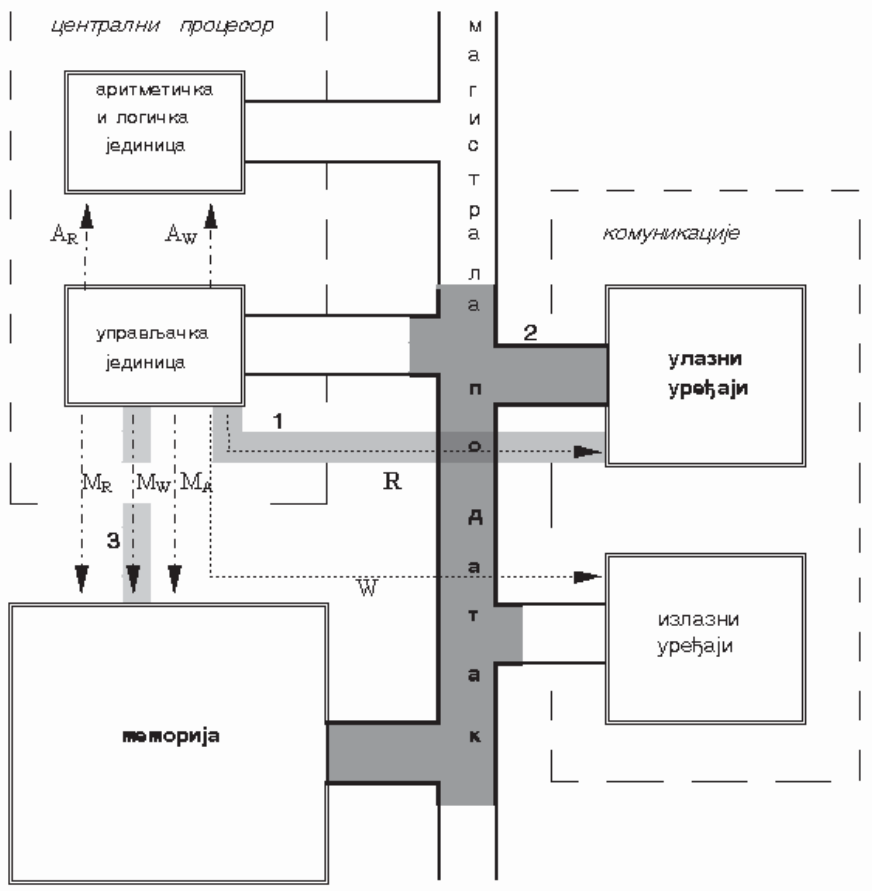
Овај проблем решава управљачка јединица тако што према меморији пошаље адресу меморијске локације (преко адресне магистрале), а посредством управљачког сигнала $\mathbf{M_A}$ обавести меморију да прихвати адресу (слика 4.5.). Процес читања података са улаза и смештања у меморију захтева неколико координираних активности.



Слика 4.5. Пошто главна меморија садржи много потенцијалних локација за смештање податка, управљачка јединица мора да пошаље потребну адресу на магистралу и саопшти меморији да је преузме ($\mathbf{M_A}$)

Управљачка јединица мора, најпре, да пошаље меморији адресу жељене локације (слика 4.5.), и да сачека потребно време да омогући меморији да прихвати адресу, а затим активира улазну јединицу помоћу линије $\mathbf{R}$.

Треба увек имати у виду чињеницу да се све операције одвијају у неком коначном интервалу времена, без обзира што су рачунари изузетно брзи, и да се све операције обављају сукцесивно, тј. једна за другом. Улазна јединица, под дејством управљачког сигнала на линији **R,** извршава постављени задатак тако што поставља податак на магистралу података.



Слика 4.6. Пошто меморија већ "зна" коју адресу (локацију) треба користити, управљачка јединица наређује улазној јединици да пошаље податак на магистралу (сигнал **R**), и наређује меморији да преузме податак са магистрале (сигнал $\mathbf{M}_W$)

Меморији се тада упућује захтев, преко линије $\mathbf{M}_W$, да прихвати податак који је улазна јединица поставила на магистралу и да га смести на изабрану локацију (слика 4.6). Све ове активности су координиране у времену да би

се обезбедило поуздано копирање података са неке од улазних јединица у одређену локацију у оперативној меморији.

Подаци се из меморије најчешће преносе (копирају) у аритметичко–логичку јединицу, односно у неки од регистара који су јој придружени (акумулатори итд.). Као и код улаза података, за пренос података у меморију потребно је извршити неколико координираних активности.

Прво, управљачка јединица поставља на адресну магистралу адресу меморијске локације у којој је записан податак, а затим шаље управљачки сигнал ка меморији да би она прихватила адресу.

Друго, управљачка јединица шаље меморији, преко контролне линије $\mathbf{M_R}$, управљачки сигнал којим се копира садржај изабране меморијске локације на магистралу, односно, чита се податак из селектоване локације, Када се подаци нађу на магистрали, управљачка јединица активира контролну линију $\mathbf{A_W}$, којом наређује аритметичко–логичкој јединици да преузме податак са магистрале.

Не смемо заборавити да **ALU** садржи у себи неколико различитих регистара, па су самим тим потребни још неки контролни сигнали да би се тачно одредио регистар у **ALU** који прихвата податак.

Пребацивање података из **ALU** у неку локацију меморије обавља се на сличан начин. Управљачка јединица прво поставља на адресну магистралу адресу меморијске локације, затим управљачки сигнал $\mathbf{A_W}$ да би **ALU** поставила податак на магистралу података. Након тога се шаље управљачки сигнал $\mathbf{M_W}$ који казује меморији да прихвати податке са магистрале.

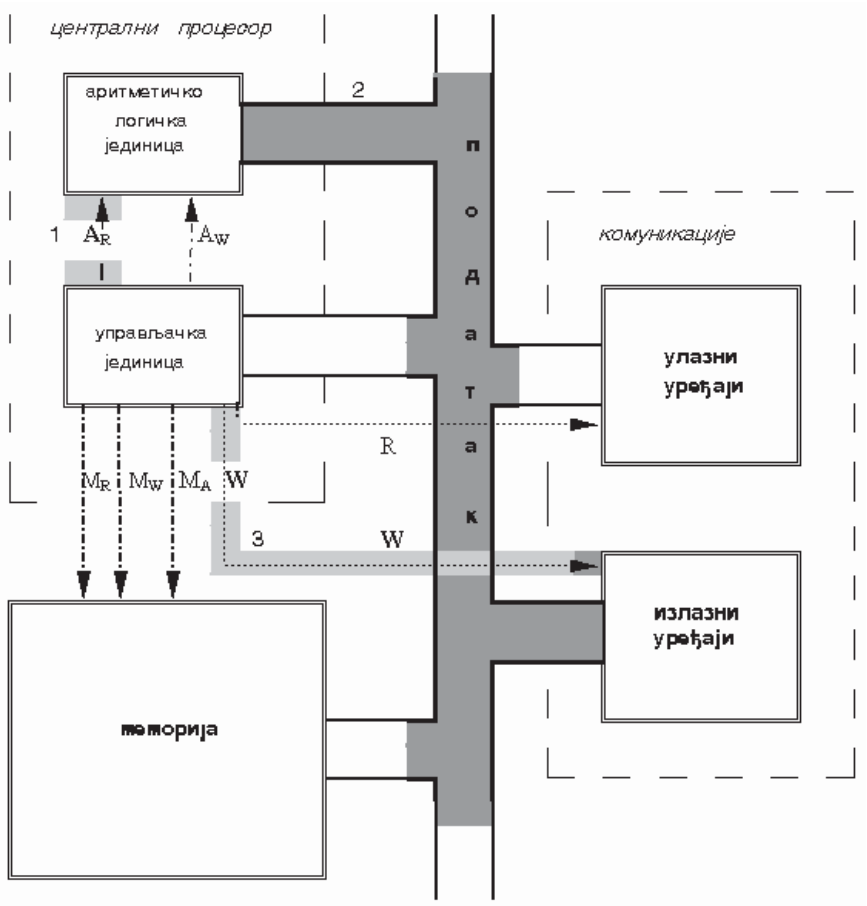
4.1.3. ИЗЛАЗ ПОДАТАКА

Податак који се шаље спољашњем свету мора бити копиран на магистралу било из аритметичко–логичке јединице (коришћењем управљачке линије $\mathbf{A_R}$ као на слици 4.7), било из меморије (коришћењем управљачке линије $\mathbf{M_R}$ као на слици 4.8).

Активирање управљачке линије **W** допушта излазној јединици да прихвати податак са магистрале.

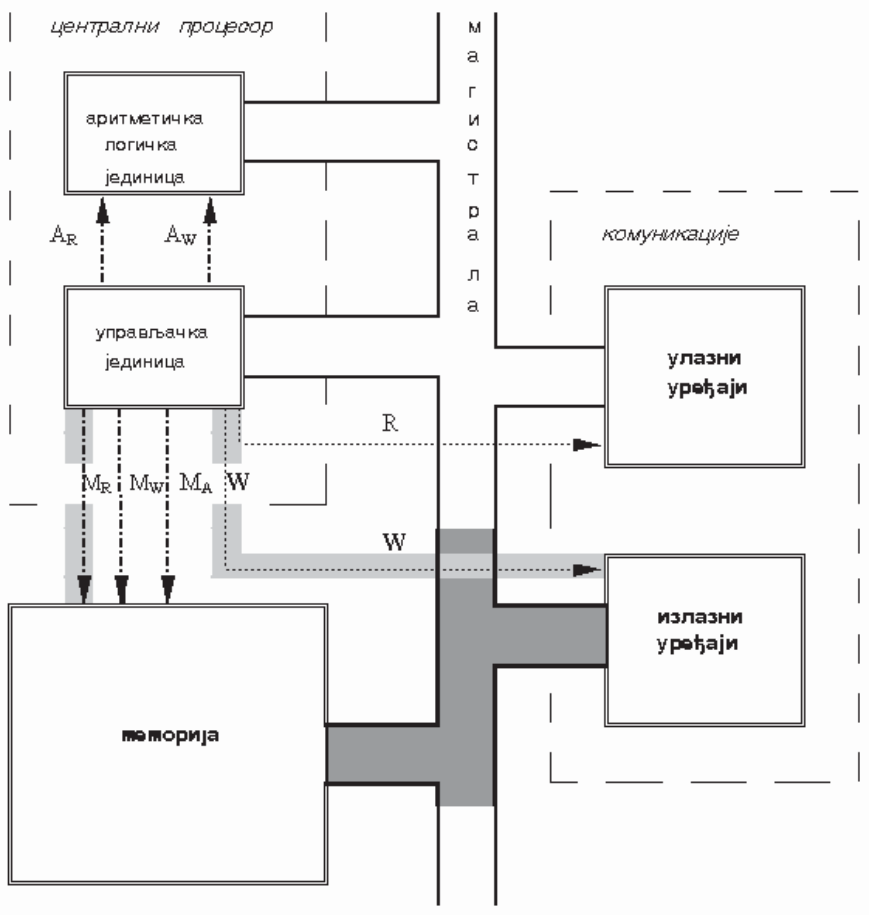
Ако се излазном уређају шаље податак из меморије, управљачка логика претходно мора да обезбеди адресу меморијске локације и да је активира помоћу сигнала $\mathbf{M_A}$. Када управљачка јединица пошаље меморији сигнал за читање (преко контролне линије $\mathbf{M_R}$), садржај изабране локације (тј. податак) преноси се на магистралу података.

На крају процеса преноса података из меморије у излазни уређај управљачка логика мора да генерише сигнал **W** након чега излазни уређај преузима податак са магистрале.



Слика 4.7. Податак може бити копиран из аритметичко–логичке јединице на излазну јединицу тако што се прво нареди **ALU** да пошаље податак на магистралу (сигнал $\mathbf{A}_R$), а потом се активира излазни уређај (сигнал **W**)

Број улазних и излазних уређаја такође може бити велики, па се и за избор једног од њих користе додатни контролни сигнали, тј. управљачка логика мора да генерише (израчуна) потребне адресе и за сваки периферијски уређај понаособ.

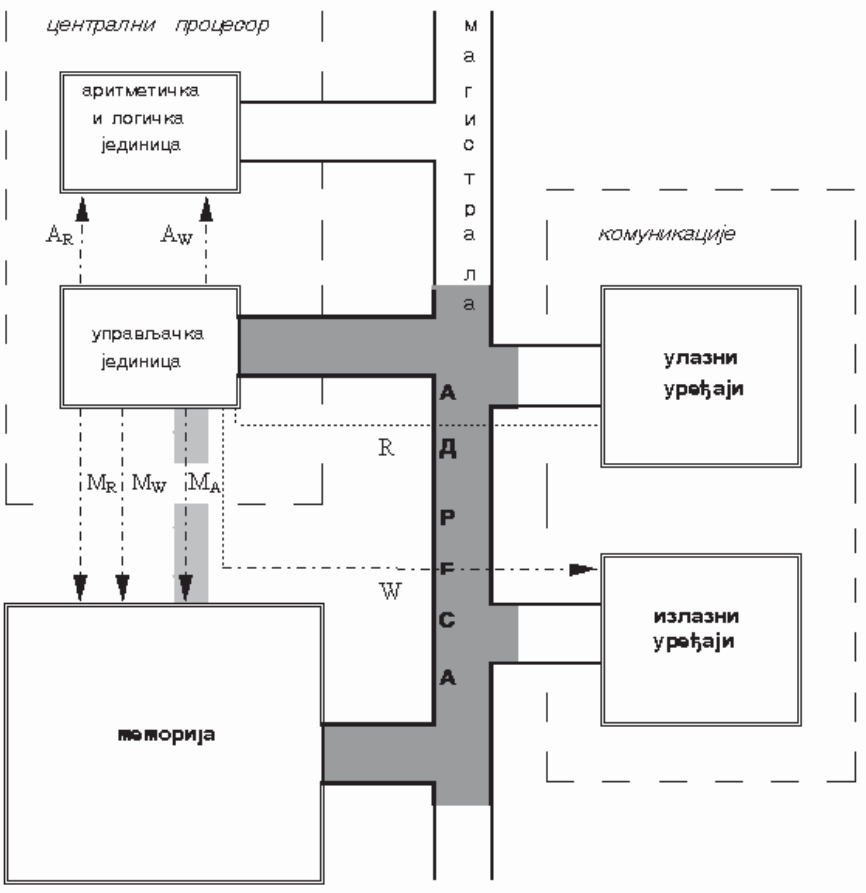


Слика 4.8. Податак може бити пренет из меморије на излазну јединицу активирањем сигнала (линије $\mathbf{M}_R$) за читање меморије и активирањем излазног кола (сигнал $\mathbf{W}$).
(Меморијска локација је претходно изабрана помоћу адресе и сигнала $\mathbf{M}_A$)

4.1.4. ПРИБАВЉАЊЕ ИНСТРУКЦИЈА

Главни проблем за један рачунар је следећи: како може управљачка јединица да зна које управљачке сигнале и када треба да генерише? Ова информација се добија из низа инструкција које су смештене у меморији. Управљачка јединица мора да прибави инструкцију из меморије пре него што под њеним дејством почне да генерише управљачке сигнале.

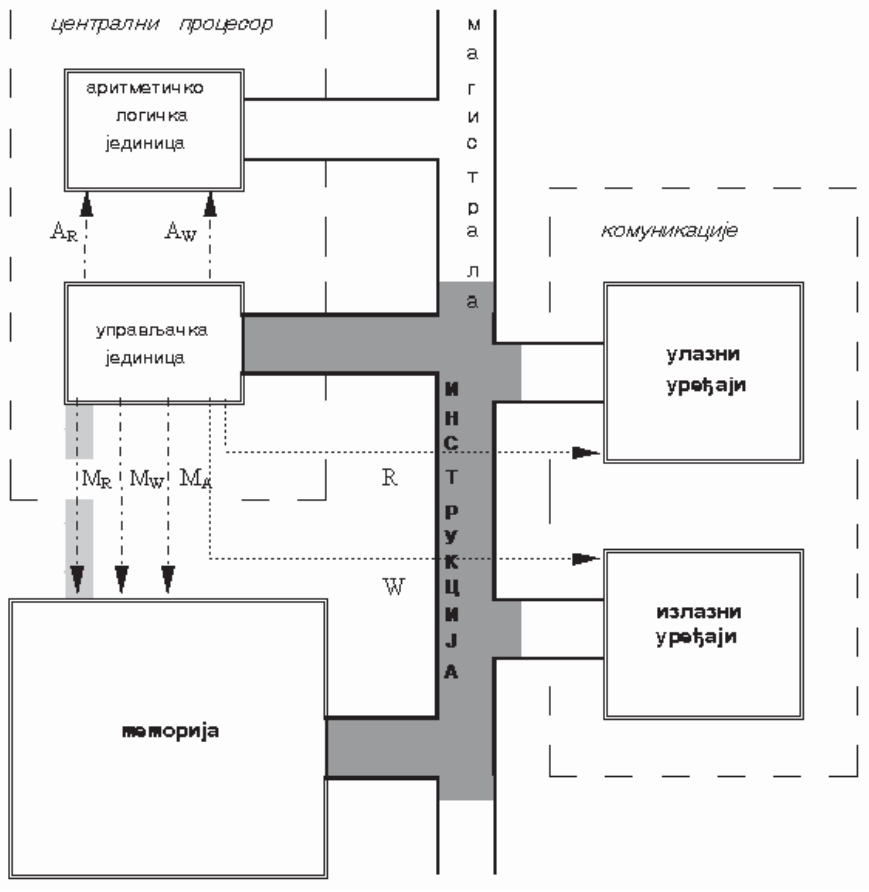
Током прибављања инструкције, управљачка јединица поставља адресу меморијске локације у којој је записана потребна инструкција. Када је адреса постављена на адресни део магистрале управљачка јединица помоћу сигнала $\mathbf{M_A}$ "казује" меморији да треба да је прихвати (слика 4.9).



Слика 4.9. Да би прибавила инструкцију из меморије, управљачка јединица мора меморији да пошаље адресу локације у којој се налази инструкција

Управљачка јединица затим шаље меморији управљачки сигнал $\mathbf{M_R}$, након којег се чита садржај селектоване (адресиране) локације и поставља се на магистралу. Инструкција која је постављена на магистралу (слика 4.10), доступна је свим деловима рачунара (али сви су неактивни), али аутоматски је преузима управљачка јединица.

За разлику од података чији токови кроз рачунар могу бити различити, инструкције се из меморије копирају искључиво у управљачку јединицу.



Слика 4.10. Након постављања адресе жељене локације, управљачка јединица захтева од меморије да копира садржај те локације (инструкцију) на магистралу

4.2. ИНСТРУКЦИЈЕ У МАШИНСКОМ ЈЕЗИКУ

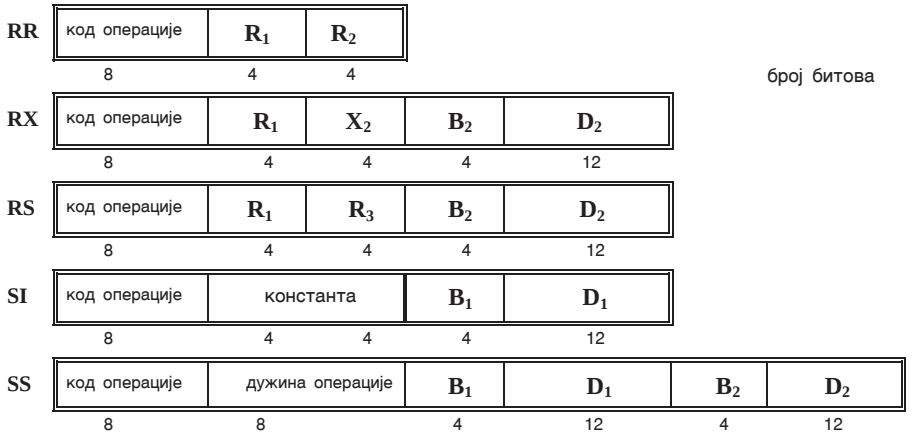
Скоро сви рачунари раде на принципу концепта меморисаног програма, који је развио **John von Neumann** са својим тимом средином 1940. године. Овај концепт допушта да једна меморијска локација може да садржи или податак или инструкцију. У оваквим рачунарима инструкције се представљају

као низови бита, који садрже бинарно кодирану информацију, која се односи на жељену операцију. На слици 4.11 приказани су неки од могућих облика инструкције, тј. формата инструкције.

Слика 4.11. Типични формати инструкција

Код операције тј. поље обраде, указује на жељену акцију која треба да се изврши под дејством те инструкције. Та акција може бити: копирање података из меморије у аритметичко–логичку јединицу, сабирање, нека логичка операција итд. После кода операције најчешће следи једно или више поља која се зову операнди. Ова поља могу да задају регистре (у CPU), или адресу локације у меморији која се користи током извршавања инструкције.

На слици 4.12 приказани су главни формати инструкција које користе рачунари IBM 370.



Слика 4.12. Главни формати инструкција за рачунаре **IBM** серија **370**

Поља са ознаком $\mathbf{R_X}$ **(**$\mathbf{R_1}$,$\mathbf{R_2}$,**...)** указују на регистре у CPU у којима се налазе подаци, или ће бити смештен резултат. Поља $\mathbf{B_X}$ указују на базне регистре, а $\mathbf{D_X}$ на помераје (**displacement**) који се користе у поступку израчунавања адресе локације. Поља $\mathbf{X_i}$ **(**$\mathbf{X_1}$ **,** $\mathbf{X_2}$ **,...)** указују на индекс–регистар који се

користи при адресирању. Уочимо да је поље кода операције величине 8 бита, што значи да може постојати 256 различитих кодова операција.

Поља регистара су четворобитна, што значи да се може задати један од 16 регистара. Између поља код операције и адресног дела инструкције понекад се умеђу додатни бити који указују на специјалне методе адресирања, (као на слици 4.13), где је приказан формат инструкције рачунара **Unisys 1100**.

opcode	**mod**	**reg.**	**reg.**	**h**	**i**	**operand**
6	4	4	4	1	1	16

Слика 4.13. Формат инструкција рачунара **Unisys 1100**

Рачунари VAX имају у свом списку и такве инструкције, које имају велики број операнада као што је то приказано на слици 4.14.

код операције (1 или 2 бајта)
спецификатор операнда 1
спецификатор операнда 2
спецификатор операнда 3
. .
спецификатор операнда N

Слика. 4.14. Општи формат инструкције **VAX** рачунара

Када говоримо о формату инструкције очигледно је да сваки рачунар има свој формат, или чак више различитих формата, па се не може извршити никаква генерализација, но, уз одређена апстраховања, могуће је закључити следеће: све инструкције садрже поље кода операције – **opcode**. Што се тиче поља адресе операнда, постоји више врста инструкција:

– *нулаадресне, без поља операнда,*

– *једноадресне, са једним пољем операнда,*

– *двоадресне, са два поља операнада,*

– *троадресне, са три поља операнада,*

– *четвороадресне, са четири поља операнада.*

Од броја операнада, односно адресности инструкција, у многоме зависи ефикасност обраде података помоћу рачунара. Дугачке наредбе (вишеадресне инструкције), захтевају употребу дугачких меморијских речи, или се прибављају у више приступа меморији. Програми су прегледнији, читљивији и краћи, тј. заузимају мањи број меморијских локација. Наизглед, употреба вишеадресних инструкција повећава ефикасност обраде.

Но, у стварности то није случај, па су се четвороадресни рачунари користили само у почетној, експерименталној фази развоја рачунара. Већ први рачунар прве генерације **EDVAC,** који је почео са радом 1951. године, био је једноадресни, а имао је следећи формат инструкције:

opcode A .

Када се каже једноадресна, онда се мисли искључиво на адресирање меморијских локација. Адресе регистара у централном процесору, као и имплицитне адресе садржане у коду операције не одређују адресност инструкције.

4.2.1. ФАЗА ПРИБАВЉАЊА МАШИНСКИХ ИНСТРУКЦИЈА

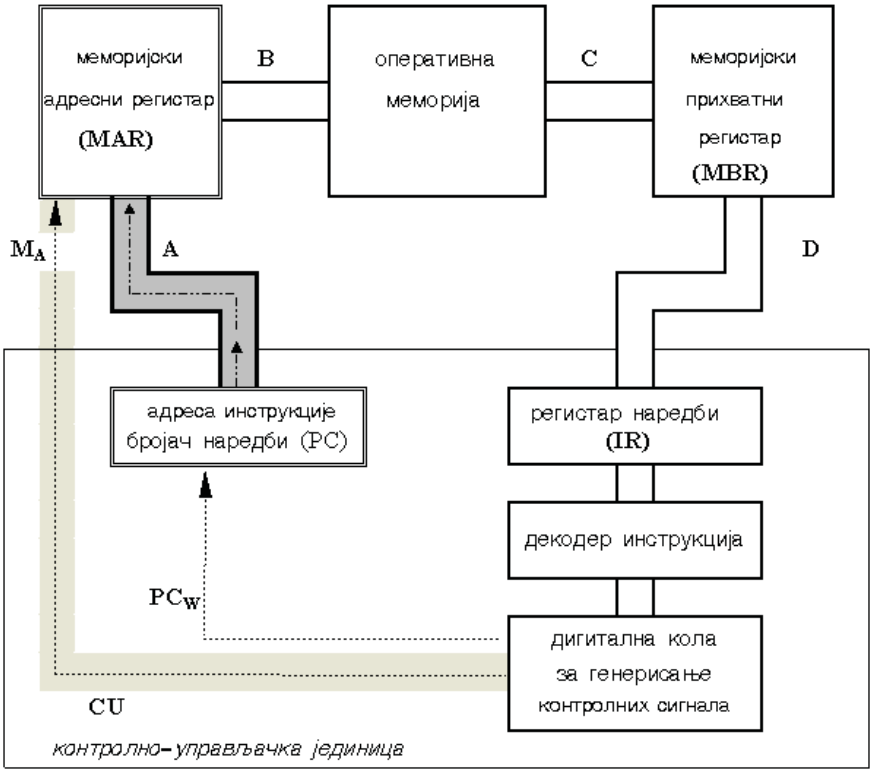
Независно од величине и формата инструкције, свака инструкција мора најпре бити копирана (прибављена) из меморије у управљачку јединицу, а тек после тога отпочиње њено извршавање и она преузима контролу над процесором. Посебне логичке мреже, у управљачкој јединици, користе се да декодирају и изврше инструкцију. Низ активности које су потребне да се инструкција пренесе из меморије у централни процесор, називамо **фаза прибављања** инструкције. За ова разматрања користићемо нешто сложенији модел централног процесора који је приказан на слици 4.15.

Адреса следеће наредбе, која ће бити извршена, налази се записана у регистру специјалне намене који се зове бројач наредби или програмски бројач (**program counter, PC**). Процес прибављања инструкције започиње копирањем садржаја програмског бројача у меморијски адресни регистар (**MAR**) преко линија означених са **A** (као на слици 4.15).

Да би се то остварило најпре **CU** генерише сигнал $\mathbf{PC_W}$, да се пренесе садржај програмског бројача на магистралу **A**, а затим **CU** генерише већ познати сигнал $\mathbf{M_A}$, који наређује меморији, тј., меморијском адресном регистру да прихвати адресу. Ове активности се стално понављају при прибављању, и зато их више нећемо помињати..

Адреса која се налази у **MAR**–у, пролази кроз логичку мрежу за декодирање типа 1 од 2^n, и генерише се сигнал који активира само једну од могућих 2^n локација у меморији. Стварни приступ меморији догађа се под

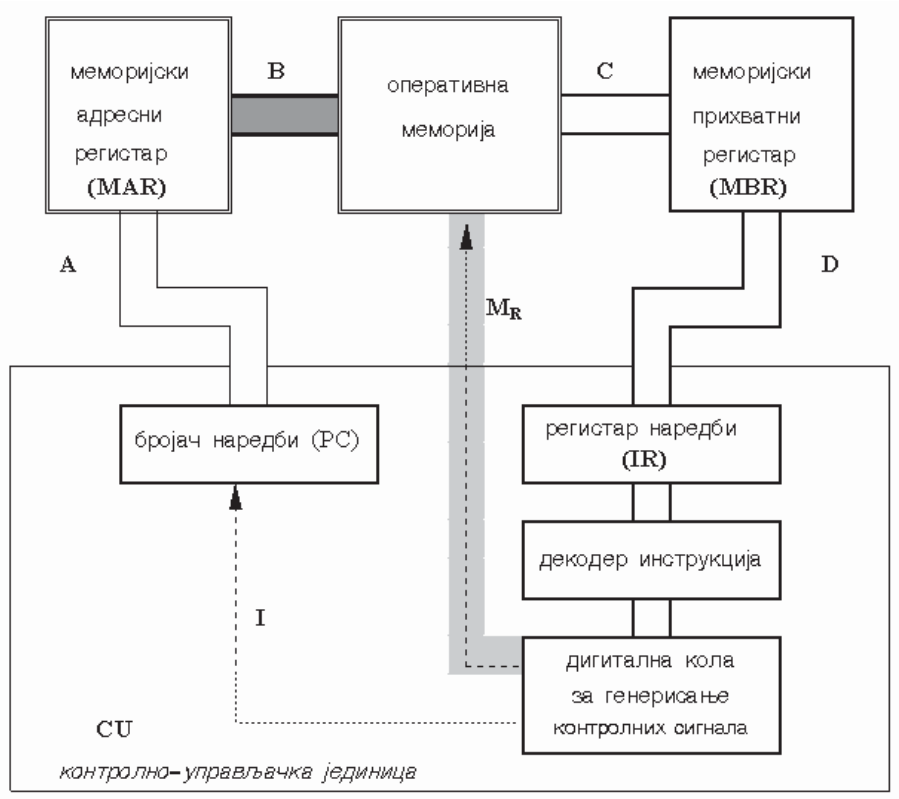
контролом сигнала за читање ($\mathbf{M_R}$) који поставља управљачка јединица (види слику 4.16).



Слика 4.15. Адреса локације у којој се налази следећа инструкција записана је у бројачу наредби. Адреса мора бити пренета у меморијски адресни регистар пре него што се приступи локацији и из ње се прочита инструкција.

Након пријема сигнала за читање, податак (који овог пута представља инструкцију) се из изабране меморијске локације копира, преко магистрале **C,** у други специјални регистар, меморијски међурегистар или прихватни регистар меморије (**memory buffer register, MBR**), као што се види на слици 4.17. Подаци који се налазе у **MBR**–у пребацују се преко магистрале **D** (слика 4.18) у специјални регистар такозвани **регистар инструкција** или регистар наредби (**instruction register, IR**). Из регистра наредби, део инструкције који чини код операције, пребацује се у логичку мрежу која се зове декодер инструкција, где се најпре проверава исправност кода, а за–

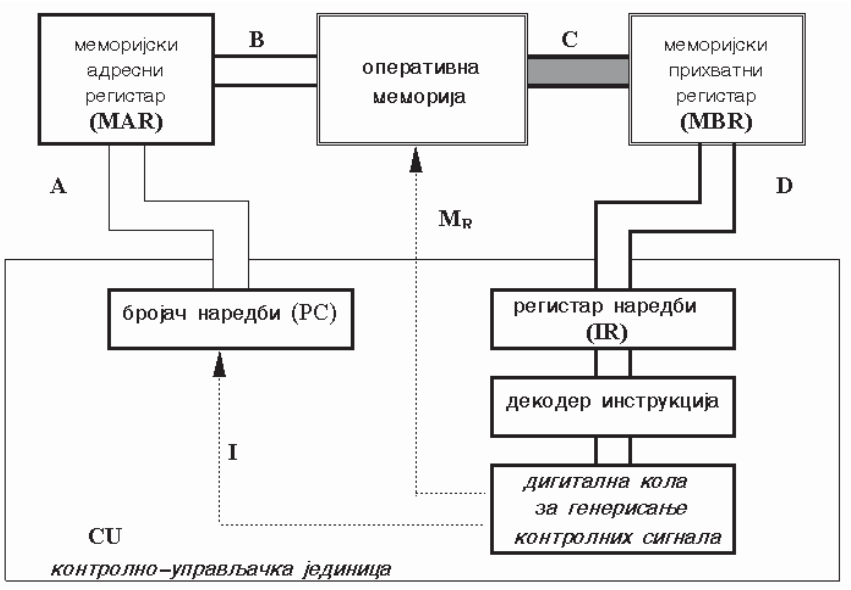
тим се код декодира, а касније генерише низ управљачких сигнала, који обезбеђују извођење оне операције која је задата у коду.



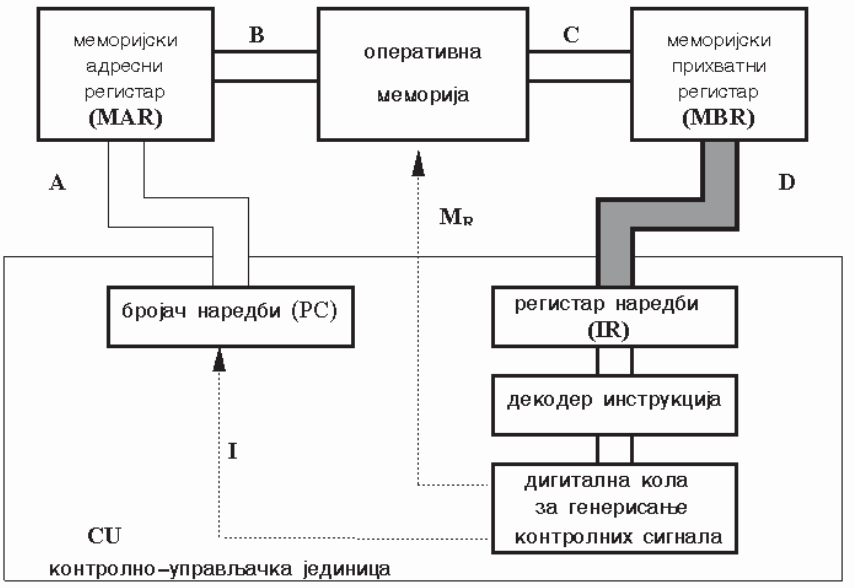
Слика 4.16. Постављање сигнала $\mathbf{M_R}$ " казује " меморији да копира на магистралу **C** податак запамћен на адреси на коју показује **MAR**

Последњи корак у току фазе прибављања инструкције представља инкрементирање програмског бројача (слика 4.19) помоћу управљачког сигнала **I**. Нови садржај програмског бројача је једнак старом садржају увећаном за један (**PC = [PC] + 1**).

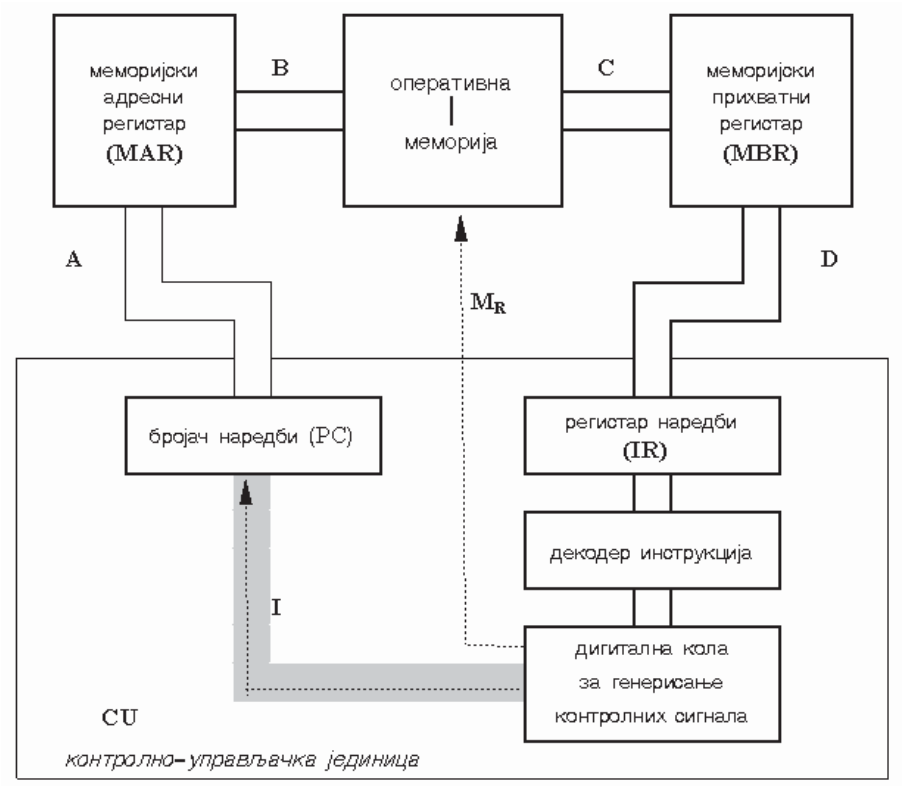
Овај корак поставља нову вредност у програмски бројач, тако да он сада садржи адресу нове локације у меморији, која непосредно следи иза локације која садржи инструкцију (код операције инструкције) која је управо прибављена. На тој локацији се најчешће налази следећа инструкција. Али, код неких рачунара који имају кратке меморијске речи (нарочито код осмобитних и шеснаестобитних микрорачунара) на следећој меморијској локацији се често налази операнд инструкције која је управо прибављена.



Слика 4.17. Податак који је прочитан из меморије копира се у **MBR**



Слика 4.18. Током фазе прибављања садржај **MBR** се копира у регистар наредби (**IR**)



Слика 4.19. Бројач наредби (**Program Counter, PC**) се инкрементира на крају циклуса прибављања инструкције

Из овога непосредно проистиче претпоставка да су операнди (подаци, њихове адресе или инструкције), које ће бити коришћене непосредно по прибављању инструкције, смештени у меморији на локацији одмах иза текуће инструкције.

Ова претпоставка се заснива на чињеницама да операнди непосредно следе иза кода операције, и да се већина инструкција извршава секвенцијално, односно једна за другом, оним редоследом којим су смештене у меморију (односно оним редоследом којим су написане у програму).

Сва ова разматрања су апроксимативна, јер инструкције се међу собом пуно разликују од рачунара до рачунара. У једном рачунару постоје различите инструкције по дужини, по месту где су операнди (меморија или регистри), по броју операнада, по начинима адресирања итд. Зато ћемо се ограничити на разматрање једноадресних инструкција (и рачунара), јер су оне најрас–прострањеније.

Адресност наредби битно утиче на ефикасност обраде података у рачунару, а мери се следећим параметрима:

– *временом потребним за извођење програма (мери се укупним бројем меморијских циклуса потребних за извршавање свих инструкција програма),*

– *меморијским простором потребним за записивање програма.*

Четвороадресне наредбе, којима се описују бинарне аритметичке и логичке операције, садрже: код операције, адресе првог и другог операнда, адресу резултата и адресу следеће наредбе.

Али, ако се наредбе смештају у меморију оним редоследом којим се извршавају, адреса следеће наредбе није неопходна, већ се уводи специјални хардвер, односно бројач наредби (**program counter, PC**). Бројач наредби је регистар специјалне намене који у себи увек садржи адресу следеће инструкције, која тек треба да се прибави и изврши. На овај начин се потребни број адреса смањио, тј. довољне су троадресне инструкције. Пошто се при извођењу програма понекад мора одступити од секвенцијалног редоследа извођења инструкција (тј. следећа инструкција није на следећој меморијској локацији), морају се увести специјалне инструкције за контролу тока програма, тј. инструкције скока (**jump**) или гранања (**branch**).

Даље поједностављење се постиже увођењем ограничења тако да се резултат бинарних операција увек привремено смешта у неки регистар **CPU**, најчешће у акумулатор. Но, да би се добијени резултат сачувао за даљу обраду, неопходно га је пренети у неку локацију у оперативној меморији, односно, неопходно је имати и наредбу за пренос садржаја акумулатора у меморијску локацију (**store, move**).

Да би била могућа употреба само једноадресних наредби уводе се додатна ограничења у погледу смештања операнада и резултата обраде. Наиме, код бинарних операција један од операнада је увек у једном од акумулатора, па је неопходно проширити списак наредби увођењем наредби за пуњење акумулатора, тј. за пренос садржаја из неке меморијске локације у акумулатор (**load, move**).

Да би користили наредбе без адреса, сви операнди (тј. подаци) морају бити у тачно одређеним регистрима централног процесора, а њихове адресе су садржане у самом коду операције. Да би се операнди сместили у одређене регистре, сет инструкција рачунарског система мора се проширити инструкцијама за пренос података између два регистра у централном процесору.

На крају, можемо закључити следеће: смањење адресности инструкција постиже се увођењем нових хардверских компоненти у централни процесор, нових инструкција и одређених ограничења у погледу смештања инструкција, операнада и разултата операција.

4.2.2. ФАЗА ИЗВРШАВАЊА МАШИНСКИХ ИНСТРУКЦИЈА

Након што је инструкција смештена у управљачку јединицу и у програмски бројач постављена нова адреса меморијске локације, фаза прибављања је завршена. Контролу над управљачком јединицом преузима прибављена инструкција, и отпочиње фаза извршења. Да би лакше схватили ову фазу, посматрајмо је на примеру извођења једне елементарне операције сабирања два броја:

$$\mathbf{sum = num1 + num2}\ .$$

Преведено на говорни језик ова наредба казује следеће: бројну вредност (на пример: 05_{10}) запамћену у меморијској локацији са симболичком адресом **num1,** сабрати са бројем (на пример: 07_{10}) запамћеним у локацији са симболичком адресом **num2** и резултат сместити у локацију са симболичком адресом **sum**.

Већина рачунара смешта податке у меморију, али логичке, аритметичке и друге операције изводи, по правилу, над садржајима регистра у централном процесору, тачније у аритметичко–логичкој јединици. То одмах има за последицу да напред наведена наредба не би могла да се изведе непосредно, већ у неколико корака (тј. као низ неколико машинских инструкција).

Најпре, треба податке пребацити из меморије у регистре централног процесора, затим извршити сабирање, а потом резултат вратити натраг у меморију. Стварни низ машинских инструкција које решавају наведени проблем варира од рачунара до рачунара. Једна од могућности је решење помоћу три машинске инструкције.

Прва инструкција има код операције који указује на копирање податка из меморије у регистар (нпр. акумулатор у ALU или слично). Ово подсећа на поступак пуњења акумулатора, па се симболички ова операција може назвати пуњење (**load**). Ова инструкција мора указивати на меморијску локацију где је смештен податак, а коју зовемо **num1.**

Друга инструкција има код операције који казује да садржај друге меморијске локације (коју зовемо **num2**) треба додати (сабрати–**add**) броју у регистру (који је коришћен у претходној инструкцији), а резултат поново вратити у тај регистар. И ова инструкција садржи указатељ на податак смештен у меморији.

Трећа инструкција казује да резултат треба пренети из регистра у централном процесору, у неку локацију у меморији и запамтити га (**store**). Као и у претходним, и у овој инструкцији морамо на неки начин указати на адресу жељене меморијске локације, а у овом случају то је локација симболичким именом **sum**.

Нека је променљива **num1** смештена у меморији на локацији $\mathbf{26}_{10}$. Онда је њена бинарна адреса $\mathbf{00011010}_2$, односно $\mathbf{1A}_{(16)}$ у хексадецималном бројном систему. По аналогији, **num2** је смештен на следећој меморијској локацији чија је адреса $\mathbf{1B}_{(16)}$, а **sum** је на локацији са адресом $\mathbf{1C}_{(16)}$.

Претпоставимо, такође, да су све инструкције исте дужине, тј. 2 бајта. У првом бајту је код операције, а у другом адреса операнда. Претпоставимо и то да су ове инструкције смештене у меморију једна за другом почев од локације са адресом $\mathbf{100}_{(16)}$, као што је то приказано на слици 4.20 б). Изглед програма на асемблеру дат је на слици 4.20 а):

адреса	**садржај**
100:	**load data**
101:	**1A**
102:	**add data**
103:	**1B**
104:	**store data**
105:	**1C**

а) машински програм

податак	симболичка адреса	физичка адреса	меморија
05	num1	1A	00000101
07	num2	1B	00000111
?	sum	1C	-
		...	...
		100	load data
		101	00011010
		102	add data
		103	00011011
		104	store data
		105	00011100
		106	...

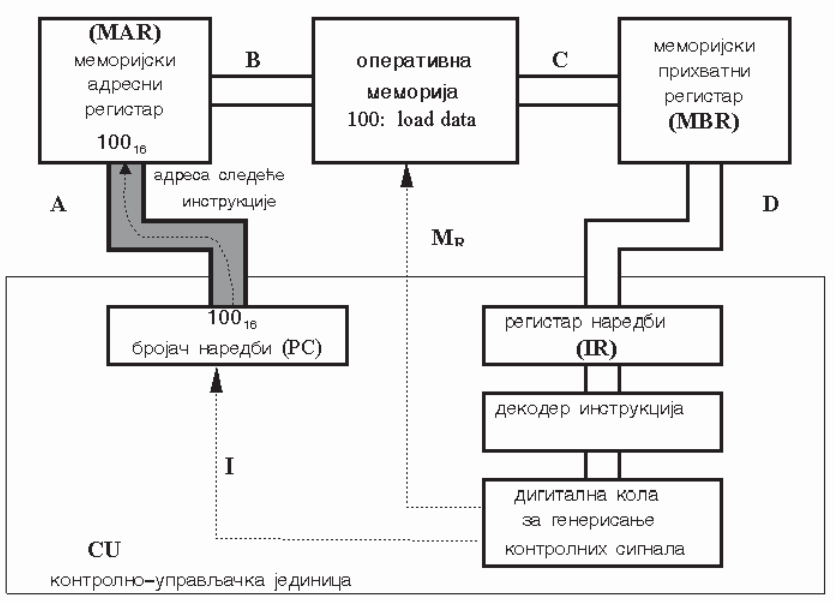
б) стање меморије

Слика 4.20. Сабирање два броја

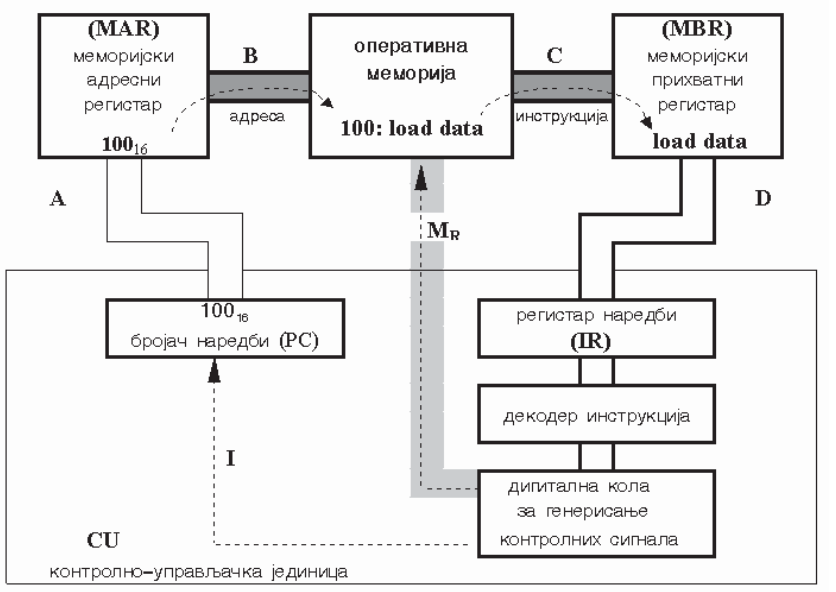
Уочимо да смо због прегледности и разумљивости користили симболичка имена операција (иначе, у меморији, оне су кодиране као осмобитни низови **0** и **1,** тј. приказане су као бинарни бројеви).

Први корак у извођењу овог низа инструкција је отпочињање фазе прибављања прве инструкције. У бројач наредби се упише број $\mathbf{100}_{(16)}$, и отпочиње фаза прибављања. По завршетку фазе прибављања, у регистру наредби (**IR**) налази се код операције (**load data**), а програмски бројач се инкрементира на $\mathbf{101}_{(16)}$. Шта се током прибављања инструкције збивало у рачунару приказано је на сликама 4.21, 4.22 и 4.23.

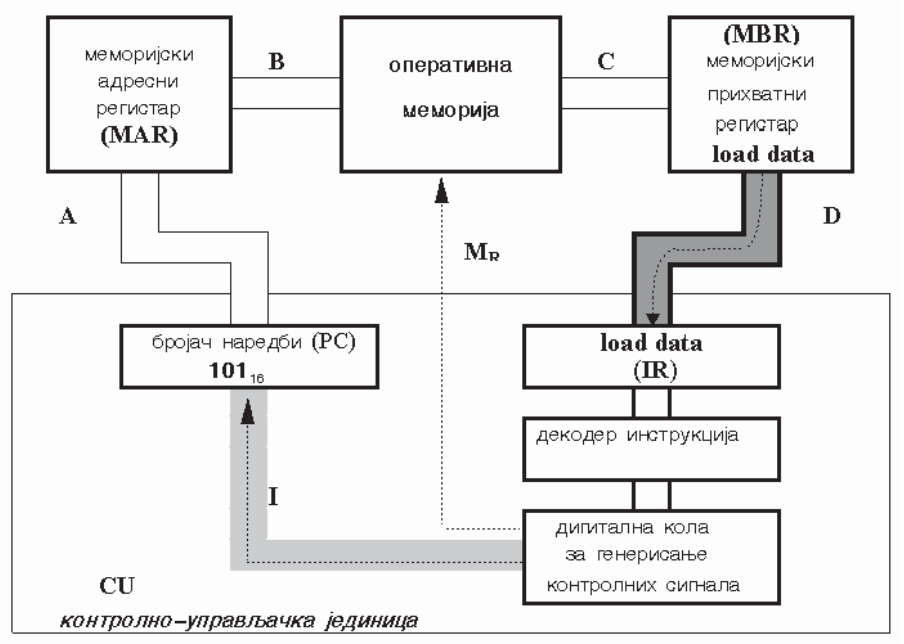
Током декодирања ове наредбе, управљачка логичка кола утврдила су да треба садржај једне меморијске локације прекопирати у један од регистара CPU. У коду операције (**load**) такође је специфицирано да поље операнда садржи адресу податка. Да би податак могао бити копиран у неки регистар централног процесора, његова адреса мора бити претходно одређена. Бројач наредби, чији је садржај увећан за **1** на крају циклуса прибављања, садржи адресу меморијске локације у којој је смештен операнд.



Слика 4.21. Прибављање инструкције са локације 100_{16}, најпре се садржај програмског бројача преноси у меморијски адресни регистар (**MAR**)



Слика 4.22. Када је управљачки сигнал за читање постављен (**R**), у меморији се проналази инструкција на адреси на коју указује **MAR**, и преноси је у **MBR**



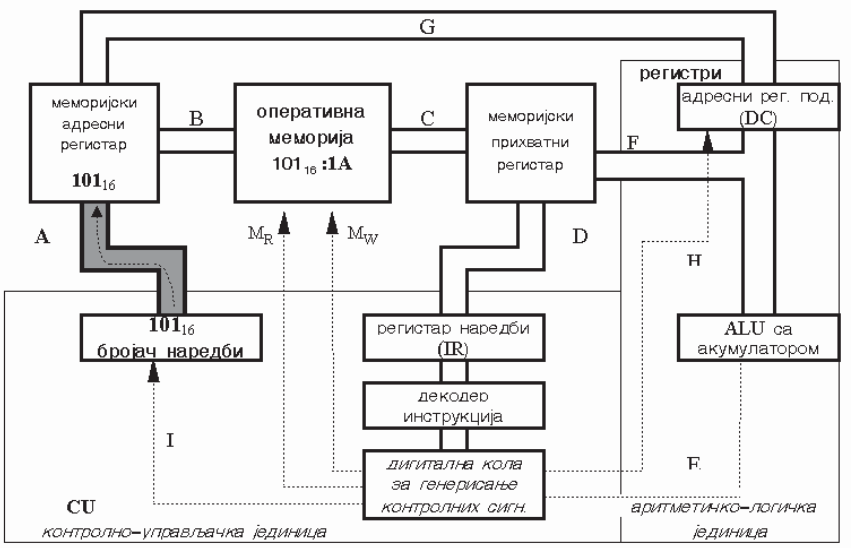
Слика 4.23. Током последње фазе прибављања инструкције, код операције се преноси из **MBR** у регистар инструкција (**IR**), а програмски бројач се инкрементира.

Процес прибављања операнда захтева од управљачке јединице следеће активности:

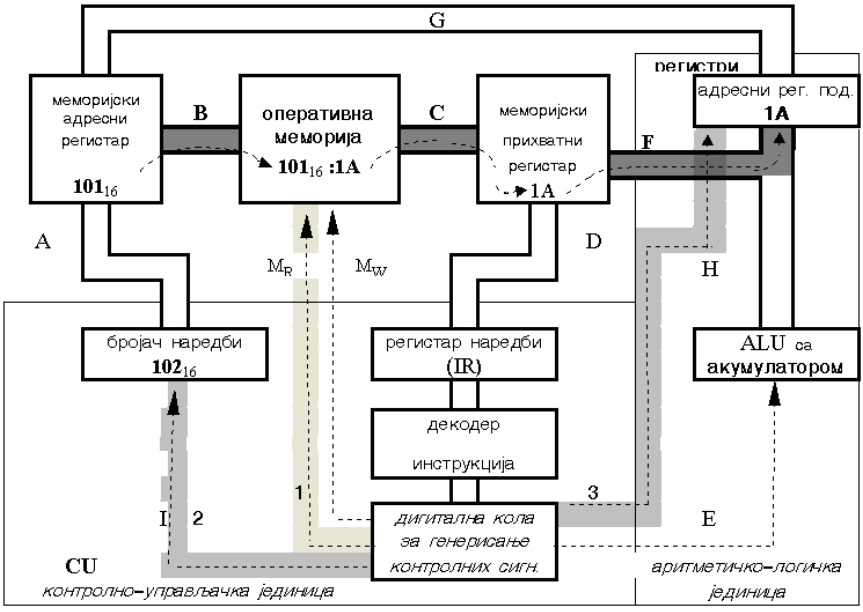
1. *Копирање садржаја програмског бројача (**PC**) преко адресне магистрале (**A**), у меморијски адресни регистар (**MAR**) – слика 4.24.*
2. *Постављање сигнала за читање из меморије (M_R на слици 4.25).*
3. *Пребацивање операнда из меморије помоћу бафер регистра (**MBR**) у регистар адресе података (**data counter, DC**), преко линија **C** и **F** на слици 4.25.*
4. *Инкрементирање садржаја бројача наредби, сигнал **I** на слици 4.25.*

Поновимо још једном: податак који се налази у **MBR** није број који треба пренети у **ALU** или регистре, већ адреса тог броја.

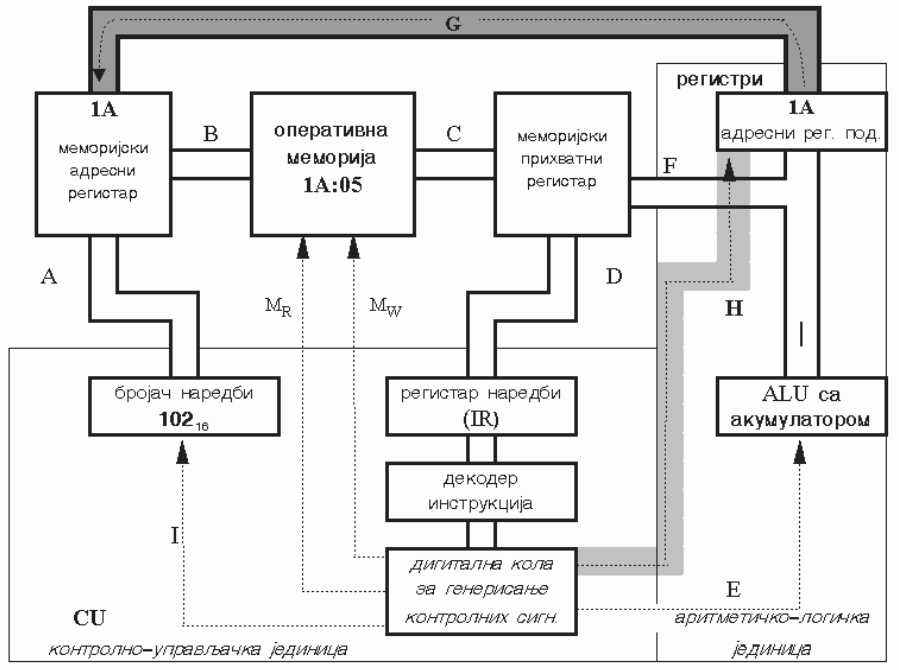
Користећи ту адресу, управљачка јединица ће коначно пренети податак, но за комплетирање те активности потребно је извршити још један низ елементарних операција.



Слика 4.24. Први корак у фази извођења инструкције преноса је пребацивање садржаја **PC**, тј. адресе операнда у **MBR**



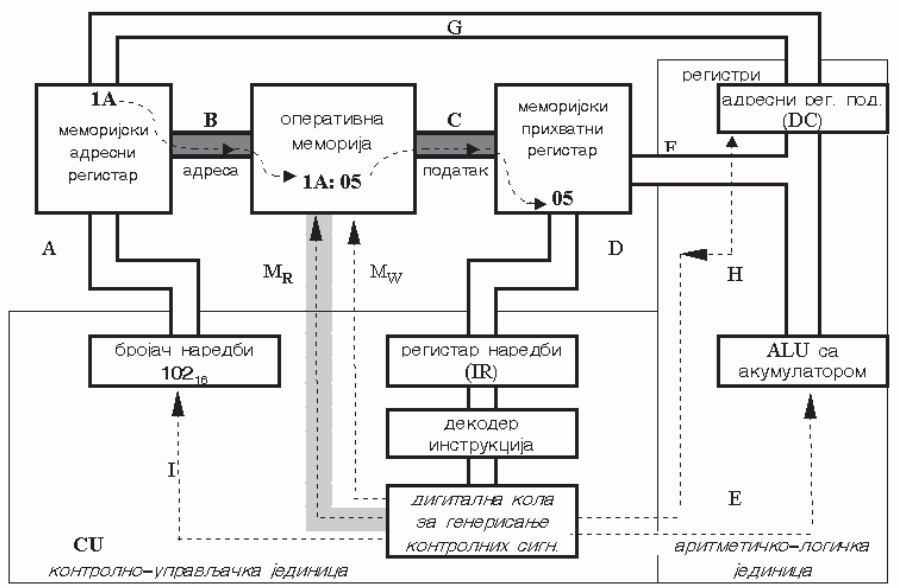
Слика 4.25. Сигналом $\mathbf{M_R}$ се чита операнд и пребацује се преко магистрале **C, MBR** и магистрале **F** у бројач података. Последњи корак је инкрементирање **PC**.



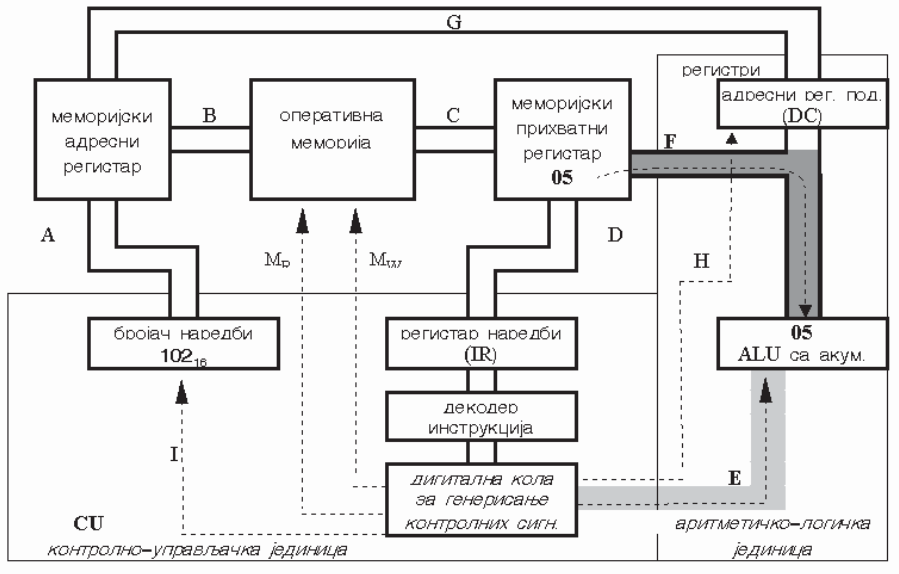
Слика 4.26. Операнд је уствари адреса податка, па се из адресног регистра података, преко магистрале **G,** шаље у меморијски адресни регистар **MAR**

5. *Пренети садржај бројача података (**DC**) у меморијски адресни регистар (**MAR**) преко адресне магистрале (**G** на слици 4.26).*
6. *Послати нови захтев за читање из меморије (**M_R** на слици 4.27).*
7. *Прихватити податак у **MBR** (преко линије **C**, слика 4.27).*
8. *Рећи једном од регистара у процесору да преузме податак (контролни сигнал **H** на слици 4.28).*
9. *Копирати податак преко магистрале **F** у један од регистара (то је најчешће акумулатор, слика 4.28).*

Након што су све ове операције обављене, фаза извршења прве инструкције је завршена, и управљачка јединица се враћа у режим прибављања инструкције. Програмски бројач указује на следећу инструкцију, тј. на њен први део. (код операције) који је смештен на локацији са адресом $\mathbf{102}_{(16)}$. Адреса из **PC** се копира у **MAR** и активира се захтев за читање из меморије. Инструкција **add data** се (из локације $\mathbf{102}_{(16)}$ у меморији), копира у **MBR** и аутоматски се, преко магистрале **D**, преноси у регистар **IR**. Фаза прибављања се завршава инкрементирањем бројача података на $\mathbf{103}_{(16)}$.

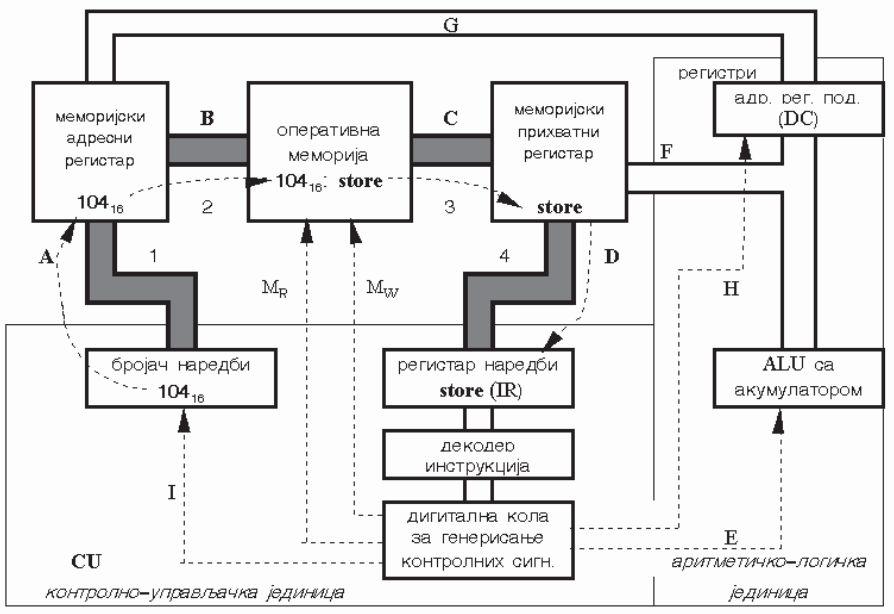


Слика 4.27. Копира се податак 05 из меморије (са адресе на коју указује операнд) у меморијски регистар података **MBR**



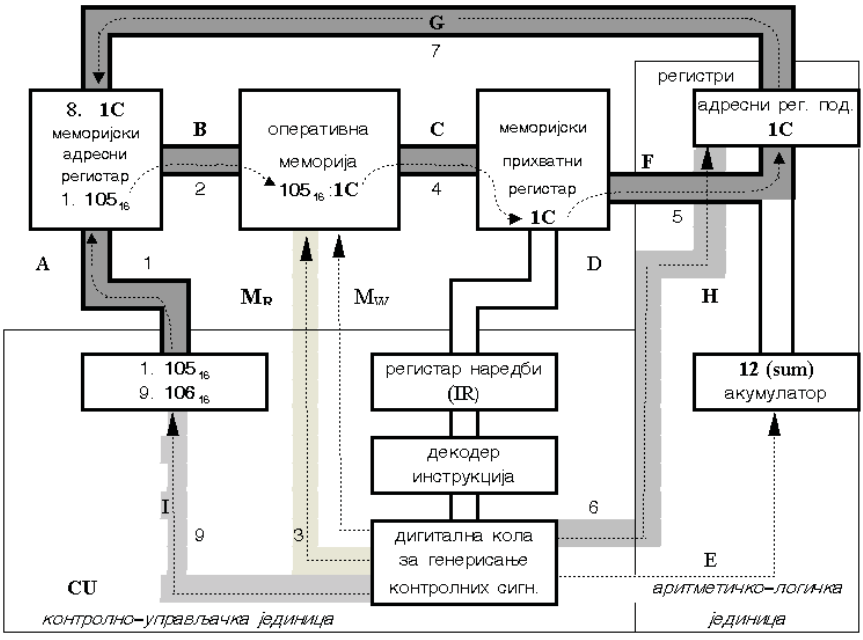
Слика 4.28. Податак се преноси из **MBR** у неки регистар у **CPU,** односно у акумулатор

Уочимо да је фаза прибављања инструкције **add** идентична фази прибављања инструкције **load,** приказаној на слици 4.21, 4.22 и 4.23. Током фазе извршења инструкције **add**, управљачка јединица мора да прибави операнд и употреби га као адресу; затим мора да приступи меморијској локацији са том адресом и њен садржај пребаци у **MBR** и инкрементира **PC** на $\mathbf{104}_{(16)}$, (слично као у инструкцији **load**). Но након тога, преко управљачких сигнала **E**, казује ALU да прихвати податак са магистрале **F** и дода га садржају изабраног регистра, тј. изврши сабирање, и резултат сачува у том истом регистру. Након прибављања кода операције програмски бројач се инкрементира на $\mathbf{103}_{(16)}$, а након прибављања операнда инкрементира се на $\mathbf{104}_{(16)}$. Након извршења инструкције сабирања отпочиње фаза прибављања следеће инструкције: пребацује се садржај из локације $\mathbf{104}_{(16)}$ (а то је инструкција **store**) у регистар наредби (**IR**), и инкрементира бројач наредби на $\mathbf{105}_{(16)}$, слика 4.29.

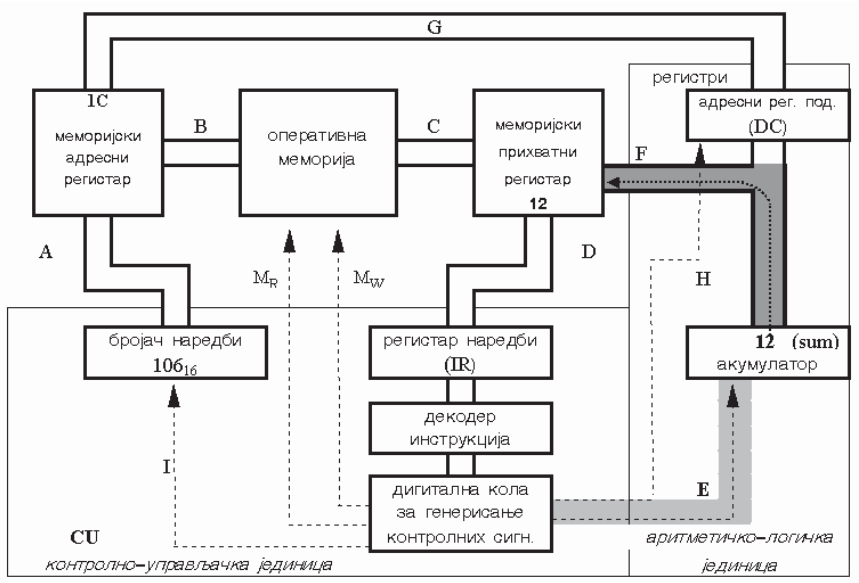


Слика 4.29. Приказ фазе прибављања инструкције **store** са локације $\mathbf{104}_{16}$

Први кораци извршења инструкције **store**, су исти као код претходне две инструкције. Након прибављања инструкције, приступа се операнду и **PC** се инкрементира на $\mathbf{106}_{(16)}$. Операнд се поставља у **MAR** (преко магистрале **G** слика 4.30), као адреса локације у којој ће бити смештен садржај акумулатора. Затим се шаље управљачки сигнал **E** ка акумулатору, и садржај акумулатора се преноси у **MBR** преко магистрале **F** (слика 4.31).

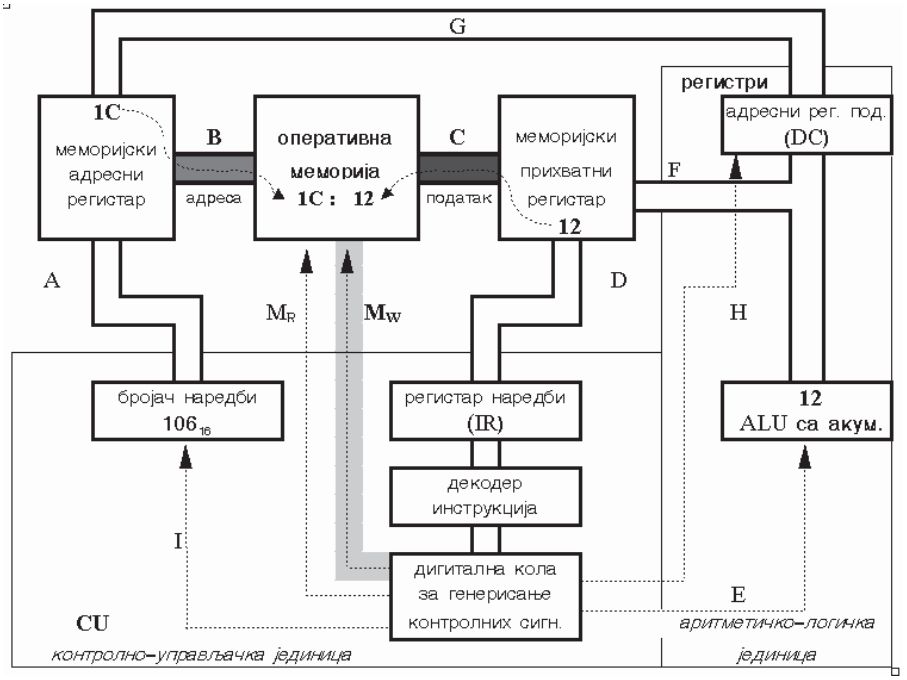


Слика 4.30. Обједињени приказ преноса операнда преко бројача података у меморијски адресни регистар (**MAR**)



Слика 4.31. Пренос податка из регистра у CPU (из акумулатора) у **MBR**

Коначно–управљачка јединица шаље команду у меморију ($\mathbf{M_W}$, контролни сигнал за уписивање садржаја у меморијску локацију), а садржај меморијског прихватног регистра (**MBR**) се смешта на локацију чија се адреса налази у меморијском адресном регистру (**MAR**) (види слику 4.32).



Слика 4.32. Слањем управљачког сигнала **W** (write), врши се уписивање податка из **MBR** на селектовану локацију у меморији (на коју указује адреса у **MAR**)

Да још једном истакнемо: управљачка јединица има неколико режима рада, али два су базична:

а) стање прибављања инструкције (**fetch**), када се управљачки сигнали генеришу аутоматски. Као што се види из описа прибављања инструкција **load, store** и **add**, ови управљачки сигнали су увек исти, независно од врсте инструкције која се прибавља.

б) стање извршавања инструкције (**execution**), када се управљачки сигнали генеришу под дејством кода операције инструкције. За сваку инструкцију имамо другачији скуп контролно–управљачких сигнала. Свака машинска инструкција се реализује преко другачијег низа елементарних операција (микро операција).

4.3. МИКРОПРОГРАМСКА ОРГАНИЗАЦИЈА УПРАВЉАЧКИХ ЈЕДИНИЦА

Да би логичке мреже, које улазе у састав свих делова рачунара, могле да обављају своју функцију, неопходни су им различити управљачки сигнали које генерише управљачка јединица. Постоје две врсте управљачких сигнала: синхронизациони и функцијски. Синхронизациони сигнали су обично импулсног облика и служе за одређивање различитих временских интервала и усклађивање разних етапа у раду рачунара.

Функцијским сигналима се селектују различите функције на бази скупа синхронизационих сигнала и скупа сигнала добијених из регистара у процесору, или из спољашњег окружења управљачке јединице (па чак и из спољашњег света).

Управљачки сигнали се могу генерисати на два начина, па имамо две врсте организовања управљачке јединице: хардверску и микропрограмску.

Код хардверске организације, управљачки сигнали се генеришу помоћу посебних дигиталних мрежа. Овакве управљачке јединице су брже, али често и врло сложене, а нису ни флексибилне, нити доступне кориснику да их модификује по својим потребама. Овакву организацију имају и **RISC** процесори (**reduced instruction set computers**).

Код микропрограмског или фирмверског генерисања, управљачки сигнали су меморисани у посебној меморији (унутар управљачке јединице), која се зове микропрограмска меморија, и из ње се читају када је то потребно. Овакав управљачки орган је спорији, јер се генерисање управљачких сигнала обавља помоћу посебних програма, **микропрограма**, који се састоје од микроинструкција. За сваку машинску инструкцију постоји посебан низ микроинструкција помоћу којих се генеришу управљачки сигнали. Овакву организацију имају и **CISC** процесори (**complete instruction set computers**).

Да би се извршио неки микропрограм, потребно је, најпре, генерисати почетну адресу микропрограма, и то се ради на бази кода операције и неких спољашних услова. Наредбе микропрограма се обично одвијају једна за другом оним редоследом којим су записане у микропрограмској меморији. И овде се користи посебан регистар, као програмски бројач, чији се садржај после сваке микроинструкције инкрементира.

Као и код корисничких програма, и овде се може одступити од нормалног редоследа извршавања микроинструкција, тј. могући су такозвани микропрограмски скокови под дејством разних узрока. На хијерархијском моделу рачунарског система, видимо да су дигитална логичка кола обухваћена микрокодом, а ниво микрокода је обухваћен машинским језиком. При разматрању фаза прибављања и извршења машинске инструкције, имплицитно смо претпоставили да је сваки код операције директно имплементиран у хардвер. Многи рачунари стварно тако и раде, али знатан је број и

оних који, између машинског језика и дигиталних кола, имају један додатни нижи ниво назван микропрограмски ниво или ниво микрокодирања.

Основна идеја, која се скрива иза микрокодирања, је у томе што се многе ствари које се догађају на нивоу машинских инструкција често понављају: копирање адресе из **PC** у **MAR**, копирање података из меморијске локације у **MBR**, израчунавање адресе података, итд. Обзиром да се ове активности понављају при извођењу сваке елементарне инструкције (машинске инструкције), то значи да су ове активности још једноставније од самих машинских инструкција и да, у ствари, оне представљају елементарне операције.

На основу оваквог приступа можемо уобичајене машинске наредбе, третирати као **макроинструкције** које се извршавају као низови малих корака. Микроинструкције које имплементирају ове кораке запамћене су унутар CPU у специјалној меморији која се зове микропрограмска управљачка меморија. Код неких система, ова меморија је типа ROM и назива се фирмвер (**firmware**). Њен садржај попуњава произвођач CPU–а. Код неких других система, кориснику се допушта да мења садржај ове меморије, и да сам креира микропрограме према својим потребама, тј. да врши микропрограмирање.

Дужина речи процесора (нпр. 8 бита код многих микрорачунара) нема директне везе са дужином речи микроинструкције (нпр, 18 бита, код микропроцесора Intel 8080, слика 4.33). Микроинструкција, управљачка реч, из микропрограмске меморије, преко контролне логике (декодера микроинструкција), генерише управљачке сигнале који се упућују у независне управљачке тачке у свим деловима CPU и рачунара, који омогућавају извођење микрооперација.

а		б			ц				д				е			ф	
0	0	1	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	1	0	0	0	0	1	0	0	0	0	0	0

Слика 4.33. Однос микроинструкције и машинске инструкције

Однос (макро) инструкција – микроинструкција је приказан преко примера извођења машинске инструкције **COMA** (комплементирај садржај акумулатора) за процесор **Intel 8080**. Извођење машинске инструкције **COMA** изазива следеће микрооперације, које се извршавају у посебним интервалима времена:

1. *такт: микроинструкција за пренос садржаја акумулатора преко интерне магистрале у **ALU** у склоп за комплементирање,*

2. такт: микроинструкција за активирање логичке мреже за комплементирање,

3. такт: микроинструкција за пренос комплементираног садржаја преко интерне магистрале у акумулатор.

Из овог примера видимо да се микропрограм за машинску инструкцију **COMA** састоји од три 18–битне микроинструкције. Микроинструкције за процесор **Intel 8080** имају дужину од 18 бита који су груписани у 6 поља (слика 4.33), која имају следеће значење:

а) поље одређује интерну микрооперацију у управљачкој јединици,

б) одређује логичку мрежу у аритметичко–логичкој јединици (нпр. 100–склоп за комплеметирање, или 111–ALU није активан),

ц) одређује одредиште података (нпр. 0100 је акумулатор , 0010 је склоп за комплементирање),

д) одређује регистар или део ALU који представља извор података,

*е) одређује информацију о следећој адреси микроинструкције (нпр. 000 повећај адресу микропрограма за 1, тј. инкрементирај микропрограмски бројач **MPC** за један),*

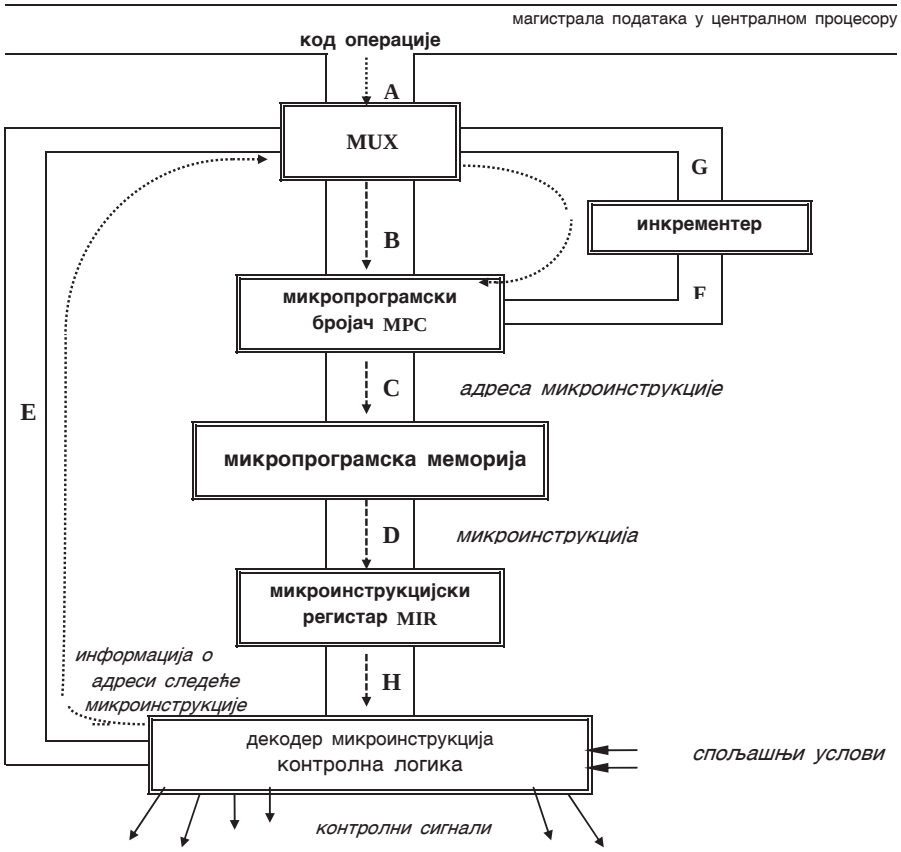
ф) одређује спољашњи услов који учествује у генерисању следеће адресе микроинструкције (нпр. 01 избор бита преноса).

Микропрограми су смештени у брзој интерној меморији (ROM, PLA), са временом приступа од неколико наносекунди. **PLA** (**programable logic array**) је специјална логичка мрежа, која је реализована у облику чипа, а у коју се може уписати произвољна логичка функција. У овакво поље корисник може сам записати микропрограм за свој рачунар.

4.3.1. ФАЗА ПРИБАВЉАЊА МИКРОИНСТРУКЦИЈА

При извођењу микроинструкција постоји читав низ корака налик на оне при извођењу машинских инструкција. Секвенца почиње када **код операције** машинске инструкције, из регистра наредби (**IR**), ступи у део управљачке јединице који врши декодирање кода операције. Овај склоп се назива генератор адресе микропрограма (слика 4.34).

Инструкцијски код садржан у инструкцијском регистру, услови (спољашњи и унутрашњи–интерни, као резултат извођења раније микроинструкције), као и информација о следећој адреси микропрограма, генеришу стварну адресу микропрограма у блоку који се назива генератор адресе микропрограма (који може бити попут мултиплексера (као на слици 4.34), или специјални склоп познат под именом **microprogram sequencer** (као на слици 4.35). Микроинструкција је реч записана на адресираној локацији микропрограмске меморије.



Слика 4.34. Код операције машинске инструкције је почетна адреса микропрограма

Код операције може да се декодира, најчешће тако, да одмах представља почетну адресу низа микроинструкција у управљачкој меморији. У ту сврху се може, уместо декодера кода операције, користити и мултиплексер којим се изабира једна од неколико могућих адреса.

Почетна адреса, која је одређена на основу кода операције машинске наредбе, ступа у микропрограмски бројач (**MPC**), који има исту намену као бројач наредби (**PC**), али истовремено служи и као меморијски адресни регистар. Адреса садржана у **MPC**, преко линија **B**, долази у микро–програмску меморију из које се чита садржај и микроинструкција се копира у микроинструкцијски регистар (**MIR**). И овај регистар има двојаку улогу, тј. уједно представља регистар наредби и меморијски бафер регистар. Двојака улога **MIR** и **MPC** је омогућена тиме што ова меморија садржи само ин–струкције, па нема потребе да се раздвајају подаци од наредби као у

случају оперативне меморије. На крају фазе прибављања (док се микро–инструкција извршава), треба генерисати адресу следеће микро–инструкције у микропрограмској меморији. То се може учинити или инкрементирањем садржаја **MPC,** или на бази неког податка садржаног у текућој инструкцији или на бази спољашњих услова који се у генератор адресе преносе такође преко линија **E**.

Свака микроинструкција садржи најмање једно поље за израчунавање адресе следеће инструкције која се враћа назад у генератор адресе под контролом сигнала генерисаних током фазе извршења микроинструкције.

4.3.2. ФАЗА ИЗВРШЕЊА МИКРОИНСТРУКЦИЈЕ

Када микроинструкција дође у **MIR**, почиње фаза извршења. Као и код машинских наредби, микроинструкција се декодира и дигиталне логичке мреже генеришу управљачке сигнале. Под контролом ових импулса раде сви саставни делови рачунара. Помоћу њих се селектују регистри у CPU који учествују у преносу података и адреса. Ако у CPU има 16 регистара, треба нам најмање 4 линије за њихово селектовање. У неке од ових регистара су записане неке често коришћене константе (нпр. 1,0,–1). Неки управљачки сигнали се користе за селектовање појединих логичких мрежа у ALU које обављају засебне елементарне операције. Ако ALU има четири функције треба нам две линије за избор елементарне операције, а за ALU са 8 функција три линије.

Постоји још један начин модификовања инструкција, који се такође назива микропрограмирање, мада то није у пуном смислу те речи. Но, ако под микропрограмирањем подразумевамо могућност корисника да сам креира и модификује основне инструкције, онда и ово може бити микропрограмирање. Овај поступак се састоји у спајању две или више основних инструкција у једну, при чему се она и изводи као једна инструкција. Нека рецимо треба извести једну за другом две инструкције:

clear (брисање садржаја акумулатора)
clear C (брисање бита преноса, **carry**).

Ако се оне кодирају у бинарном облику као низови шеснаест **0** и **1**:

clear $0\ 111\ 010\ 000\ 000\ 000_{(2)} = 072000_{(8)}$
clear C $0\ 111\ 001\ 000\ 000\ 000_{(2)} = 071000_{(8)}$

Кодови ових инструкција се разликују само у два бита, па ако извршимо логичко сабирање (**ИЛИ** операцију), спајањем ових инструкција добијамо:

clear clear C $0\ 111\ 011\ 000\ 000\ 000_{(2)} = 073000_{(8)}$.

При извођењу ове нове инструкције избрише се и акумулатор и бит преноса у само једном инструкцијском циклусу. На овај начин се и убрзава рад, (штеди се време) и штеди се меморијски простор.

4.4. ХАРДВЕРСКА ОРГАНИЗАЦИЈА УПРАВЉАЧКИХ ЈЕДИНИЦА

Код **RISC** рачунара инструкције су директно уграђене у хардвер и самим тим могу да се извршавају знатно брже, јер машинске инструкције понекад захтевају извођење десетак или чак стотину микроинструкција. Рачунари са архитектуром названом **RISC** (**reduced instruction set computers**), генерално имају мањи број инструкција него они микропрограмирани.

Друга разлика између ове две концепције реализације, односи се на имплементацију виших програмских језика. Микропрограмирани рачунари допуштају креирање микроинструкција које могу директно реализовати неку функцију виших језика. Код ових рачунара сам корисник може програмирати, тј. уписивати низове микроинструкција у микропрограмску меморију. На тај начин сам корисник добија могућност да обликује специфичан скуп инструкција оријентисан ка одговарајућој примени рачунара. Промена скупа инструкција се изводи врло једноставно, заменом микропрограмске меморије, тј. **ROM**–а, па се може уз један рачунар користити више сетова инструкција, зависно од задатака које треба решавати. Ефикасност обраде много зависи од скупа инструкција. **RISC** рачунари, с друге стране, захтевају од језичких преводилаца да генеришу низ инструкција у које се ова функција имплементира.

То значи да микропрограмирани рачунари извршавају многе функције на нивоу хардвера (или **firmware**-а), док **RISC** рачунари имају решење у софтверу. **RISC** рачунари имају фиксну дужину инструкција, па је прибављање инструкција брже. **RISC** рачунари имају ограничен број начина адресирања, па је и израчунавање адресе брже, али, програмирање је онда сложеније.

Цена којом се плаћа ова штедња у броју наредби и начина адресирања огледа се у повећаним проблемима у писању и превођењу програма са виших језика у машински. Но, кориснике то не треба превише да брине, јер су последњих година направљени транслатори (преводиоци) за многе више језике.

4.5. АРХИТЕКТУРА ТЕКУЋЕ ТРАКЕ И ПАРАЛЕЛНА ОБРАДА

Од првог рачунара до данашњих дана, пројектанти хардвера се труде да направе што брже рачунарске системе. Један приступ решењу тог проблема је заснован на чињеници да се многе операције могу разделити на једноставније обраде–подоперације, а затим се неколико подоперација може обављати истовремено.

Извођење сваке инструкције најгрубље се може поделити у две фазе: прибављање и извршавање. Неки процесори (**Intel**) могу за време извршавања прве инструкције прибавити другу инструкцију. За време извршавања друге

се прибави трећа и тако редом. Овај поступак се зове **преклапање фаза**, а време потребно за прибављање инструкција практично нестаје. Основу за технику преклапања фаза представља чињеница да се неке инструкције у време фазе извршавања не обраћају меморији (на пример, инструкције за **Intel** процесоре: **DECA** декрементирај акумулатор, **COMA** комплементирај акумулатор).

Такве инструкције употребљавају само интерну магистралу података, акумулатор и аритметичко–логичку јединицу, док су адресни регистри (бројач наредби, бројач података и сл.) и адресна магистрала слободни. Како адресни подсистем не учествује у извршавању текуће инструкције, он се може употребити за прибављање следеће инструкције. Процесори **Intel 8088** и фамилија процесора **8086** имају две независне јединице:

– *јединицу за извођење инструкција и управљање* ***(execution unit, EU)***

– *јединицу за повезивање са магистралом (**bus interface unit, BIU**), која у свом саставу има и скуп регистара звани **ред** (**queue**) који у себи садржи неколико следећих инструкција програма.*

Али, могуће је још веће убрзавање рада процесора. Посматрајмо опет низ машинских инструкција којима се решава проблем сабирања два броја:

sum = num1 + num2 .

адреса	садржај	значење садржаја локације
100:	**load data**	пренеси из меморије у акумулатор податак
101:	**$1A_{(16)}$**	са локације 1A
102:	**add data**	сабери садржај акумулатора са податком
103:	**$1B_{(16)}$**	са локације 1B
104:	**store data**	пренеси из акумулатора податак у меморију
105:	**$1C_{(16)}$**	на локацију са адресом 1C

Ако се пак извођење сваке од инструкција подели на четири корака, потпроцеса: прибављање кода операције, декодирање, прибављање операнда, извршавање. Ако било који потпроцес захтева један временски интервал, онда се овај мали низ обавља у **12** тактова у стандардном извођењу. Ако се сваки од потпроцеса може обавити одвојено, онда се овај низ инструкција (програм) може обавити као следећи низ корака:

1. *Прибављање* **load** *инструкције,*
2. *Декодирање* **load** *инструкције,*
   Прибављање **add** *инструкције.*

3. *Прибављање операнда за* **load** *инструкцију,*
   Декодирање **add** *инструкције,*
   Прибављање **store** *инструкције.*

4. *Копирање податка за* **load** *инструкцију,*
   Прибављање операнда за **add** *инструкцију,*
   Декодирање **store** *инструкције,*
   Прибављање инструкције на адреси $\mathbf{106}_{(16)}$,

5. *Сабирање, тј. извршавање инструкције* **add**,
   Прибављање операнда за **store** *инструкцију,*
   Декодирање инструкције на $\mathbf{106}_{(16)}$,
   Прибављање инструкције на $\mathbf{108}_{(16)}$,

6. *Смештање резултата, тј. извођење инструкције* **store sum**.

. . .

Коришћењем преклапања независних обрада у поткорацима, потребно укупно време за извођење програма скраћује се са **12** на **6** интервала времена, односно обрада је обављена за половину времена. Теоријски ова уштеда времена може бити и већа, тј. може се свести на четвртину првобитног времена. Овај начин обраде се зове архитектура текуће траке (**pipelining**). При овом поступку процес обраде се дели на низ потпроцеса који се извршавају са преклапањем у аутономним јединицама за обраду.

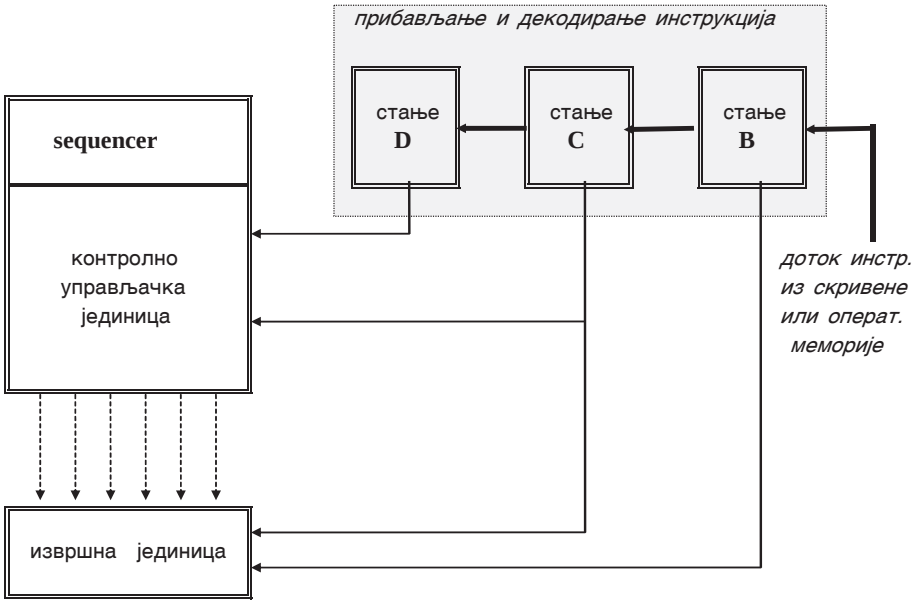
Рачунар са архитектуром текуће траке, који дели процес на четири аутономна потпроцеса, може, теоретски, да четвороструко повећа брзину обраде. То је, међутим, нереално очекивање, пре свега због употребе заједничких магистрала и инструкција скока и гранања.

На слици 4.35 приказана је архитектура типа текуће траке коју има **Motorola MC 68020** процесор. Овај процесор (микропроцесор) проводи инструкцију у току извршења кроз три стања **B, C** и **D.** Стање **D** даје управљачкој јединици потпуно декодирану инструкцију. У току извршења инструкције, податак из стања **C** може да послужи као константа или као проширење кода операције.

Архитектуру текуће траке поседовао је и рачунар **CDC STAR-100**, који је имао два паралелна процесора за 64–битне операнде у покретном зарезу. Процесори су радили паралелно и обрађивали два скупа 32–битних операнада, јер је сваки имао по две линије текуће траке које су давале резултат обраде сваких 40 наносекунди. Овај систем је дакле био способан да обави 10^8 32–битних операција у покретном зарезу у секунди (**100 megaflops, millions of floating-point operations per second**).

Овај рачунар је користио меморијске речи од 512 бита. Меморија капацитета 4 мегабајта, направљена је од феритних језгара са временом приступа од 1.28 микросекунди, и била је подељена на 32 модула (капацитета 2048 речи) којима се независно приступало. Обзиром да је процесор сваких 40 ns

обрађивао 128 бита, то је у једном меморијском циклусу преко магистрале преношено 128*4*32=16384 бита.



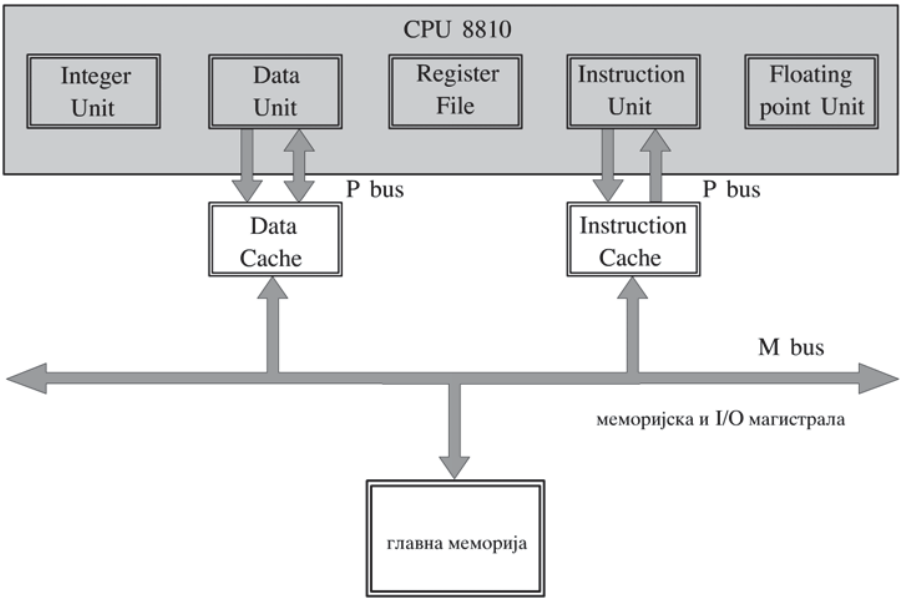
Слика 4.35. Процесор **MC 68020** обрађује податке на текућој траци

На слици 4.36. приказана је типична организација једног система Моторола 88000 чија је главна компонента централни процессор 88100. Овај процесор састоји се од пет јединица: јединице података (дата унит) и инструкцијске јединице (инструцтион унит) које одговорне sa прибављање инструкција и података из главне меморије, затим има две извршне јединице: јединицу за целобројно рачунање и јединицу за аритметику у покретној тачки (која обавља и дељење целих бројева) и скуп 32–битних регистара (регистер филе) преко којег међусобно комуницирају четири претходно поменуте јединице. Пренос података и инструкција између централног процесора и скривених меморија обавља се истовремено, дуж две засебне магистрале (које се зову **P bus**) и преклапа се са извршавањем у јединицама за обраду (**execution units**).

4.6. МУЛТИПРОЦЕСОРСКИ СИСТЕМИ

Један други приступ повећања перфомансi подразумева коришћење неко–лико процесора у исто време, или паралелно. Рачунари са паралелним процесирањем користе од две до неколико стотина јединица за обраду које

раде једновремено. На слици 4.37 приказана је једна могућа конфигурација паралелног процесора са три ALU јединице и са три скупа регистра који користе заједничку меморију и управљачку јединицу.



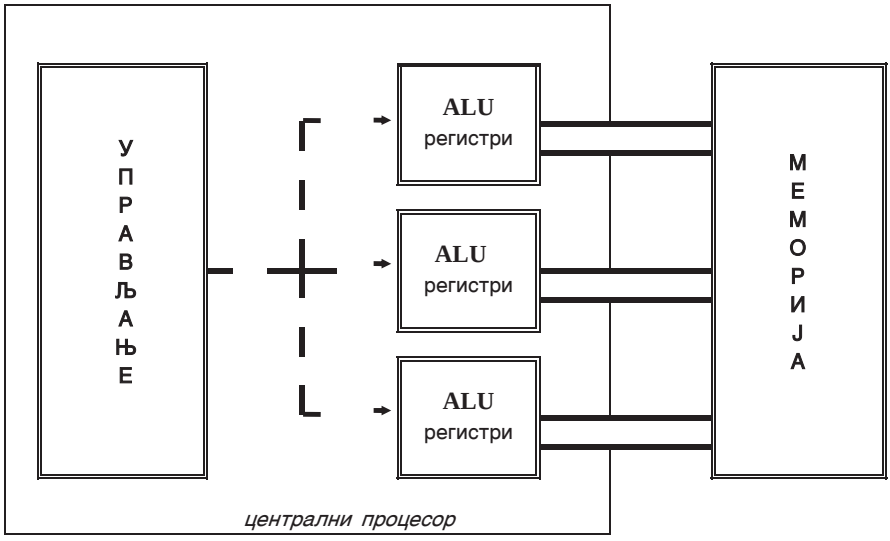
Слика 4.36. Типична организација микрочунарског система 88000

Оваква конфигурација позната је под именом бит–одрезак (**bit-sliced**) архитектура. Свака ALU јединица обрађује по један део податка, па се комбиновањем неколико јединица практично може реализовати рачунар са произвољном дужином речи. Ако свака од ALU на слици 4.37 обрађује по 4 бита, онда овај процесор има реч од 12 бита. На сличан начин је реализован рачунар **IBM 370**, **Honeywell Level 6**, **Univac 1100**, **CDC 6600** са десет извршних јединица као и експериметални рачунар **ILLIAC IV** (**Burroughs Corporation**).

ILLIAC IV је имао једну контролну јединицу и 64 процесне јединице. Свака процесна јединица садржи сопствену процесну меморију и аритметичко–логичку јединицу која је, поред уобичајених операција, обављала и инструкције над 64–битним бројевима у покретном зарезу. Свака процесна меморија је имала 2048 речи од 64 бита.

Рачунарски системи могу имати и више од једног централног процесора. Постоје две врсте оваквих система: мултипроцесорски системи и рачунарске мреже. Кад један рачунар садржи два или више централних процесора, онда се он зове се мултипроцесорски системи. Овакви системи су пред–

одређени за мултипроцесирање (истовремено извршавање две или више инструкција истог програма), али и за мултипрограмски рад (паралелно одвијање више програма).

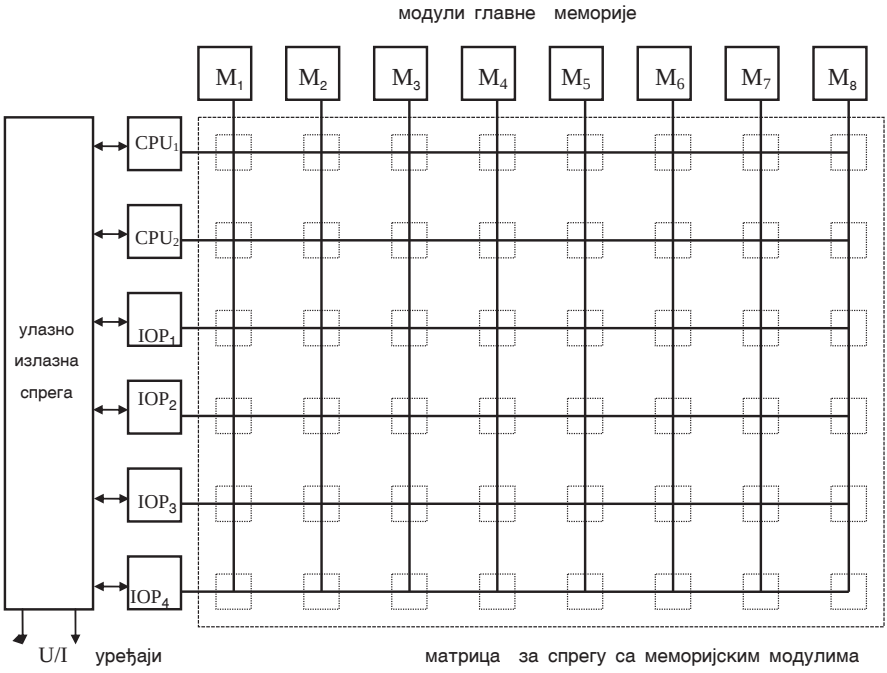


Слика 4.37. Једноставна конфигурација паралелног процесора

На слици 4.38. приказан је вишепроцесорски систем са два централна процесора и са четири периферијска процесора, од којих сваки има одређени степен аутономије. Пример на слици 4.38 представља конфигурацију рачунара **Burroughs B5000** и **B5500**. Главна меморија је подељена на осам делова, тј. модула, којима се може независно приступати. Меморије су повезане са процесорима помоћу спрежне мреже која допушта истовремени приступ за сваки процесор понаособ једном (различитом) модулу меморије.

Међутим, повећање броја процесора не доводи обавезно до пропорционалног повећања ефикасности обраде. То је пре свега последица чињенице да алгоритми најчешће предвиђају обављање неке обраде корак по корак. Знатно веће повећање ефиксности постиже се код такозваних рекурзивних поступака обраде.

Посебан тип вишепроцесорског система чине системи са дистрибуираном обрадом (**прилог Б**), тј. рачунарске мреже, које представљају рачунарски систем који у свом раду обједињује више десетина, па и више стотина независних рачунара (најчешће микрорачунара), који су повезани комуникационом мрежом. Овакви системи су погодни за мултипрограмски рад и паралелну обраду.



Слика 4.38. Вишепроцесорска конфигурација рачунара **Burroughs B5000**

4.7. ЗАКЉУЧАК

Независно од тога како су инструкције имплементиране, многи кораци захтевају извођење врло простих операција. Свака инструкција мора бити прибављена из меморије и декодирана. Извршење ма које инструкције захтева генерисање и временску контролу великог броја сигнала који управљају радом дигиталних логичких кола која чине хардвер рачунара.

Извођење инструкција често се може посматрати на два нивоа. Виши ниво, видљив за обичне програмере, је ниво уобичајеног машинског језика. Многи рачунари (CISC) имају и нижи ниво, ниво микрокодирања, који користи низове микроинструкција (микропрограме) за извођење сваке машинске наредбе. Друга врста процесора (RISC) има за сваку машинску инструкцију засебну логичку мрежу која омогућава извршавање инструкције у једном такту, што знатно убрзава рад рачунара.

Архитектура текуће траке и разних видова паралелне обраде, омогућавају истовремено извођење две или више инструкција. Модерни рачунари су способни да извршавају широк спектар аритметичких и логичких функција. Они, такође,

користе бројне технике за лоцирање, тј. израчунавање апсолутне адресе података и инструкција у меморији.

4.8. ПИТАЊА

1. Који су основни делови генералисане архитектуре рачунара?
2. Које кораке треба извршити да би се подаци изнели из рачунара?
3. У чему се разликује фаза прибављања инструкција од прибављања податка?
4. Из којих делова се састоји машинска инструкција (формати инструкција)?
5. Како се врши прибављање машинских инструкција?
6. Која ограничења и хардвер омогућавају смањење адресности наредби?
7. Написати програм за сабирање два броја за једноадресну машину.
8. Описати кораке који чине фазу прибављања машинске инструкције, и да ли зависе од врсте инструкције?.
9. Описати кораке који чине фазу извршавања инструкције **load**.
10. У чему се разликују фазе извршавања инструкција load и **add**?
11. Описати кораке који чине фазу извршавања инструкције **store**.
12. Шта су микрооперације и микроинструкције?
13. Описати фазе прибављања и извршавања микроинструкције.
14. Шта су RISC процесори, које су им предности и мане?
15. Како се убрзава рад рачунара, и које врсте паралелне обраде постоје?
16. Шта представља мултипроцесирање а шта мултипрограмирање?

4.9. КЉУЧНЕ РЕЧИ

- адресна магистрала (**address bus**)
- акумулатор (**accumulator**)
- архитектура рачунара,
- фаза прибављања (**fetch cycle**)
- фаза извођења **(execution cycle**)
- фирмвер (**firmware**)
- главна, оперативна меморија (**main storage, operating memory**)
- инструкција, наредба (**instruction**)
- код операције (**opcode**)
- магистрала података (**data bus**)
- меморијски адресни регистар (**MAR**)
- меморијски бафер регистар (**MBR**)
- микроинструкција (**microinstruction**)
- микроинструкцијска меморија (**microinstruction storage**)
- микропрограм (**microprogram**)
- операнд (**operand**)
- паралелна обрада (**paralel processing**)

- програмски бројач (**PC**)
- регистар наредби (**IR**),
- **RISC (reduced instruction set computer)**
- текућа трака, цевовод, (**pipeline**)
- управљачка јединица (**control unit**)
- преклапање фаза
- дистрибуирана обрада (**distributed computing**)
- **CISC** (**complete instruction set computers**)

5. ВРСТЕ НАРЕДБИ И НАЧИНИ АДРЕСИРАЊА

Ако дигиталне мреже посматрамо као основне саставне јединице рачунарског система, онда програме можемо посматрати као средство које те саставне делове оживљава и повезује у једну функционалну целину. Рачунар извршава програме техником цикличних понављања фаза прибављања и извршења инструкција.

Централни процесор је део рачунарског система чији је један од основних задатака да прибави и декодира прибављену инструкцију, и генерише одговарајући низ управљачких сигнала потребних за извођење те инструкције. Ови сигнали управљају преносом података преко магистрала, надзиру рад аритметичко–логичке јединице, побуђују одговарајуће регистре и саставне делове рачунара итд. Дакле, CPU генерише и распоређује такт и управљачке сигнале којима се усклађују све активности не само рачунара, већ и свих других делова рачунарског система.

Главни саставни делови CPU–а су: регистри, аритметичко–логичка јединица и управљачка јединица који су међусобно повезани помоћу магистрала.

Логичке мреже које улазе у састав централног процесора одређују скуп операција које могу бити непосредно изведене. Ове операције се могу груписати по различитим критеријумима, па самим тим постоји више различитих подела за један исти скуп инструкција.

Другу важну ствар представљају начини за одређивање апсолутних адреса података и инструкција у меморији. Велики број начина адресирања (**addressing modes**) је развијен и уграђен у разне рачунаре. Типови начина адресирања, који су на располагању за дати рачунар, директно утичу на имплементацију функција виших програмских језика у рачунар.

Дигиталне мреже, које су неопходне за имплементацију различитих наредби и начина адресирања, налазе се у централној процесорској јединици (**CPU**). CPU је одговорна за прибављање, декодирање и извршење инструкција, као и за усклађивање преноса података у и из улазно–излазних јединица. Информације које улазе у, и излазе из CPU–а, путују дуж магистрала.

Многи рачунари имају одвојене магистрале за све главне типове информација (за податке, за адресе и управљачке сигнале). Могућа су и другачија решења. Разматрање саставних делова и функције централног процесора почећемо од регистара који улазе у његов састав.

5.1. РЕГИСТРИ

У опису фазе прибављања инструкција (и операнада) сусрели смо се са следећим регистрима:

– *програмским бројачем, бројачем наредби (**PC**),*
– *меморијским адресним регистром (**MAR**),*
– *меморијским бафер регистром (међурегистар за прихват података **MBR**),*
– *регистром наредби, инструкцијским регистром (**IR**).*

Програмски бројач (**PC**) садржи адресу меморијске локације у којој је записана следећа инструкција која ће бити прибављена у наредном циклусу. Програмски бројач се инкрементира током фазе прибављања, али и у току приступања операндима за време фазе извршења. Садржај овог регистра се, такође, мења у току извршења наредби скока и прескока, затим за време обраде прекида и при позивању потпрограма.

Меморијски адресни регистар (**MAR**) садржи адресу меморијске локације којој се приступа. У рачунару обично постоје два оваква регистра: један у саставу CPU, а други у саставу меморије, а међу собом су повезани адресном магистралом.

Бројач података је адресни регистар података (**DC**), а садржи у себи адресу меморијске локације у којој се налази операнд или податак.

Регистар за прихват података (**data buffer register**) служи за привремено меморисање информација које улазе или излазе из CPU. Регистри сличне намене налази се и у меморији, (**MBR**), а и у свим улазно–излазним склоповима, и истовремено служе за повезивање свих саставних делова рачунара на магистралу података.

Регистар наредби (**IR**) прихвата код операције из инструкције која је прибављена, и која ће следећа бити извршена, тј. на основу које ће управљачка јединица генерисати временске и управљачке сигнале.
Већина аритметичких и логичких операција користи регистре познате под називом **акумулатори**. Неки процесори имају један (**Intel** 8080 и надаље), а неки више акумулатора (**Motorola** 6800 има два акумулатора, а **Unisys** 1100 их има 16). Они имају двојаку улогу, користе се за привремено прихватање једног од улазних података у ALU а, такође, и за прихватање резултата обраде, односно излаза из ALU. Акумулатори имају средишњу улогу у преносу података у, и из централног процесора. Како се акумулатори налазе у CPU (уз саму ALU, а по мишљењу многих аутора акумулатор је

регистар у саставу ALU), то је време приступа подацима у акумулаторима изузетно кратко, па више акумулатора омогућава већу брзину обраде података. Код рачунара организованих око једне магистрале, на један од улаза ALU увек се поставља акумулатор који служи да заједно са привременим регистрима одвоји улаз ALU од излаза **ALU**. Ово је потребно да не би дошло до нежељеног утицаја излаза **ALU** на улаз (**critical race**), јер су и улаз и излаз **ALU** на истој магистрали.

5.1.1. БРОЈАЧИ И ПОМЕРАЧКИ РЕГИСТРИ

Бројачи (**counters**) су дигиталне мреже чији се садржај, под дејством импулса, може мењати у одређеном редоследу, тако да се може интерпретирати као низ сукцесивних бројева. Како низ бројева може монотоно да расте или опада, то и бројачи могу да броје унапред и уназад (инкрементирање, декрементирање). Најчешће се користе бројачи који броје у бинарном и декадном бројном систему.

Померачки регистар (**shifter**) је скуп меморијских кола која су повезана тако да меморисани податак може да се помера од једног до другог меморијског кола. Померање се може вршити од бита мање тежине на бит веће тежине (**улево**) или са бита веће тежине на бит мање тежине (**удесно**).

5.1.2. ПОКАЗИВАЧ СТЕКА

Показивач стека (**stack pointer**) садржи адресу дела меморијског простора познатог под именом стек, а који се користи за привремено смештање података или адреса. Најчешће су то садржаји неких регистара из **CPU**. Садржај овог регистра се аутоматски увећава и умањује, тј. инкрементира и декрементира при извођењу неких операција са стек меморијом. Стек меморија и овај регистар користе се приликом обраде прекида, код операција са потпрограмима и за чување специјалне групе података таквих да се прво користи последњи уписани податак (**LIFO - Last In First Out**).

5.1.3. ИНДЕКС РЕГИСТРИ

Индекс регистри се најчешће користе при израчунавању физичке адресе меморијских локација, када се садржају неког од ових регистара додаје садржај из поља операнда у инструкцији. Овакав приступ меморији се зове индексно адресирање. Ови регистри се често користе за контролу програмских петљи. У ове регистре се могу уписивати подаци, а такође се њихов садржај може инкрементирати и декрементирати.

5.1.4. БАЗНИ И СЕГМЕНТНИ РЕГИСТРИ

Неки рачунари имају специјалне базне регистре, док рачунари засновани на интеловим микропроцесорима (**Intel** серија **8086** и надаље), имају низ регистара који се зову сегментни регистри. И базни и сегментни регистри се користе при израчунавању меморијских адреса инструкција и података. Бројач наредби, бројач података, показивач стека, индекс регистри, базни и сегментни регистри чине групу адресних регистара, тј. служе за израчунавање адреса.

5.1.5. РЕГИСТРИ ОПШТЕ НАМЕНЕ

Процесори **Intel 8086** и надаље имају групу од четири регистра за податке (AX, BX, CX, DX који се састоје из два дела: горњег H и доњег L), слика 5.1, и сваки од њих се може употребити при извршавању аритметичких и логичких операција. **IBM 370** има 16 регистара опште намене. Они се могу користити и као акумулатори, и као индекс регистри, или као базни регистри. Рачунари **Unisys 1100** имају четири регистра који се могу користити као индекс регистри или као акумулатори.

Вишенаменски регистри омогућавају програмеру велику флексибилност у решавању проблема. У њима се привремено записују различити подаци, а приступа им се знатно брже и једноставније него меморији. Ови регистри знатно убрзавају рад рачунара: **Прво**, време приступа тих регистара је изузетно кратко, и износи десетак наносекунди, па чак и мање. **Друго**, инструкције су краће, а код организације око магистрале, адреса регистра је део проширеног кода операције, тј. нема поља операнда.

Но, понекад се ови регистри организују као мала меморија, и онда се она зове **scratchpad store** (прибелешка), меморија за привремено чување. У оваквој организацији, регистар се бира помоћу адресе, али су и ове инструкције краће од меморијских.

5.1.6. РЕГИСТРИ СТАЊА ПРОЦЕСОРА И РЕГИСТРИ СТАЊА ПРОГРАМА

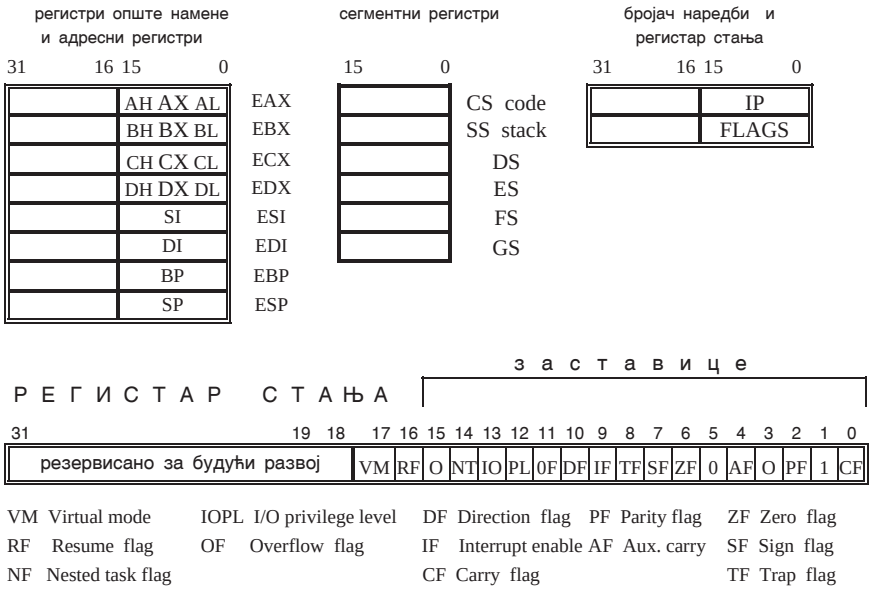
Регистар стања (**SR, Status Register**) или регистар кодова услова (**Condition Code, CCR**) је посебан регистар код кога појединачни битови приказују различита стања која могу настати у току обраде података. Ови битови се називају **заставице** (**flags**).

Све операције које користе **ALU** могу постављати неке од заставица да би означиле да је израчунавање дало, на пример нулу или негативан резултат. Неке друге заставице могу указивати да се појавио један бит вишка на крају регистра (пренос, **carry**), или да је резултат сувише велики да би стао

у регистар (препуњење, **overflow**), или да је пак резултат (број у покретном зарезу) сувише мали (поткорачење, **underflow**). Та стања могу утицати на даље одвијање програма, или рада рачунара, јер програмер може провером стања одређене заставице да усмери одвијање програма, и промени редослед извршавања инструкција.

Не постоје стандарди шта све и у ком облику садржи регистар који описује стање процесора и програма. Слика 5.1. показује регистре опште намене и адресне регистре, сегментне регистре и регистар стања (регистар заставица, **flag register**) у **Intel 80386** микропроцесору.

На слици 5.2. приказан је регистар стања за **Motorola MC 68020** микропроцесор. Овај регистар је подељен на кориснички и системски бајт.



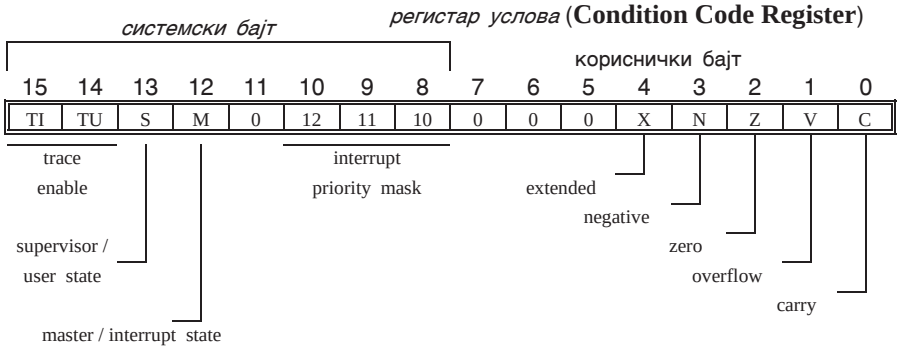
Слика 5.1. Регистри опште намене и регистар стања процесора Intel **80386**

Код неких рачунара се постојање услова приказује у кодираном облику (као код **IBM 360/ 370**), а код неких је за сваки услов резервисана посебна заставица. Поменућемо неке од заставица на основу чијег садржаја се најчешће контролише ток програма, тј. редослед извођења инструкција.

Заставица C (**carry**) која означава пренос, елемент је регистра стања (или регистра услова, **condition code register, CCR**), и има две функције:

– *садржи бит преноса након извођења аритметичких операција (пренос из бита највеће тежине, **MSB**),*

– *заставица* **C** *се употребљава као помоћни бит при операцијама померања, или ротирања садржаја акумулатора.*



Слика 5.2. Поља у регистру стања микропроцесора **Motorola MC68020**

Заставица V (**overflow**) је бит који означава да је дошло до прекорачења вредности (ако се ради о 8–битном процесору **V = 1,** означава да је резултат операције по апсолутној вредности прекорачио границу од 127).

Заставица N (**negative**) се употребљава за индикацију негативног резултата након извршења аритметичке операције, и још неких инструкција.

Заставица Z (**zero**) је бит који јавља да је резултат аритметичке операције **0**. Такође се употребљава и за логичке операције, нпр. упоређивање, где се, уколико су два операнда једнака, поставља заставица (**Z = 1**).

Заставица H (**nibble**) означава пренос из доњег полубајта у горњи полубајт (тетрада), и употребљава се у **BCD** операцијама, обзиром да се у коду **BCD** употребљавају четири бита за приказ једне декадне цифре.

Заставица P (**parity**) је бит парности и налази се у регистру услова код процесора **Intel 8080** за испитивање тачности преноса података. Већина процесора нема бит **P** у регистру стања, пошто је функција генерисања и испитивања парности пренета у надлежност програмабилне улазно–излазне јединице или улазно–излазног канала.

Заставица I (**interrupt enable**) је бит прекида и обично се употребљава у специјалним програмима за обраду изненадних ситуација које су настале у рачунарском систему или ван њега. Бит **I** се поставља у случају забране неких врста прекида (**I=1**). Заставица **I** се код процесора назива прекидна маска, јер се у случају да је **I=1,** и ако се појави захтев за прекид нижег приоритета, прекид текућег програма се неће догодити.

На слици 5.3. приказан је сличан регистар који се налазио у **COSMAC CDP 1802** микропроцесору. Он нема класичан регистар стања већ има седам

заставица стања специфичне намене: **DF** је заставица преноса. **IE** је заставица за дозволу или маскирање прекида, заставица **Q** коју постављају спољашња логичка кола (али је мењају и неке инструкције) и такозване **U/I** заставице **EF1, EF2, EF3, EF4** које искључиво постављају спољашња логичка кола и могу се само читати. Последњих пет заставица се користи за реализацију инструкција условног скока тј. гранања.

Слика 5.3. Реч стања процесора за микропроцесор **CDP 1802**

Рачунари са процесором **MC 68020**, као и већина модерних рачунара, садрже у меморији, у исто време, део оперативног система и један или више програма. Централни процесор (**CPU)** и оперативни систем, морају увек знати где се сваки од текућих корисничких програма налази, где се налазе подаци, и која инструкција се управо извршава.

CPU мора, такође, сачувати траг о специјалним условима који могу утицати на текућу или будућу инструкцију. Ове информације се, код неких процесора, памте у регистру стања програма (**Program Status Word, PSW**), или у регистру стања процесора (**Processor Status Register, PSR**), који су сасвим слични регистру стања (**Status Register, SR**).

Но, као што смо већ напоменули, многи рачунари не постављају директно заставице, већ кодирају стање процесора и програма. У том случају ти кодови морају садржати и информације које нам дају поменуте заставице. На слици 5.4 приказана је дупла реч стања програма (**PSW** од 64 бита), за рачунаре IBM 360 / 370 серије.

Битови од 0 до 7 су системска маска за контролу разних **I/O** прекида, и означавају да ли централни процесор може прихватити прекид са неког канала. Битови 0 до 5, ако су укључени (постављени на 1), означавају да је са канала 0 до 5 допуштен прекид.

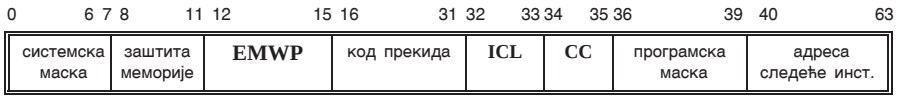
Бит 7, ако је укључен, означава да су допуштени спољни прекиди. Ако је укључен бит 6, сви канали могу сигнализирати прекиде. Ако су улазно–излазни прекиди маскирани, хардвер аутоматски задржава прекид за касније, када се маска скине и прекиди допусте.

Битови од 8 до 11 представљају кључ за заштиту меморије, и користе се за контролу приступа разним деловима меморије. Поље је дугачко 4 бита. У то поље уписана је бинарна шифра (број) који се мора приликом сваког

приступа у меморију упоредити с бинарним бројем који у меморији постоји. Поље заштитног кључа служи за заштиту појединих делова оперативне меморије од недопуштеног приступа.

Наиме, код овог система меморија је подељена у блокове од 2048 бајта и сваки такав блок има придружену шифру дужине 4 бита. Ако се, дакле, кључ у **PSW**–у и кључ у меморији подударају, корисниковом програму је дозвољен приступ у тај део меморије.

Приступ се, такође, може селективно регулисати, тако да је у неке блокове меморије могуће уписивати податке и читати их, док за друге блокове може бити допуштено само читање садржаја.



Слика 5.4. Регистар стања програма за рачунаре **IBM** серије 360 / 370

Бит 12 (**E**) код рачунара IBM 360/370 означава да је рачунар у проширеном режиму рада (**extended mode**). Бит **M** (13) и битови за програмску маску (36–39), показују врсту грешака које могу да буду занемарене. Бит **W** означава да је процесор у стању извођења (када је **W=0**), или у стању застоја (**W=1**).

Бит **P** означава да ли је процесор у стању извођења проблемског програма (**P=0**), или изводи неки од програма из састава оперативног система (**P=1**), односно бит **15** указује да је рачунар у корисничком режиму (**problem state**), или у привилегованом–супервизор режиму рада (**supervisor mode**). Битови 32, 33 (**ILC**) зову се код дужине инструкције, и показују величину (дужину) последње машинске наредбе изражену у полуречима (16 бита = 2 бајта = 1 полуреч).

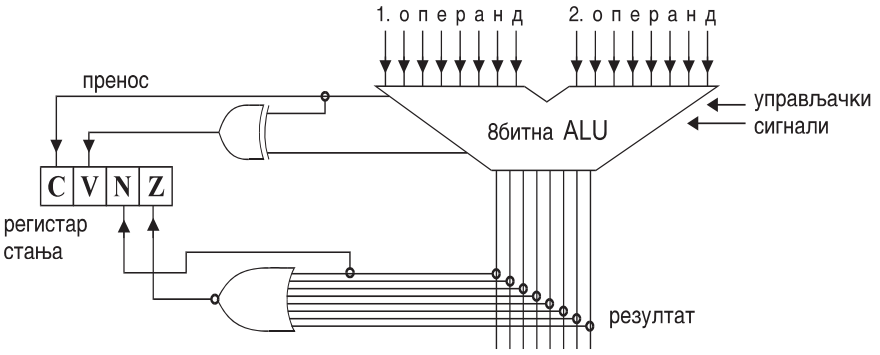
Поље 34, 35 (**CC**) је тзв. код услова, или индикатор стања након извођења неких инструкција, и указује да ли је резултат претходне аритметичке операције био једнак нули, негативан, позитиван или сувише велики. Код рачунара IBM, из серије 360/370, не постављају се директно заставице за сваки услов посебно, већ се појава одређених услова збирно кодира.

PSW регистар се налази у централном процесору и служи за контролу редоследа извођења инструкција и комплетну индикацију стања система у односу на програм који се тренутно изводи. Наиме, битови од 40 до 63 садрже адресу следеће инструкције која треба да се изврши, односно имају улогу бројача наредби.

Тај део **PSW** регистра код **IBM** рачунара има исту улогу као бројач наредби **(Program Counter, PC**), или инструкцијски адресни регистар (**IAS**) код неких других рачунара.

5.2. АРИТМЕТИЧКО - ЛОГИЧКА ЈЕДИНИЦА

Аритметичко – логичка јединица је вишефункцијски дигитални склоп. **ALU** је централна компонента у фази извођења инструкција у рачунару. Обично има два улаза на које се доводе операнди **A** и **B**, и један излаз где се добија резултат (слика 5.5).



Слика 5.5. Аритметичко–логичка јединица **ALU**

Посматрана на нивоу машинског језика, **ALU** обавља мноштво функција под контролом низа управљачких сигнала. Сем непосредног израчунавања резултата, **ALU** на излазу генерише и информације о стањима која се смештају у регистар стања. Неке операције сем операнада захтевају, као улазни податак, и бит преноса (типично за обраду вишецифарских декадних и вишеречних бинарних бројева).

Аритметичко–логичка јединица у свом саставу има дигитална кола за: комплементирање, померање, инкрементирање, декрементирање, сабирање, итд. **ALU** обавља аритметичке и логичке операције над бинарним бројевима, а такође и њихово ротирање (померање). Број операција које се непосредно изводе (тј. за које постоје посебна дигитална кола) није велики.

Неки аутори у састав **ALU** укључују и акумулатор и регистар стања, због њихове функционалне повезаности. Они у састав аритметичко–логичке јединице укључују и регистар података (**Data Register, DR**), као и привремене регистре.

Сем у инструкцијама које у себи садрже неку аритметичку или логичку операцију, аритметичко–логичка јединица се користи и при различитим поступцима израчунавања адреса података и инструкција.

У наредном делу размотрићемо инструкције које су типичне за већину рачунара, као и најчешће начине адресирања.

5.3. ТИПОВИ МАШИНСКИХ ИНСТРУКЦИЈА

Скуп машинских инструкција дефинише основне операције које рачунар може да изведе. Инструкције се могу класификовати по различитим критеријумима, а једна од могућих подела је базирана на типу операције која се изводи применом дате инструкције. Тако се инструкције рачунара могу сврстати у три групе:

– *аритметичко–логичке инструкције,*

– *инструкције за пренос података,*

– *инструкције за контролу тока програма, тј. управљачке инструкције.*

5.3.1. АРИТМЕТИЧКЕ И ЛОГИЧКЕ ИНСТРУКЦИЈЕ

Ове инструкције се деле у две групе, зависно од тога колико операнада садрже:

– *унарне, са једним операндом (**monadic instructions**),*

– *бинарне, са два операнда (**diadic instructions**).*

5.3.1.1 Унарне инструкције

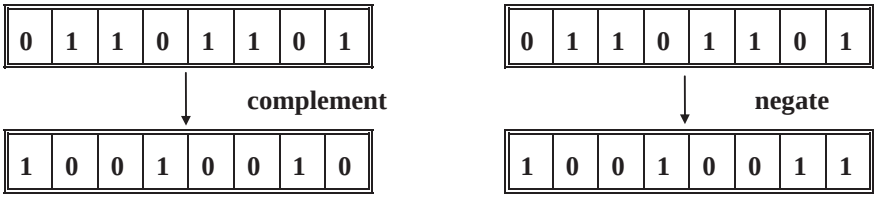
Типичне унарне инструкције су: **постављање** (**set**), **брисање** (**clear**), **комплементирање** (**complement**), **негација** (**negate**), **инкрементирање** (**increment**), **декрементирање** (**decrement**), **померање** (**shift**) и **ротација** (**rotate**).

Set и **Clear** инструкције се користе за промену, тј. за постављање жељеног садржаја појединих битова у регистар стања, читавих регистара у **CPU** и меморијских локација. Процесор може имати инструкцију за брисање садржаја читавог регистра (садржај нула), или ће вршити брисање (**reset**) појединих заставица: **clear carry**, **clear overflow** итд. Нормално, на располагању су и инструкције за постављање јединице у поједине заставице: **set carry**, **set decimal mode**, **set interrupt** итд. На слици 5.6. је приказан ефекат инструкције **CLC** за процесор **Motorola MC 6800** (**clear carry**) на промену садржаја дела регистра стања, тј. на заставицу за пренос. Уочимо да је ова инструкција утицала само на садржај бита за пренос, док су садржаји осталих заставица остали непромењени.

Напоменимо да неки микропроцесори аутоматски врше сабирање са учешћем бита за пренос. Да не би случајни стари садржај овог бита изазвао појаву нетачног резултата, то ова инструкција, по правилу, претходи инструкцији за сабирање.

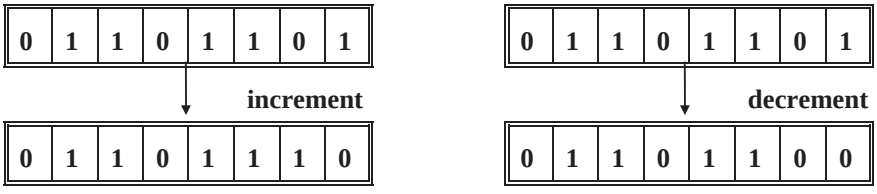
Слика 5.6. Утицај инструкције **clear carry** на регистар стања

Комплементирање и негација су инструкције које се најчешће односе на садржај целог регистра, а сличног су дејства. Наиме, процесор **Intel 8080** нема инструкцију типа **clear carry** већ се брисање заставице за пренос врши помоћу инструкције **CMC - complement carry**. Комплементирање регистра мења све нула битове у јединице, а јединице у нуле, а као резултат се добија први комплемент (инверзни код) броја. Рачунарски системи који раде у другом комплементу имају и инструкцију негације, која као резултат даје други комплемент броја (слика 5.7.).



Слика 5.7. Инструкције комплементирања и негације у рачунару са другим комплементом

Инкрементирање и декрементирање су инструкције за повећање и смањење њиховог операнда за један (слика 5.8.). У неким рачунарима, ове инструкције се примењују само на регистре који се налазе у **CPU**, док се код других могу применити и на садржаје меморијских локација.



Слика 5.8. Инструкције повећања и смањења садржаја регистра за један

Померање и ротирање су инструкције које померају битове унутар једног регистра, у једном или другом смеру. Микропроцесори обично померају

садржај регистра за једну позицију у једном такту, док велики рачунари могу у једном такту померити садржај за више позиција. Ове операције се врше најчешће над садржајем акумулатора.

Претпоставимо да акумулатор садржи следеће битове:

(10011001)

Померањем садржаја овог регистра за једно место лево, добија се следећи резултат:

1 (0011001**?**).

Уочимо да је крајње леви бит (**MSB**) изван регистра, и да је крајње десни бит (**LSB**) недефинисане вредности.

Многи рачунари попуњавају испражњене позиције нулама. Бит који на левом крају излази, биће за неке рачунаре изгубљен, а код других ће бити прихваћен као бит преноса.

Посматрајмо шта се догађа при померању садржаја регистра за једно место удесно. Нека је почетни садржај акумулатора (01110110), после померања удесно биће (**?**0111011)**0**. **LSB** ће бити или изгубљен, или ће се уписати у заставицу преноса (у овом примеру заставица преноса је **0**). Код померања удесно остаје недефинисан крајње леви бит (**MSB**). Једно од решења је да се увек у овај бит уписује нула, па би онда нови садржај акумулатора био (**0**0111011). Овакав померај, када се у **MSB** уписује нула, назива се логички померај удесно.

Померање садржаја регистра има низ значајних примена. Прва је испитивање садржаја бита у регистру. Ако нам треба да сазнамо шта се налази у трећем биту са леве стране, једноставно померимо садржај регистра за три места улево, и бит чији садржај истражујемо биће смештен у заставицу преноса, а потом се испита садржај ове заставице, (слика 5.9).

пренос	**регистар**	
(**?**)	(10101101)	основни садржај
(**1**)	(01011010)	први померај
(**0**)	(10110100)	други померај
(**1**)	(01101000)	трећи померај

Слика 5.9. Коришћење померања за испитивање садржаја бита

Друга примена инструкције померања је за множење и дељење броја са бројем 2^n, слика 5.10. Померање садржаја регистра за једно место улево има исто својство као множење садржаја регистра са 2 (2^1). Померање за

два места улево множи основни број са 4 (2^2), а померање за **n** места улево множи садржај регистра са 2^n.

Померање удесно има супротан ефекат, тј. одговара дељењу садржаја регистра са 2 или 2^n, где је **n** број места за колико се помера садржај. Ово је целобројно дељење, без остатка.

Тачније, остатак није сачуван у регистру, али је информација о постојању остатка сачувана у заставици за пренос.

(00000101) = 5		(00010100) = 20
(00001010) = 10	померање за 1 место	(00001010) = 10 (20:2)
(00010100) = 20	померање за 2 места	(00000101) = 5 (20:4)
(00101000) = 40	померање за 3 места	(00000010) = 2 (20:8)
(а) померање улево		(б) померање удесно

Слика 5.10. Множење и дељење са 2^n помоћу померања бинарног броја за **n** места улево и удесно

Но, посматрајмо шта би се десило кад би поступак дељења са 2 помоћу померања удесно, применили на број (11110011). После померања удесно на описани начин, у **MSB** се уписује нула, добићемо број (01111001).

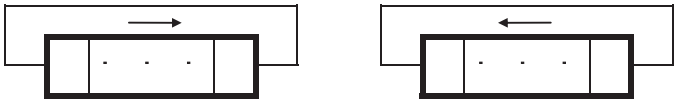
Ако је ово рачунар који ради у првом комплементу, онда је почетни број био негативан, тачније то је $-12_{(10)}$. Дељењем са 2 требало би да добијемо број $-6_{(10)}$. Међутим, након померања добијен је број који је позитиван и једнак је $121_{(10)}$.

Да би превазишли овај проблем, многи рачунари имају и другу врсту помераја удесно, такозвани аритметички померај. Код овог помераја у **MSB** (бит знака), при померању удесно, уписује се његова претходна вредност, тако да се задржава знак броја. Ако је основни број био позитиван, онда је **MSB** нула, и након померања удесно, опет се уписује нула.

Ако је број био негативан, при померању удесно, у крајње леви бит треба уписати јединицу, тј. и резултат ће бити негативан. Ако применимо ово правило, након померања регистра (11110011) удесно, добијамо (11111001), што је у првом комплементу број $-6_{(10)}$, па је резултат тачан.

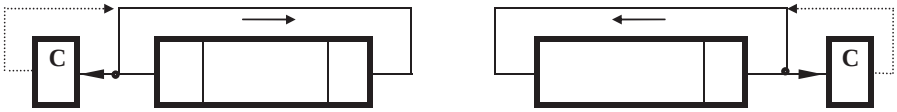
Неки процесори имају и аритметички и логички померај. Такви су и **Motorola MC 68000**, **Intel 8086** (и каснији типови ових фамилија процесора). **VAX** рачунари имају пак само аритметичке помераје, а смер померања зависи од знака (померање удесно за шест места означава се са **-6**, а улево са **+6**). Микропроцесор **RCA COSMAC 1802** има само логички померај.

Ротирање бинарног броја је слично померању, само се бит који напушта регистар уписује на место испражњеног (недефинисаног) бита, на другом крају регистра, слика 5.11.



Слика 5.11. Типична ротација садржаја регистра

Ротирање садржаја регистра (11001100), за једно место улево, даје резултат (10011001), (јединица са левог краја уписана је на десном крају). Ротирање садржаја регистра (00001111) за једно место удесно, даје резултат (10000111). И ротирање се, као и померање, користи за испитивање садржаја неког бита. Тада се бит који напушта регистар уписује у заставицу преноса и уједно враћа на други крај регистра, као на слици 5.12. Код неких процесора (**Motorola MC6800**) и сам бит преноса учествује у ротацији (испрекидане линије на слици 5.12.).



Слика 5.12. Ротација садржаја регистра са памћењем преноса

Код неких процесора се у процесу померања и ротације уместо заставице за пренос, користи такозвани **link** регистар. **Link** је једнобитни регистар и има исту функцију као заставица за пренос.

5.3.1.2 Бинарне инструкције

Бинарне инструкције имају два операнда, од којих је најчешће бар један у неком од акумулатора (или другим регистрима у **CPU**), док је други у неком од регистара меморије. Ове операције су знатно брже и компактније, ако су оба операнда у регистрима унутар централног процесора. Операције које се врше над операндима могу бити аритметичке или логичке, па постоје и две групе ових инструкција.

5.3.1.2.1 БИНАРНЕ ЛОГИЧКЕ ИНСТРУКЦИЈЕ

Ове инструкције извршавају фамилију операција **И**, **ИЛИ**, и **ексклузивно ИЛИ** (**AND, OR, XOR**). Логичке инструкције обрађују све битове у регистру,

али се примењују на сваки пар бита понаособ, и за сваки пар генеришу посебан резултат, који не утиче на резултат у другим битима – овде нема преноса између два суседна разреда у регистру.

Ако регистар **A** садржи низ $11010011_{(2)}$, а регистар **B** низ $01001110_{(2)}$, онда су резултати логичких операција и операције сабирања дати на слици 5.13.

	A AND B	**A OR B**	**A XOR B**	**A + B**
A	**11010011**	**11010011**	**11010011**	**11010011**
B	**01001110**	**01001110**	**01001110**	**01001110**
пренос				**11 011110**
резултат	**01000010**	**11011111**	**1 0011101**	**100100001**

Слика 5.13. Однос логичких операција над садржајем регистра и сабирања бинарних бројева

Резултат инструкције **AND** има јединице само у оним битима где су оба бита и у **A** и у **B** јединица. Инструкција **OR** даје у резултату јединицу, у свим битима где је јединица било у **A** било у **B**, а **XOR** даје у резултату јединицу, ако је тај бит јединица, само у **A** или само у **B**. Уочимо да ни један резултат логичких операција није ни налик на збир бинарних бројева који чак има девет цифара у резултату.

Операција **AND** се може користити за издвајање садржаја дела регистра. Претпоставимо, на пример, да треба да издвојимо доњу цифру BCD броја из акумулатора, чији је садржај $10010101_{(2)}$.

Ово се може извести тако што направимо маску која садржи јединицу у битима који нас интересују (нпр. $00001111_{(2)}$ је маска за доњу тетраду), и извршимо операцију **AND** маске и садржаја акумулатора, слика 5.14(а).

		10 010101	*оригинални садржај акумулатора*
акумулатор	10 010101	01011001	*садржај после ротације за 4 места*
маска	00001111	00001111	маска
AND	0000**0101**	0000**1001**	**AND**
а) издвајање доње тетраде		(б) издвајање горње тетраде ротирањем	

Слика 5.14. Коришћење **AND** операције за издвајање **BCD** цифара из регистра

Обзиром да операција **AND** даје јединице само на позицијама где оба операнда имају јединице, то резултат копира само доњи полубајт (тетраду) акумулатора, јер је маска у горњем полубајту нула, па је самим тим и у резултату тај полубајт нула. Процес прављења маски и њеног логичког множења са регистром може се комбиновати са ротацијом и померањем. На

слици 5.14(б) је приказано издвајање горњег полубајта, помоћу маске за доњи полу–бајт, и уз помоћ ротације акумулатора за 4 места. На сличан начин може се издвојити било који бит, постављањем одговарајуће маске (све нуле сем на позицији од интереса где се поставља јединица).

XOR операција има такође разне примене, а једна од њих је и начин за уписивање броја нула у регистар (**ексклузивно ИЛИ** садржаја регистра са самим собом), но најчешће се користи за комплементирање садржаја жељених битова. **XOR** се користи за инвертовање неког бита, помоћу маске где су сви битови нула, само је јединица на позицији од интереса. Ако хоћемо да инвертујемо садржаје битова $\mathbf{b}_4$ и $\mathbf{b}_5$ користимо маску 00110000, која на местима која инвертујемо има јединице а остало су нуле, слика 5.15.

XOR	10**01**0111	оригинални садржај регистра
	00**11**0000	маска
	10**10**0111	резултат

Слика 5.15. Инвертовање битова помоћу **XOR** инструкције

Операција **OR** се може користити за паковање података у једну целину. Нека регистар **B** садржи број $00000100_{(2)}$, тј $(4_{(10)})$, а **C** регистар садржи број $00001001_{(2)}$ $(9_{(10)})$. Ово су две BCD цифре које хоћемо да спојимо у један BCD број $49_{(10)}$ у регистру **A**. Да бисмо то постигли треба извршити следећи низ операција:

- извршити операцију **clear A**, којом се у акумулатор уписује нула,
- извршити операцију **A OR B**, у регистру **A** добијамо **0000100**,
- померити садржај регистра **A** за четири места улево; добијемо **01000000** у **A**,
- извршити операцију **A OR C;** у регистру **A** добијемо коначни резултат, тј. низ **01001001 BCD** односно $49_{(10)}$.

Овај поступак се зове паковање **BCD** цифара, јер се **BCD** бројеви памте по принципу **1 цифра 1 бајт**, у такозваном распакованом облику, а обрађују се у пакованом облику тј. **2 цифре у 1-ом бајту**.

Погледајмо сада један могући пример употребе рачунара и логичких операција за управљање неким производним процесом. Нека се један 16–то битни регистар користи за управљање (укључивање и искључивање) 16 прекидача: S_0, S_1, . . . , S_{14}, S_{15} (прекидач–**switch**). Ови прекидачи се могу користити за укључивање–искључивање: светиљки, мотора, машина, грејалица и сл. Стање сваког прекидача може се описати једним битом: јединица значи да је прекидач укључен, а нула да је искључен. Нека су у неком случајном тренутку времена укључени само прекидачи S_1, S_7, S_{12} и S_{14}. Овом стању прекидача одговара следећи садржај једног регистра, на пример акумулатора:

A_{cc}	0	1	0	1	0	0	0	0	1	0	0	0	0	0	1	0

Почетно стање акумулатора

1. Селективно укључење појединих прекидача, на пример $S_8 \div S_{11}$ без промене стања осталих прекидача може се извршити логичком операцијом **ИЛИ** (**OR**) између садржаја акумулатора и маске која има јединице на местима $M_8 \div M_{11}$ а у осталим разредима су нуле:

M	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0

Маска за укључивање прекидача $S_8 \div S_{11}$

Треба извршити операцију **ИЛИ** над садржајима акумулатора и регистра за маску:

	0	1	0	1	0	0	0	0	1	0	0	0	0	0	1	0
	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
OR	0	1	0	1	1	1	1	1	1	0	0	0	0	0	1	0

Почетно стање акумулатора

Маска за укључивање прекидача $S_8 \div S_{11}$

Ново стање акумулатора

2. Селективно искључење појединих прекидача, на пример $S_7 \div S_{10}$ без промене стања осталих прекидача може се извршити логичком операцијом **И** (**AND**) између садржаја акумулатора и маске која има нуле на местима $M_7 \div M_{10}$ а у осталим разредима су јединице:

	0	1	0	1	1	1	1	1	1	0	0	0	0	0	1	0
	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1
AND	0	1	0	1	1	0	0	0	0	0	0	0	0	0	1	0

Старо стање акумулатора

Маска за искључивање прекидача $S_7 \div S_{10}$

Ново стање акумулатора

3. Селективно инвертовање стања појединих прекидача, на пример $S_4 \div S_{11}$ без промене стања осталих прекидача може се извршити логичком операцијом ексклузивно **ИЛИ** (**XOR**) између садржаја акумулатора и маске која има јединице на местима $M_4 \div M_{11}$ а у осталим разредима су нуле:

	0	1	0	1	1	0	0	0	0	0	0	0	0	0	1	0
	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
XOR	0	1	0	1	0	1	1	1	1	1	1	1	0	0	1	0

Старо стање акумулатора

Маска за инвертовање прекидача $S_4 \div S_{11}$

Ново стање акумулатора

4. Инвертовање стања свих прекидача може се извршити логичком операцијом ексклузивно **ИЛИ** (**XOR**) садржаја акумулатора и маске која има јединице у свим разредима:

	0	1	0	1	0	1	1	1	1	1	1	1	0	0	1	0
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
XOR	1	0	1	0	1	0	0	0	0	0	0	0	1	1	0	1

Старо стање акумулатора

Маска за инвертовање свих прекидача

Ново стање акумулатора

5.3.1.2.2 АРИТМЕТИЧКЕ БИНАРНЕ ИНСТРУКЦИЈЕ

Аритметичке (бинарне) инструкције механизују основне аритметичке операције (а пре свега сабирање), над низовима нула и јединица, који представљају бинарне кодове бројева. С обзиром да постоји више врста бројева, чак и унутар једног рачунара, онда је могућ и велики број различитих облика ових инструкција. Оно што је сасвим извесно јесте да

сви рачунари имају инструкцију за сабирање два цела позитивна бинарна броја. Рачунари који раде у првом и другом комплементу, најчешће одузимају бројеве правећи комплемент умањиоца, и при томе врше сабирање. **Unisys 1100** рачунари, на пример, имају инструкцију **add negative** да би заменили одузимање.

Вредност бројева који могу бити сабрани једном инструкцијом, природно је ограничена величином регистра, тј. осмобитна машина сабира у једном интервалу времена осам бита, 16–битна сабира 16 бита одједном. Сабирање већих бројева захтева низ инструкција и коришћење заставице за пренос. На пример, сабирање хексадецималних бројева $\mathbf{0DE2}_{(16)}$ и $\mathbf{1D95}_{(16)}$ у осмобитном рачунару захтева две инструкције.

Прво се саберу бајти мање тежине ($\mathbf{E2}_{16} + \mathbf{95}_{16}$), и добије се делимична сума $\mathbf{177}_{(16)}$. Две задње цифре чине збир **77** који се памти, а јединица представља пренос у следећу инструкцију где се сабирају бајти веће тежине, тј. $\mathbf{0D}_{16} + \mathbf{1D}_{16} + \mathbf{1}$, и добија се $\mathbf{2B}_{(16)}$.

Коначан резултат је $\mathbf{2B77}_{(16)}$, а добија се спајањем два парцијална збира.

Многи рачунари могу да сабирају **BCD** бројеве, као што сабирају природно кодиране бинарне бројеве. Неки процесори (и микропроцесори) имају специјалне инструкције за операције над **BCD** бројевима, док други користе заставице у регистру стања, да промене значење уобичајене инструкције сабирања и одузимања (**6502** процесор), и прећу на **BCD** аритметику. **MC68020** рачунари имају посебне инструкције за сабирање и одузимање бинарних бројева (**ADD** и **SUB**), а посебне за **BCD** бројеве (**ABCD-Add Decimal with Extend** и **SBCD-Subtract Decimal with Extend**). **VAX** рачунари имају специјалне инструкције за сабирање пакованих **BCD** бројева, бинарних бројева и бројева у покретном зарезу.

Већина осмобитних микропроцесора нема инструкције за множење и дељење, већ их реализује програмски. Шеснаестобитни микропроцесори имају инструкције за целобројно множење и дељење. Велики рачунари имају инструкције за целобројно и **BCD** множење и дељење, а такође и за извођење аритметичких операција над бројевима у покретном зарезу. Велики број микропроцесора за аритметику у покретном зарезу користи специјалне математичке процесоре, копроцесоре.

Велики рачунари имају пун сет инструкција у покретном зарезу, укључујући сабирање, одузимање, множење и дељење. Они, такође, имају и инструкције за конверзију целих бројева у бројеве у покретном зарезу, и обратно, инструкције за нормализацију бројева у покретном зарезу итд. **VAX** рачунари имају више од 50 инструкција које подржавају конверзију бројева из једног облика у други.

5.3.2. ИНСТРУКЦИЈЕ ПРЕНОСА ПОДАТАКА

У ову групу инструкција спадају инструкције које преносе податке између регистара у централном процесору, између регистара централног процесора и меморије, затим између регистара у процесору и стека, као и у излазне и из улазних уређаја. Јасно је да се ради о великом броју врло разнородних типова инструкција које се међу собом разликују по броју операнада, дужини, времену потребном за извршавање, итд.

5.3.2.1 Пренос података између регистара

Ове инструкције имају највећу брзину извођења у односу на остале инструкције преноса. Саме инструкције су краће и код неких рачунара у самом коду операције садржане су информације о регистрима који учествују у преносу, и о смеру преноса података. Тада читава инструкција има само поље кода операције. Код других рачунара, у инструкцији се експлицитно указује на регистре који учествују у преносу преко њихових адреса. Ове инструкције се зову **register-to-register instruction**.

Примери ових инструкција за микропроцесор **Motorola MC6800** су:

TAB (**T**ransfer from Accumulator **A** to Accumulator **B**),
TBA (**T**ransfer from Accumulator **B** to Accumulator **A**),
TAP (**T**ransfer from Accumulator **A** to Condition Code Register),
TXS (**T**ransfer from Inde**x** Register to **S**tack pointer), итд.

Примери ових инструкција за микропроцесор **Intel 8080**:

MOV A, E – пренеси садржај регистра **E** у регистар **A**,
MOV D, D – пренеси садржај регистра **D** у регистар **D** (делује као наредба **NOP, no-operation**).

5.3.2.2 Пренос података између меморије и регистара

Ове инструкције врше пренос података између регистара у централном процесору и меморијских локација. Постоје две врсте ових инструкција, од којих једне преносе податке из меморије у неки од регистара (инструкције типа **load**), и друге које преносе податке из неког регистра **CPU** у неку меморијску локацију (инструкције типа **store**).

Неки рачунари за оба типа преноса имају инструкције истог типа **move x, y** која означава пренос података са локације ***y*** (извор, **source**) на локацију ***x*** (одредиште, **destination**).

Инструкција **MOV AX, STUDENT** код **Intel** процесора означава пренос података са меморијске локације са симболичком адресом **STUDENT** у регистар **AX** (акумулатор). Инструкција **MOV STUDENT**, **AX** преноси податак из акумулатора у меморијску локацију **STUDENT**.

Процесор **MC 68020** има специјалну инструкцију преноса која допушта да се садржаји неколико регистара копирају у меморију у једном интервалу времена, што је основа за мултипрограмски режим рада. Већина процесора (и микропроцесора) има инструкције за пренос блока података. На пример, код процесора **Z80** инструкција **LDIR** (**LoaD Increment and Repeat** - напуни, инкрементирај и понављај).

Ове инструкције имају и врло различите начине адресирања меморијских локација, што омогућава велику флексибилност приступања подацима.

5.3.2.3 Операције са стеком

Као што је већ описано, стек је једно подручје меморије које се користи за привремено памћење. На локацију на врху стека се указује помоћу адресе у регистру, који се зове показивач стека **(Stack Pointer**, **SP**). Смештање података у, и узимање података из стека, захтева коришћење специјалних инструкција које, такође, могу да иновирају и садржај показивача стека.

Смештање података у стек најчешће се назива **push**, док се узимање података са стека зове **pull** или **pop**. Смештање података на стек (**push**) аутоматски декрементира садржај показивача стека, тако да он најчешће указује на прву слободну локацију (а код неких рачунара на врх стека, тј. на последњу заузету локацију).

Показивач стека се повећава за један (инкрементира), непосредно пре читања меморијске речи из стека. Затим се податак чита са локације чија је адреса у показивачу стека, и након читања та локације се сматра слободном за упис, тј. постаје врх стека.

Код рачунара код којих показивач стека садржи адресу последње заузете локације у стеку, пре уписа се прво садржај показивача стека декрементира, а затим се уписује податак на врх стека.

При читању података са стека, прво се прочита садржај локације чија је адреса у показивачу, а затим се садржај показивача стека инкрементира.

Пример ових инструкција за процесор MC6800:

PSH A запиши садржај акумулатора **A** на стек,
PSH B запиши садржај акумулатора **B** на стек,
PUL A узми са стека податак и смести га у акумулатор **A**,
PUL B узми са стека податак и смести га у акумулатор **B**.

5.3.3. УЛАЗНО – ИЗЛАЗНЕ ОПЕРАЦИЈЕ

Улазно–излазне (**U/I**) операције могу се посматрати као облик преноса података. Неки рачунари, као **MC 68020** и **VAX**, посматрају улазно – излазни простор исто као и меморију. Свакој **U/I** јединици је придружена једна или више меморијских адреса. Онда је пренос података ка излазној јединици исто што и упис у меморију, а пренос из улазне јединице исто што и читање из меморије. Код оваквог, тзв. меморијски мапираног **U/I** простора, за приступање улазним и излазним локацијама можемо користити све инструкције и начине адресирања, као код приступа меморији.

Други рачунари, укључујући оне централне у систему, имају потпуно одвојене **U/I** операције од меморијских операција. Ово је тзв. изоловани или програмски контролисани **U/I** простор. Ови рачунарски системи имају посебне инструкције попут **IN** и **OUT**, или користе специјалне процесоре (који се понекад зову канали) за обављање свих улазних и излазних активности. **IBM 370** системи, имају на пример, **start I/O, halt I/O** и **test I/O** инструкције, за контролу и комуницирање са каналима.

Инструкције које контролишу улазно–излазне операције, сматрају се често привилегованим због безбедности и због сложености ових операција, и може их користити само оперативни систем. Када корисник, односно његов програм, жели да приступи некој улазној или излазној јединици, он се обрати оперативном систему помоћу једне специјалне инструкције. Оперативни систем након тога преузима контролу улаза и излаза података уместо корисничког програма.

Кориснички програм који жели да изврши **I/O** операцију, мора да тражи услугу од оперативног система. Овај захтев је познат као **supervisor call**, или прекид (**interrupt**).

Другим речима привремено се прекида рад корисничког програма, а контролу над процесором (и рачунарским системом у целини) преузима један други програм из оперативног система. Кад оперативни систем обави задатак, контрола се враћа корисничком програму, и то тачно оној инструкцији која је требала да буде извршена да није било прекида.

5.3.4. ИНСТРУКЦИЈЕ ЗА УПРАВЉАЊЕ ТОКОМ ПРОГРАМА

Принцип рада модерних рачунара заснован је на претпоставци да су инструкције смештене у меморију секвенцијално, тј. инструкције су смештене у суседне меморијске локације, оним редоследом којим треба да буду извршене. Овај секвенцијални режим рада може бити измењен уз помоћ измене садржаја програмског бројача (**PC**). С обзиром да садржај

програмског бројача одређује која ће се инструкција извршити у следећем интервалу времена, могуће је управљати током извршења програма мењањем његовог садржаја.

Инструкције које омогућавају промену садржаја бројача наредби, а тиме и скретање тока извођења програма називају се **управљачке инструкције**.

Управљачке инструкције скрећу ток извођења програма или **безусловно**, или након извођења теста (на пример, над заставицама у регистру стања), тј. **условно**. Овај условни начин скретања тока програма оличава способност процесора да доноси одлуке.

Управљачке инструкције могу се поделити у три подгрупе:

– *инструкције безусловног скока (**jump**) или гранања (**branch**),*

– *инструкције условног скока или гранања,*

– *специјалне управљачке инструкције.*

5.3.4.1 Инструкције безусловног скока

Ове инструкције преусмеравају извођење програма на одређену адресу без испитивања било каквих услова. Изводе се једноставним уписом адресе меморијске локације, у којој се налази следећа инструкција у бројач наредби. Постоји више могућих начина одређивања адресе следеће инструкције, па самим тим има и више типова инструкција безусловног скока. Коришћење ових инструкција је врло често извор озбиљних грешака у програму, па се не препоручује њихова честа употреба.

5.3.4.2 Инструкције условног скока или гранања

Ове инструкције омогућавају скок (гранање програма), само ако је претходном обрадом података испуњен неки услов. Те инструкције испитују задовољење постављених услова, и у зависности од исхода испитивања мењају или не мењају садржај програмских бројача.

Услови могу бити разнородни, на пример, број у акумулатору: позитиван, нула или негативан, стање у неком биту регистра стања нула или јединица, итд. Даљи ток одвијања програма зависи од резултата претходне обраде података, па се унапред не зна којим ће путем кренути даље одвијање програма.

Постоји више врста инструкција условног скока, но све оне се у суштини заснивају на тестирању појединих битова у регистру стања. Посебну групу ових инструкција чине инструкције *прескока* (**skip**). Инструкције за

тестирање и гранање, поред наведених врста, могу бити: **скочи ако има преноса**, **скочи ако нема преноса**, **скочи ако је препуњен акумулатор** итд. Неки рачунари допуштају директно поређење садржаја два регистра у **CPU,** или између регистра и меморијске локације, и допуштају гранање ако је поређење истинито. Инструкције овог типа допуштају гранање програма ако је, на пример, садржај регистра **A** већи од садржаја регистра **B** и слично.

Други рачунари имају инструкције које врше само поређење и постављање заставица (уместо директног гранања), а заставице се касније могу тестирати инструкцијом гранања. Но, у свим случајевима гранање се извршава изменом садржаја **PC**.

Претпоставимо да бројач наредби садржи број $3721_{(16)}$ на крају фазе прибављања инструкције, и нека је то инструкција условног скока која каже скочи на $3740_{(16)}$, ако постоји пренос. Током извршења ове инструкције, **CPU** ће испитати заставицу преноса у регистру стања, и пренети операнд $3740_{(16)}$ у програмски бројач ако је заставица постављена, тј. ако је бит преноса јединица. Ако **CPU** открије да је заставица преноса нула, тј. нема преноса, ништа се неће догодити са садржајем бројача наредби, и биће извршена инструкција на коју је бројач наредби указивао, тј. инструкција са локације $3721_{(16).}$

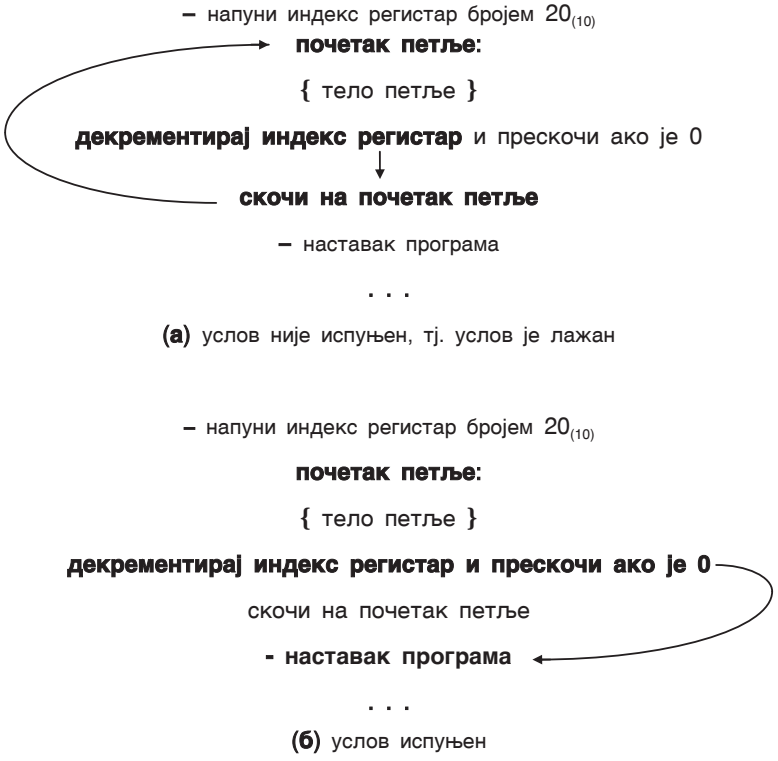
Програмске петље су варијанта инструкција гранања по принципу **тестирај и прескочи**, па се често и називају **инструкције прескока (test-and-skip instructions**). Ове инструкције тестирају услов и прескачу прву наредну инструкцију ако је услов задовољен, тј. истинит. Следећа инструкција ће бити извршена само ако је резултат теста нетачан, тј. услов није испуњен. Ова идеја се може комбиновати са инструкцијама инкрементирања и декрементирања, па се добијају инструкције типа: **инкрементирај и прескочи** (ако је испуњен услов), или **декрементирај и прескочи** (ако је резултат нула). На слици 5.16 је показано како се овакве инструкције могу искористити за реализацију програмских петљи.

Ово је пример петље која је контролисана помоћу једног од индекс регистара. Почетна вредност $20_{(10)}$ (број понављања петље), смешта се у индекс регистар, и отпочиње се петља.

После обраде низа инструкција које чине тело петље, индекс регистар се декрементира и анализира се резултат. Ако је резултат различит од нуле, нема прескока и извршава се следећа инструкција, а она изазива скок програма назад на меморијску локацију на којој се налази почетак петље (слика 5.16–а).

Када се индекс регистар декрементира на нулу, услов је задовољен и догађа се прескок следеће инструкције, тј. прескаче се инструкција безусловног скока на почетак петље, и петља се завршава.

На овај начин се постиже вишеструко понављање обраде дефинисане у низу инструкција које чине тело петље.



Слика 5.16. Реализација петље инструкцијом декрементирај и прескочи следећу инструкцију ако је резултат декрементирања нула

Помоћу оваквих машинских инструкција реализују се програмске петље у вишим програмским језицима. Међутим, неки процесори имају машинске инструкције за контролу понављања у програмским петљама. Тако фамилија процесора **Intel 80x86** има инструкције **LOOP, LOOPE, LOOPZ**.

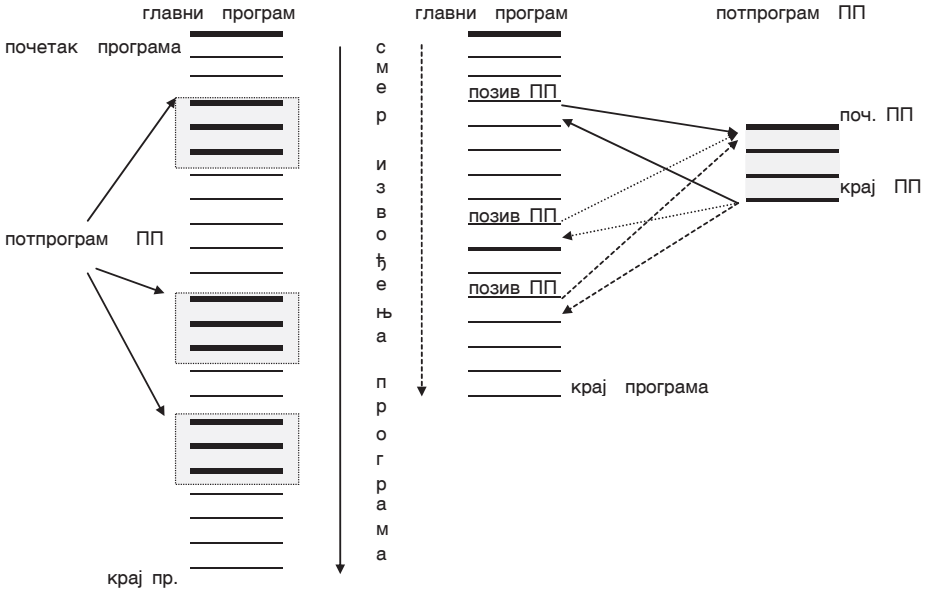
5.3.4.3 Специјалне управљачке инструкције

Подскуп специјалних управљачких инструкција чине:

– *инструкције за управљање потпрограмима,*
– *инструкције за управљање стањима процесора,*
– *инструкције за управљање прекидом.*

5.3.4.3.1 ИНСТРУКЦИЈЕ ЗА УПРАВЉАЊЕ ПОТПРОГРАМИМА

Прављење великих програма је дуготрајан и скуп процес. Због тога се велики програми најчешће разбијају у мање, заокружене целине које се називају модули, који се засебно пишу и тестирају, и затим се повезују са другим модулима у програмску целину која се зове програмски пакет. Понекад је међутим, у оквиру чак и једног модула, потребно једну те исту обраду (низ инструкција) применити више пута над разним почетним подацима. Тај низ инструкција можемо груписати тако да представља једну целину, тј. потпрограм (**subroutine**, **subprogram**). Потпрограм је програмска структура која решава одређену целину, односно задатак. Он омогућава вишеструко повезивање и извршавање одређеног низа инструкција у различитим тачкама главног програма. То се може урадити на два начина: сваки пут на месту где је то потребно тај низ се упише у програм (отворени потпрограми, макроинструкције), слика 5.17. а), или се низ инструкција једном напише као засебна целина, а програм га позива да се изврши онда када је то потребно (затворени потпрограми), слика 5.17. б). Ми ћемо обратити пажњу на затворене потпрограме.



а) програм са отвореним потпрограмима б) програм са затвореним потпрограмима

Слика 5.17. Отворени и затворени потпрограми

Постоје две групе инструкција за рад са потпрограмима, и то су: инструкције за **позивање потпрограма** (**call**) и инструкције за **враћање из потпрограма**, тј. за повратак у програм (**return**). Позивање потпрограма се разликује од инструкције скока по томе што, рачунар сачува, најчешће у

стеку, текући садржај програмског бројача, пре него што упише нову адресу у **PC**. Захваљујући томе, програм може да се врати на инструкцију која следи иза инструкције позива потпрограма у било које време, тако што поново упише у бројач наредби сачувану стару вредност, тј. адресу инструкције која следи иза позива потпрограма.

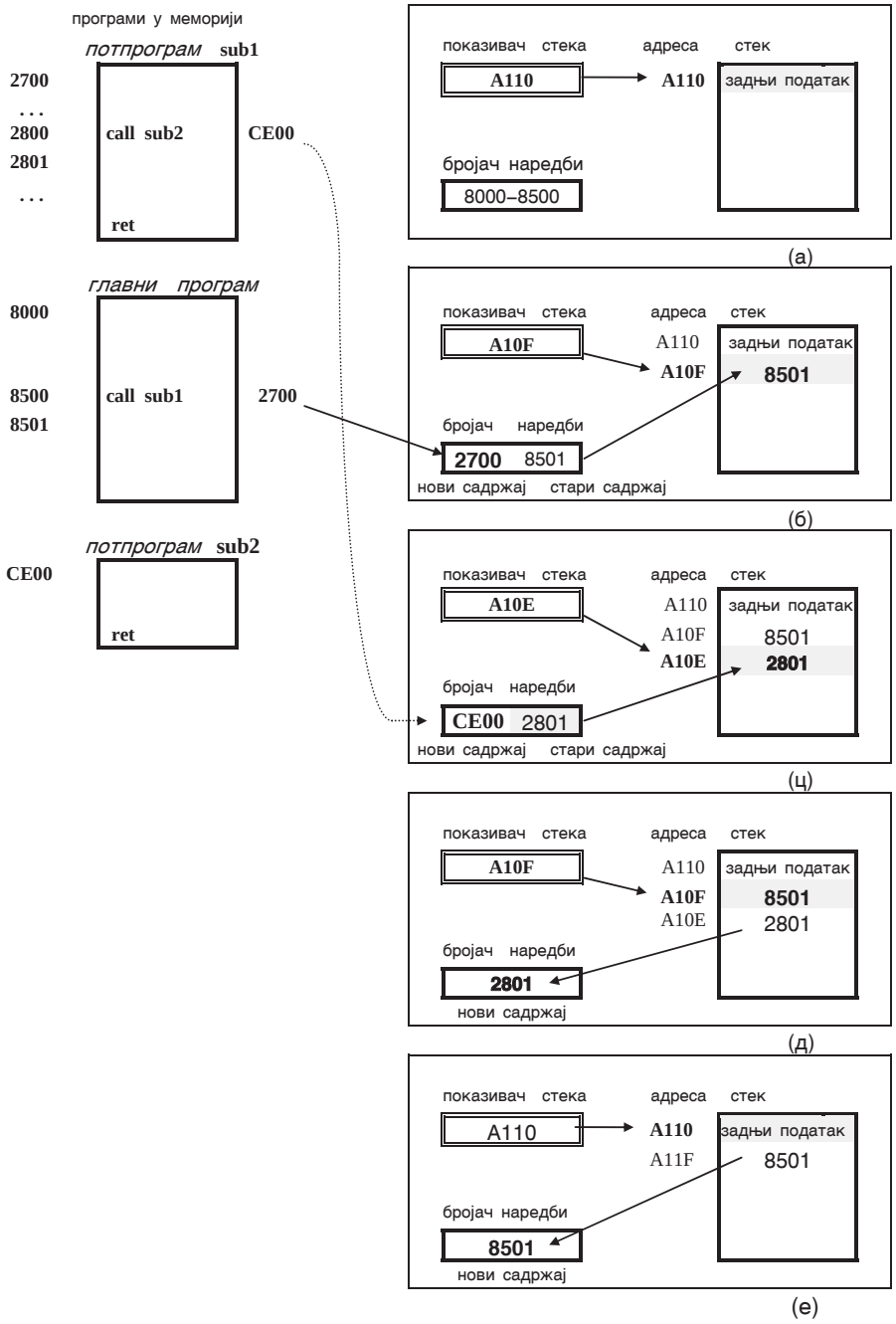
На слици 5.18 је описан програм са два потпрограма и стеком. Почетни садржај показивача стека је $A110_{16}$, слика 5.18 (а), и указује на неки податак који је претходно смештен у стек. Током извршења главног програма, на локацији 8500_{16} налази се инструкција позива потпрограма (**call**). У тренутку када је ова инструкција декодирана, програмски бројач је већ инкрементиран на $\mathbf{8501}_{16}$. Инструкција **call** декрементира показивач стека на $\mathbf{A10F}_{16}$ и смешта садржај бројача наредби на стек, а у бројач наредби се уписује адреса локације ($\mathbf{2700}_{10}$), где се налази прва инструкција потпрограма **sub1,** слика 5.18(б). Дакле, напушта се главни програм и отпочиње извршавање потпрограма.

Током извршавања потпрограма **sub1** извршава се инструкција позива другог потпрограма **call sub2**. Ова инструкција се налази на адреси $\mathbf{2800}_{16}$, и током њеног извршења показивач стека се поставља на $\mathbf{A10E}_{16}$ садржај бројача наредби ($\mathbf{2801}_{16}$) се смешта на стек, а у бројач наредби се уписује $\mathbf{CE00}_{16}$, односно адреса прве инструкције потпрограма **sub2.** Потпрограм 2 се затим извршава у целости, и на крају долазимо до његове последње инструкције **RET.** Ово је инструкција за повратак у програм (односно потпрограм) из ког је **sub2** позван на извршавање. Ова инструкција се извршава тако што се последња вредност која је записана на стек (садржај локације на коју указује показивач стека), преноси у бројач наредби, а садржај показивача стека инкрементира, слика 5.18(д).

По извршењу инструкције за повратак из потпрограма показивач стека садржи број $\mathbf{A10F}_{16}$, а бројач наредби садржи број $\mathbf{2801}_{16}$. Отпочиње прибављање инструкције са локације $\mathbf{2801}_{16}$, односно, наставља се извођење потпрограма **sub1**, тачно на оном месту где се налази инструкција која следи иза инструкције позива потпрограма **call sub2**.

Потпрограм **sub1** се завршава када се прибави и декодира његова инструкција **RET**. Тада се вредност $\mathbf{8501}_{16}$ са адресе на коју указује показивач стека, преноси у бројач наредби, а показивач стека се модификује, слика 5.18(ц). Централни процесор наставља обраду под дејством инструкције са локације $\mathbf{8501}_{16}$ тј. прибавља се и извршава инструкција главног програма која непосредно следи иза инструкције позива потпрограма **call sub1**.

Употреба затворених потпрограма скраћује време писања програма и штеди меморијски простор, али је зато време извођења програма са потпрограмима повећано (због времена извођења инструкција позива потпрограма и повратка у програм).



Слика 5.18. Операције са стеком током позива потпрограма

5.3.4.3.2 ИНСТРУКЦИЈЕ ЗА УПРАВЉАЊЕ СТАЊИМА ПРОЦЕСОРА

Многи модерни рачунари раде у два битно различита режима, мода. Један режим је за корисника, а други за оперативни систем. Режим оперативног система зове се код сваког произвођача друкчије, а у оптицају су: привилеговани режим, монитор, супервизор итд.

Привилеговани режим рада допушта приступ свим деловима меморије и извршавање свих инструкција из скупа могућих. У привилегованом режиму се могу мењати садржаји **PSW,** односно регистра стања, и базних регистара, а то значи да се у овом режиму може у потпуности контролисати рад рачунара.

У току обраде у рачунару програми пролазе кроз више фаза. Већ смо видели да извођење машинске инструкције можемо посматрати кроз најмање две фазе: фазу прибављења, и фазу извршења. Све ове фазе обраде нужно се рефлектују на стање процесора. Код већине рачунара централни процесор се може наћи у неком од следећих стања:

– *непривилегованом / привилегованом стању,*
– *стању спремности (очекивања) / стању обраде, тј. извршавања,*
– *стању чекања (застоја) / стању обраде,*
– *стању дозвољеног прекида / стању забрањеног прекида.*

Ова стања су дата у пару, јер се међусобно искључују.

Постоје и разни режими рада, на пример: режим основног управљања и режим проширеног управљања. Код разних рачунара могући су још неки режими и стања. Сва ова стања приказују се у регистру стања (процесора), па се инструкције које се односе на управљање процесором углавном односе на постављање маски и заставица, уписивање нових садржаја у регистар стања или **PSW (**или **PSR),** читање маски и слично.

Већина инструкција за управљање стањима процесора спада у скуп привилегованих инструкција, и као такве, по правилу, не могу се употребљавати у корисничким програмима.

5.3.4.3.3 ИНСТРУКЦИЈЕ ЗА УПРАВЉАЊЕ ПРЕКИДОМ

Сви рачунари поседују један механизам који им омогућава реаговање на ситуације чије се време настанка и узрок не могу ни на који начин предвидети.

У тренутку када дође до таквих ситуација централни процесор обуставља обраду инструкција текућег програма (прекида се извођење програма) и прелази на извршење неког другог програма.

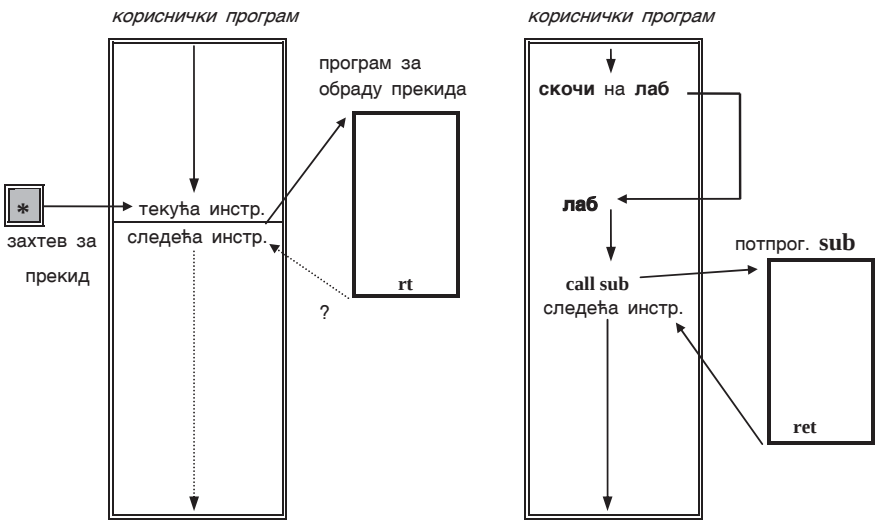
Сигнали који изазивају прекид програма могу настати у самом процесору (ако при извршењу инструкције дође до препуњења акумулатора), у некој улазно–излазној јединици, или, пак, спољашњем окружењу (ван рачунарског система).

Ти сигнали се називају **захтеви за прекид** (**interrupt requests**).

Када се појаве захтеви за прекид, текући програм се прекида, и прелази се на извршење посебног контролно–управљачког програма за обраду прекида. Након извршења програма за обраду прекида треба се вратити у програм који је прекинут и наставити са извршавањем његових инструкција, слика 5.19 (а).

Да би повратак у прекинути програм био могућ, треба, као и код потпрограма, сачувати садржај бројача наредби (адреса следеће инструкције), али додатно треба сачувати још и садржај свих радних регистара у CPU које ће користити програм за обраду прекида. Чување садржаја регистара врши се њиховим копирањем у посебан део меморије, а бројач наредби и регистар стања се обично смештају на стек.

Обрада прекида личи на обраду потпрограма, али за разлику од скока у потпрограм који се врши под контролом инструкције скока у програму, **захтев за прекид може настати било када, на било ком месту у програму, и није под контролом програма.**



(а) *обрада прекида* (б) *наредбе скока и потпрограма*

Слика 5.19. Разлика између прекида програма и гранања и позива потпрограма

Обрада прекида је, због значаја овог механизма за рад рачунарског система, у великој мери подржана хардверски, да би одговор система био што бржи. Обраду прекида најчешће врши један програм оперативног система који се зове анализатор прекида. Сваки извор прекида захтева посебан поступак обраде, па анализатор прекида одабира један од тих програма, управо онај који одговара тој врсти прекида. Механизам прекида је основа за вишепрограмски и вишекорисnички рад рачунара. Захваљујући постојању механизма прекида, могуће је прекинути извршавање једног програма и омогућити извршавање неког другог програма, односно могућ је вишепрограмски режим рада.

С обзиром да постоји велики број могућих извора прекида, и како они нису сви од исте важности за рад рачунарског система, то се неки од њих могу привремено или трајно спречити, односно маскирати. На располагању су инструкције за постављање заставице за прекид, као и за њено брисање. Неки рачунари имају у регистру стања поље за прекидну маску, па код њих постоје и инструкције за постављање маски, њихово читање и тестирање.

Код већине процесора, почетни кораци у обради прекида су хардверски контролисани, док неки имају инструкцију за позивање потпрограма за обраду прекида. То је случај код микропроцесора **Intel 8080** који има инструкцију **RST N (restart).**

Кориснички програм може да тражи услуге од оперативног система, и то обично чини посредством прекида помоћу инструкција: **SVC** (**Supervisor Calls** код **IBM** система), **ER** (**Executive Request** код **Unisys 1100**), **CALL** (код **VAX** рачунара) или **INT** инструкција код микропроцесора фамилије **Intel 8086.**

На крају сваке рутине за обраду прекида налази се инструкција за повратак из прекида, **RT** (**Return from Interrupt**), чији је задатак да у процесор врати стање свих регистара, онако како је оно било у тренутку преласка на програм за обраду прекида. Последњи се обнављају садржаји регистра стања и програмског бројача. Након враћања старог садржаја програмског бројача, прибавља се следећа инструкција прекинутог програма и наставља се са његовим извођењем.

По завршетку обраде прекида није обавезан повратак у прекинути програм, већ се може наставити са извршавањем неког другог програма, што зависи од програма за обраду прекида.

Механизам прекида и обраду прекида треба у потпуности разликовати од гранања у програму (скок на другу инструкцију), као и од позивања и извођења потпрограма. Прекиди нам у основи служе за пребацивање контроле извођења са корисничког програма на контролни програм (најчешће је то неки програм оперативног система) у тренутку који се унапред не може предвидети. Гранање унутар програма, и ван програма

(скок у потпрограм и повратак назад) служи за пренос контроле извођења унутар програма и за извршење неких поступака који се тичу програма и служе, искључиво, самом програму. Контрола извођења безусловно се враћа у програм и остаје у њему и даље. На слици 5.19 шематски је приказана та разлика. У програму и изван програма гранање је тачно познато и унапред одређено, и то:

- *где се грана (и где се враћа ако се ради о потпрограму),*
- *који услови морају бити испуњени да би дошло до гранања,*
- *одакле се грана (тачно одређено место у програму).*

Код прекида програма, за разлику од гранања програмског типа, важи следеће:

- *грана се на различите рутине, зависно од врсте прекида,*
- *до прекида може доћи у било које време (прекид је временски непредвидив),*
- *до прекида може доћи на било ком месту у програму,*
- *у прекиду није могуће унапред предвидети хоће ли повратак уследити натраг у прекинути програм или неки други, јер то зависи од целокупне ситуације у систему.*

Обзиром на значај прекида за рад рачунара о њима ће бити више речи када будемо разматрали вишепрограмски режим рада рачунара, односно при опису језгра оперативног система.

5.4. НАЧИНИ АДРЕСИРАЊА

Брзина обраде и скуп инструкција су битни фактори квалитета рачунара. Но, снага дигиталних рачунара у великој мери зависи и од тога колико спољашњих адреса процесор може адресирати. Те спољашње адресе представљају, пре свега, локације у главној меморији, али и све локације унутар других јединица у рачунару (нпр. улазне и излазне).

Сви подаци се записују у облику бинарних бројева, којима се може дати различито значење (инструкције, специјални знак, бинарни број, BCD број, итд). Свака локација има своју адресу, која се на неки начин мора одредити сваки пут када се жели приступити тој локацији. Поступак одређивања адресе неке локације назива се адресирање. Све могуће адресе које нека процесорска јединица може адресирати чине адресно поље процесора. Процесор може директно адресирати само оне локације које се налазе унутар рачунара, тј. у унутрашњој меморији и у улазно–излазним каналима. При томе се највећи део локација налази у главној меморији, а и већина инструкција, које смо помињали, такође се обраћа локацијама у меморији. Самим тим развијене су и имплементиране бројне стратегије за одређивање

адресе жељене меморијске локације. Најмања адресибилна јединица код већине рачунара је **бајт**. Што се тиче начина адресирања локација у улазно–излазном простору, они у многоме зависе од начина организовања тог простора.

5.4.1. НАЧИНИ АДРЕСИРАЊА МЕМОРИЈСКИХ ЛОКАЦИЈА

Меморија служи за привремено чување програма и података који се управо обрађују. Садржај неке локације може бити инструкција или податак. Адреса локације у којој се налази инструкција чува се у процесору, у једном специјалном регистру који се зове бројач наредби или програмски бројач (**PC**). Обзиром да се наредбе смештају у меморију по реду (у суседне локације), а такође се и узимају и извршавају најчешће по реду, онда се за адресирање инструкција не употребљава регистар већ бројач. Садржај тог регистра се аутоматски повећава – инкрементира, тако да увек показује на следећу локацију. Само изузетно, у програму се дешавају и скокови, па мора постојати и неки начин записа адресе на коју се скаче. Адреса на коју се скаче уписује се у програмски бројач. Но, у крајњој мери, адресирање инструкција своди се на преношење садржаја из бројача наредби у меморијски адресни регистар.

Што се тиче начина адресирања операнада, тј. података над којима се врши обрада, ситуација је знатно сложенија, па се под појмом начина адресирања углавном мисли на адресирање података. Подаци се могу налазити на било којој локацији, (у регистрима у централном процесору, у меморији или U/I јединици), могу се користити било када, по било ком редоследу. Према томе, приступ локацији, ради обраде податка, у начелу је случајан (**random access**).

С друге стране, за рачунар је важно да се инструкције извршавају што брже. Један од услова да се постигне бржи рад јесте да се што мање приступа меморији ради прибављања инструкција и операнада. Другим речима, наредбе треба да су што краће (да би се цела инструкција прибавила у једном захвату), а операнди треба да су у регистрима у самом процесору (јер им се тада знатно брже приступа).

Да би ови захтеви били задовољени, развијене су бројне технике адресирања података, а најчешће су у употреби: имплицитно, непосредно, директно или апсолутно, релативно, индиректно, индексирано, базно и сегментно адресирање.

Пре него што пређемо на опис ових начина адресирања, треба напоменути да инструкције у пољу операционог кода, поред кода операције коју треба извршити, садрже и информацију о томе који начин адресирања се користи. Ова информација се кодира у пољу модификације адресе, тј. у пољу начина адресирања.

5.4.1.1 Имплицитно адресирање

Имплицитно адресирање се назива још и укључено, успутно адресирање (**implied**). У овом начину адресирања сама инструкција, тј. код операције одређује фиксну и непромењиву адресу операнда. На овај начин се, најчешће, адресирају регистри у централном процесору. Користи се обично у инструкцијама инкрементирања, декрементирања, комплементирања, померања, копирања садржаја из једног регистра у други и сл.

Примери за имплицитно адресирање за процесор **MC 68020**:

ASL - аритметичко померање садржаја акумулатора лево

LSR - логичко померање садржаја акумулатора десно.

Процесор **6502** има инструкције:

DEX - декрементирај **X** регистар,

TSX - пренеси показивач стека у **X.**

Инструкције које користе имплицитно адресирање су кратке, имају најчешће само код операције, а операнди су им у регистрима централног прицесора, па је и време извршавања кратко.

Имплицитно адресирање се најчешће користи као успутно адресирање у многим инструкцијама у којима се користе разни други начини адресирања. Овакве су инструкције које објективно користе две разне локације, од којих је једна у CPU, а друга може бити било где (меморија или U/I простор).

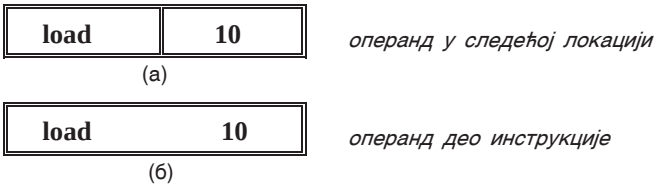
У многим рачунарима у инструкцијама преноса података у и из меморије и аритметичким и логичким инструкцијама, подразумева се употреба неког од регистара, најчешће акумулатора. Такође се у операцијама са стеком подразумева да се адреса врха стека налази у регистру који се зове показивач стека (**stack pointer**), мада се овај регистар нигде експлицитно не помиње.

5.4.1.2 Непосредно адресирање

Операције којима су потребне константе мале бројне вредности често користе непосредно адресирање **(immediate)**. Код овог адресирања поље адресе представља сам операнд, тј. податак се налази у меморији на локацији која непосредно следи иза кода операције.

Код осмобитних микрорачунара, бајт са операндом непосредно следи за бајтом у којем је записан код операције. Како бројач наредби на крају фазе прибављања инструкције бива аутоматски инкрементиран, то он уједно представља адресу податка. Садржај бројача наредби (**PC**) се након узимања податка поново инкрементира. Код већих рачунара (16–битних и

већих), једна меморијска реч поред кода операције садржи и поље операнда, а у овом случају то је сам податак који учествује у обради. На слици 5.20. приказане су ове варијације непосредног адресирања.



Слика 5.20. (а) Непосредно адресирање 8–битних микрорачунара
(б) Непосредно адресирање – **код операције** и **операнд** у једној меморијској речи

Примери ових инструкција за **Motorola** процесоре:

LDA #0429 напуни у акумулатор декадни број 429,
LDA #$0429 напуни у акумулатор хексадецимални број 429,
ASL 3 аритметичко померање акумулатора улево за три места.

Као и код имплицитног адресирања, непосредно адресирање се препознаје помоћу кода операције, тј. начин адресирања је садржан у коду операције.

Инструкције са непосредним адресирањем су брзе за извршење, тј. захтевају мањи број приступа меморији од директног адресирања, али се не могу адресирати промењиве, већ само константе.

5.4.1.3 Директно или апсолутно адресирање

Директно или апсолутно адресирање подразумева да поље операнда у инструкцији садржи физичку, апсолутну адресу локације у којој се налази податак који се обрађује у тој инструкцији. Апсолутне адресе фиксирају (забрављују – **lock**) програм на сасвим одређене меморијске локације, па нису погодне за рад у вишепрограмском режиму рада рачунара.

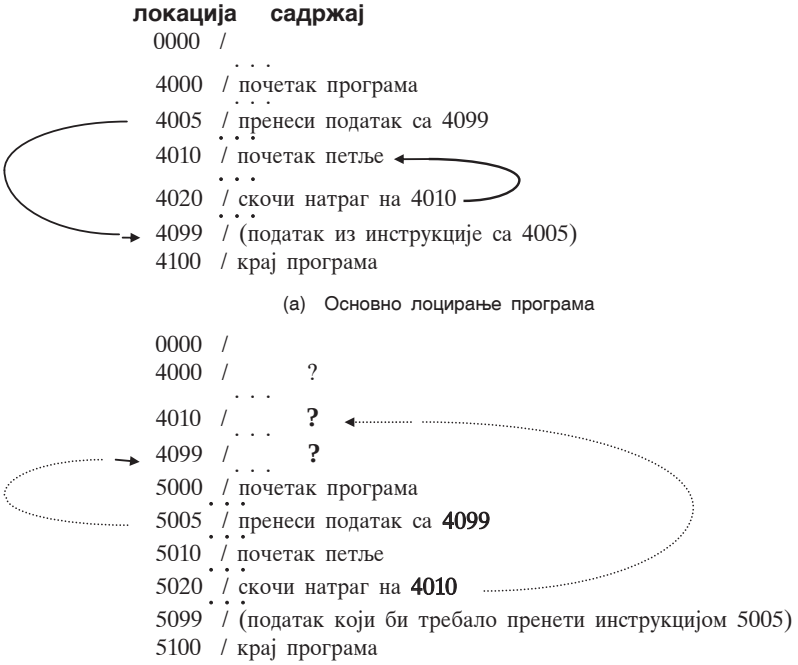
Посматрајмо део програма на слици 5.21 (а), који користи апсолутне адресе. Негде на почетку програма (4005_{16}), налази се инструкција преноса података (**load**) која се односи на податак на адреси 4099_{16}. Програм такође има скок на 4020_{16} који указује да је следећа инструкција на адреси 4010_{16}.

Нека се из неких разлога овај програм помери на локацију 5000_{16}, слика 5.21(б). Инструкција преноса податка која је била на 4005_{16}, сада је на 5005_{16}. Операнд у инструкцији, нажалост, остаје исти, тј. указује на адресу 4099_{16}. Сада видимо да локација 4099_{16} није више унутар програма, и програм више не може да приступи самом податку.

Слична ситуација ће се десити и са наредбом безусловног скока, која је такође померена (**relocated**) на 5020_{16}. Операнд ове инструкције указује на

адресу 4010_{16}, и извршење ове инструкције тера процесор да скочи изван програма и изврши као следећу инструкцију неодређени садржај са локације 4010_{16}. То значи да програми који користе апсолутне адресе не могу бити померени–релоцирани унутар меморије. Приморавање програма да остану на њиховим основним меморијским локацијама могуће је само у једно–корисничким и једнопрограмским системима.

Предност директног адресирања је велика брзина пошто се адреса не мора израчунавати, јер је директно дата иза кода операције. Међутим, за адресирање читавог меморијског простора, код неких рачунара, понекад је потребно користити више меморијских речи за смештање читаве адресе и кода операције, па се онда ради о проширеном директном адресирању. Ово је типично за осмобитне микрорачунаре.



(а) Основно лоцирање програма

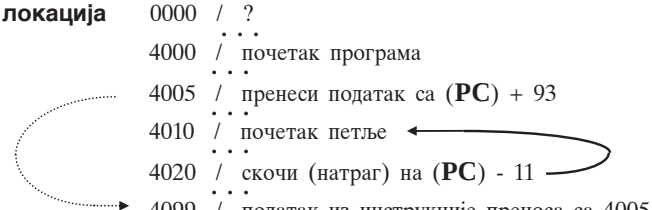
(б) *Лоцирање програма након померања*

Слика 5.21. Неки проблеми са апсолутним адресама

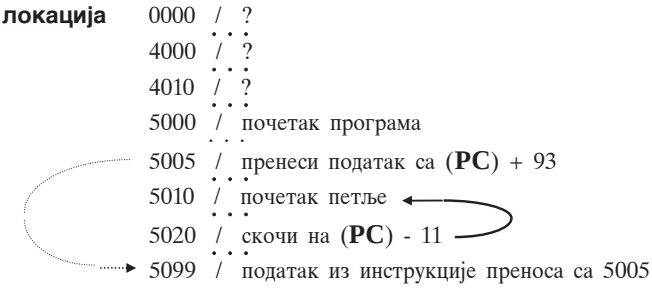
Код 8–битних процесора, први бајт инструкције садржи код операције, други бајт садржи адресни део веће тежинске вредности, а трећи бајт адресни део мање тежинске вредности. Тако се двобајтном инструкцијом може адресирати 256 локација, а тробајтном 64 **KB** меморије. За већи адресни простор треба нам још дужа инструкција, односно више приступа меморији само за прибављање инструкције и адресе податка, а тек потом се прибавља податак. Ово може доста успорити прибављање података.

5.4.1.4 Релативно адресирање

Један од начина да се превазиђу проблеми описани у претходном примеру јесте употреба релативног адресирања (**relative**). Код релативног адресирања, адреса меморијске локације се одређује у односу на текући садржај програмског бројача. Верзија претходног програма са релативним адресирањем дата је на слици 5.22. Видимо да наредба преноса података тражи податак који је смештен 93 локације даље у меморији од места где се она налази. Зашто 93 а не 94? Зато што ће након прибављања инструкције преноса, **PC** бити аутоматски инкрементиран, и указиваће на следећу инструкцију (локација 4006) пре него се релативна адреса израчуна.



(а)



(б)

Слика 5.22. Релативно адресирање

Релативно адресирање такође допушта петљи да користи инструкцију, **скочи натраг за 11** (скочи на PC – 11) и стварно адресира праву инструкцију на коју хоћемо да се вратимо. Физичка адреса ове инструкције је 4020_{16}, али за време фазе извршавања ове инструкције садржај програмског бројача је 4021_{16}, па ће скок унатраг за 11 означити повратак на инструкцију 4010_{16}.

Након померања програма, слика 5.22 (б), инструкција преноса са локације 5005 тачно указује на адресу податка 5006 + 93, тј на 5099. Инструкција скока је на локацији 5020_{16}, а програмски бројач указује на следећу

инструкцију тј. садржи број 5021_{16}. Оба инструкција захтева скок уназад за 11, односно захтева скок на локацију $5010_{16,}$ што је тачна локација.

Релативно адресирање користе најчешће разне инструкције скока (гранања). Главни проблем код релативног адресирања је његов врло ограничен опсег.

Многи осмобитни мини рачунари памте релативне адресе као осмобитне целе бројеве са знаком, што значи да је могуће гранање програма у оба смера (скок унапред и унатраг). Затим, како се ради о двобајтним инструкцијама, у току извођења инструкције скока садржај PC је већ два пута инкрементиран на PC + 2. Сам операнд може бити у опсегу од –128 до +127 код рачунара са другим комплементом, а од +127 до –127 код рачунара са првим комплементом. Адреса скока онда може бити у границама:

(PC) –126 до (PC) +129 *за други комплемент*,

(PC) –125 до (PC) +129 *за први комплемент*.

Наредбе са релативним адресирањем се изводе релативно брзо, јер је операнд одмах иза кода операције (непосредно). Главна мана је у томе што се овако могу дохватити операнди који су релативно близу инструкције, па програмер мора водити рачуна да смести податке тако да буду близу инструкција које их користе. Такође је, за микрорачунаре, сложено извођење дужих скокова од описаних. Обзиром на предности релативног адресирања, велики рачунари допуштају појаву релативних адреса веће дужине од бајта. **VAX** рачунари користе релативне адресе величине бајта, речи и дугачке речи, тј, од 8 до 32 бита.

5.4.1.5 Индексирано адресирање

При обради великог броја података јављају се различити проблеми. На слици 5.23(а) показан је део програма пројектован да сабере 100_{10} бројева смештених на суседним локацијама почев од адресе 1000_{16}. Шта би се догодило ако треба да саберемо 1000_{10} или 1000000_{10} бројева? Да ли је неопходно имати посебну инструкцију за сваки број? Са до сада виђеним начинима адресирања одговор је: **ДА**, но индексно адресирање обезбеђује врло лако решење за овај проблем.

Помоћу индексираног адресирања CPU одређује адресу податка користећи неки од претходних начина, а онда на њу дода садржај неког индекс регистра, и на тај начин се добија ефективна адреса. Садржај индекс регистра мора бити претходно дефинисан, тј. у индекс регистар се пре коришћења мора уписати коректан почетни садржај. Коришћењем овог адресирања, уместо програма, слика 5.23. (а) добија се програм, слика 5.23. (б). Прва израчуната ефективна адреса је 1001, јер је у индекс регистру

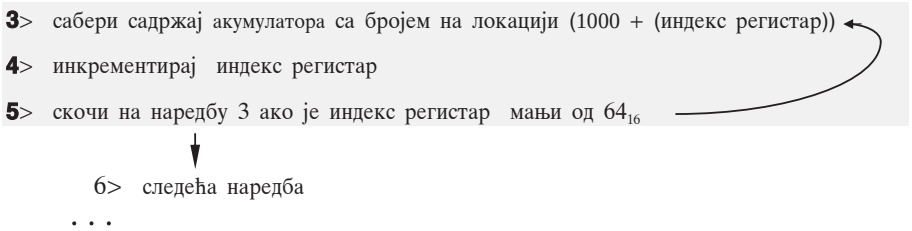
почетни садржај био 1. Након инкрементирања индекс регистра, његов садржај је 2, а ефективна адреса 1002, итд.

```
пренеси у акумулатор број са локације 1000
сабери садржај акумулатора и број са локације 1001
сабери садржај акумулатора и број са локације 1002
сабери садржај акумулатора и број са локације 1003
        . . .
сабери садржај акумулатора и број са локације 1062
сабери садржај акумулатора и број са локације 1063
```

(а) сабирање 100_{10} података смештених у суседним локацијама применом директног адресирања

```
    пренеси у акумулатор број са локације 1000₁₆
    упиши у индекс регистар број 1
1>  сабери садржај акумулатора са бројем на локацији 1000 + (индекс регистар)
    инкрементирај индекс регистар
2>  сабери садржај акумулатора са бројем на локацији 1000 + (индекс регистар)
        . . .
```

(б) сабирање 100_{10} података смештених у суседним локацијама применом индексираног адресирања



Слика 5.23. (ц) Сабирање 100_{10} података употребом петље и индексираног адресирања

Овакав приступ би довео до удвостручавања броја инструкција, али се програм може сада изменити коришћењем петље, слика 5.23 (ц), тако да дужина програма практично не зависи од броја сабирања које треба извршити. Прва наредба **1>** пребацује први податак у акумулатор, а наредба **2>** пребацује у индекс регистар почетну вредност тј. број **1**. Када се обради наредба **3>**, CPU израчунава једну ефективну адресу додајући текући садржај индекс регистра на број 1000_{16}. Како је почетни садржај индекс регистра један, израчуната ефективна адреса је 1001_{16}, и податак са локације 1001_{16} додаје се на садржај акумулатора. Инструкција у четвртом

кораку (4>) каже централном процесору да дода јединицу на садржај индекс регистра.

Корак 5> проверава да ли је текућа вредност индекс регистра мања од 100_{10}, и ако јесте онда CPU враћа контролу на наредбу 3>, тј. она се извршава као следећа инструкција. У кораку 3> израчунава се следећа ефективна адреса и то је 1002, јер је садржај индекс регистра број два.

Програм наставља да сабира податке са садржајем акумулатора, инкрементирањем индекс регистра и понављањем петље све док садржај индекс регистра не достигне 100_{10} (64_{16}). Тада услов у инструкцији скока није задовољен, она се не извршава, већ се прелази на следећу инструкцију.

Овај програм се лако модификује на сабирање 1000 или милион бројева, једноставном изменом услова у наредби 5.

5.4.1.6 Индиректно адресирање

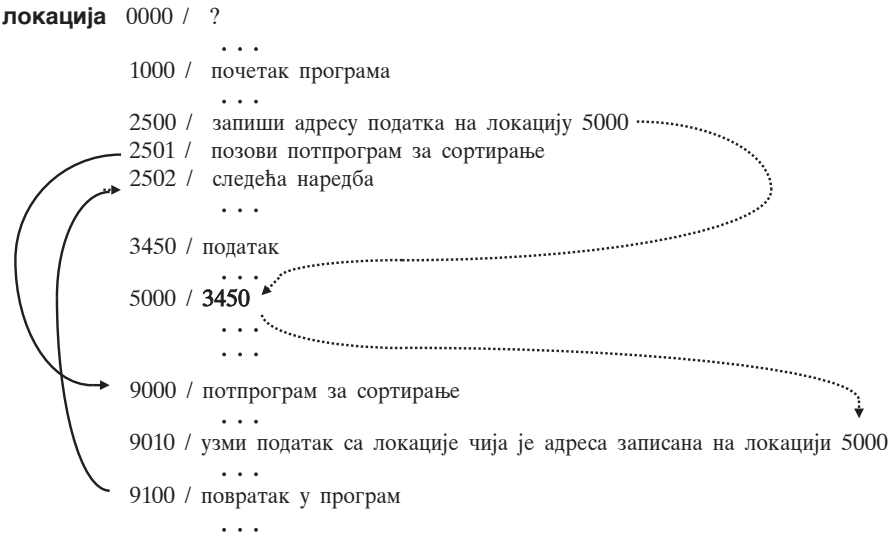
Ситуација може да се искомпликује када потребна адреса није позната у тренутку када се програм пише. На пример, код потпрограма за сортирање низа бројева не би било довољно добро ако би бројеви морали увек да почињу од исте локације у меморији. Потпрограм би био знатно прилагодљивији ако би бројеви могли да буду смештени било где у меморији.

Добро решење овог проблема је коришћење једне меморијске локације да у себи садржи–не податак, већ адресу друге локације у којој се налази податак. Та локација зове се показивачка адреса или показивач (**pointer**), јер њен садржај показује где се налази операнд. Овај начин је врло флексибилан, али је време приступања податку знатно дуже него код директног адресирања, јер је потребно више приступа меморији.

Рутина за сортирање, на пример, може бити пројектована тако да на локацији 5000_{16} тражи адресу првог податка у низу који се сортира. Потребан је један приступ за прибављање операнда 5000_{16}, у другом приступу се чита садржај 3450 са локације 5000_{16}, и то је адреса податка. У трећем приступу се узима податак чија је адреса била на локацији 3450_{16}, као што се види на слици 5.24.

Код индиректног адресирања локација податка није позната, али се зна локација показивача на податак. Значи стварна локација податка није унапред позната, приликом писања програма, већ се записује у време извођења програма. То омогућава да локацију операнда изаберемо зависно од резултата добијених у претходној обради.

Иако је индиректно адресирање врло сложено, оно има и великих предности па се често користи, али је уграђено само у моћнијим рачунарима.



Слика 5.24. Индиректно адресирање које користи једну адресу да нађе другу адресу

5.4.1.7 Адресирање помоћу базних регистара

За рад многих рачунарских система неопходно је померати програме са једног на друго место у меморији, чак и ако ти програми користе апсолутне адресе. Ове процедуре су уграђене у веће оперативне системе, у неки други програм који је резидентан у меморији или у неки други корисников програм. Заједничко за било које решење овог проблема, јесте коришћење базних регистара и адресирања помоћу базних регистара.

Када оперативни систем убацује, пуни, програм у меморију, он истовремено смешта почетну адресу програма у специјални – базни регистар. Програму се тада омогућава да се понаша као да је почео од локације нула, и као да су му све меморијске локације на располагању, **виртуелна меморија**, слика 5.25. а). У стварности, све адресе се израчунавају коришћењем неког од претходно описаних поступака адресирања, онда им се, да би се коначно добила ефективна адреса, додаје садржај базног регистра.

Ако је програм напуњен у меморију почев од локације 40000_{16}, у његов базни регистар се упише 40000_{16}. Када програм затражи податак са

апсолутне адресе 213, податак који ће стварно бити узет смештен је на адреси 40213, као што се види на слици 5.25. б).

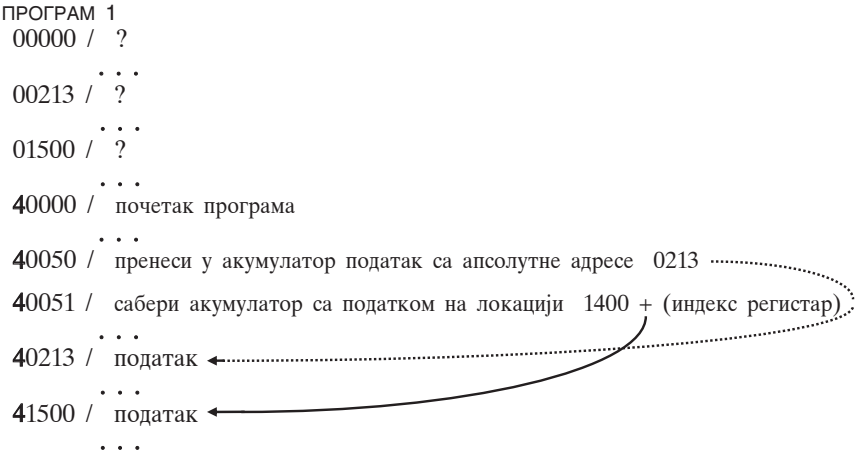
Ако програм израчунава једну индексну адресу као 1400_{16} + (индекс регистар), а у индекс регистру је уписано 100_{16}, коначна адреса којој се приступа, није 1500_{16}., као на слици 5.25. а), већ 41500_{16}, као што је приказано на слици 5.25. б).

локација

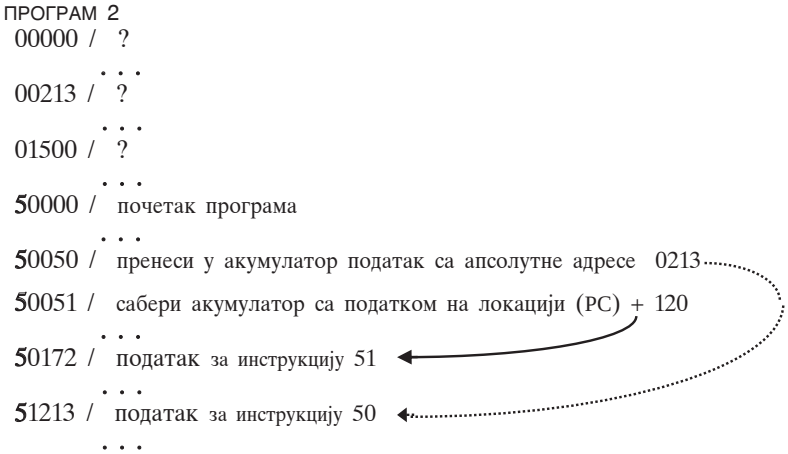
```
ПРОГРАМ 1
 00000 / почетак програма
          . . .
 00050 /  пренеси у акумулатор податак са апсолутне адресе  0213
 00051 /  сабери акумулатор са податком на локацији 1400 + (индекс регистар)
          . . .
 00213 /  податак за инструкцију 50
          . . .
 01500 /  податак за инструкцију 51
          . . .

ПРОГРАМ 2
 00000 / почетак програма
          . . .
 00050 /  пренеси у акумулатор податак са апсолутне адресе  0213
 00051 /  сабери акумулатор са податком на локацији  (PC) + 120
          . . .
 00172 /  податак за инструкцију 51
          . . .
 00213 /  податак за инструкцију 50
          . . .
```

а) како **програми 1** и **2** замишљају свој адресни простор



б) стварни адресни простор програма након пуњења **програма 1** почев од локације 40000



Ц) стварни адресни простор програма након пуњења **програма 2** почев од локације 50000

Слика 5.25. Адресирање помоћу базних регистара, концепт виртуелне меморије

Базно адресирање представља једну од битних претпоставки реализације вишепрограмског рада рачунарског система, као и релокације, тј. померања програма у меморији. Као што се види са слике 5.25. а) оба програма (ПРОГРАМ 1 и ПРОГРАМ 2) замишљају меморију на исти начин, тј., као да је читава меморија само њихова. Стварне адресе инструкција и података одређују се тек након пуњења програма у онај део оперативне меморије који је омеђен базним регистрима који су придружени тим програмима. Тако се податак из инструкције 50 (програма 1) стварно налази на локацији 40213_{16}, а податак из инструкције 50 (програма 2) се стварно налази на локацији 50213_{16}. Из претходног примера следи да се у оперативној меморији рачунара могу истовремено наћи два програма који користе исти логички адресни простор, али се стварне физичке локације које одговарју тим логичким адресама разликују. Наиме, стварна адреса је једнака збиру логичке адресе (коју види програм) и базне адресе (коју је програму доделио оперативни систем).

5.4.1.8 Адресирање помоћу сегмент регистара

Једна варијација базног адресирања налази се у **IBM - PC** компатибилним рачунарима. Наиме, већ су микропроцесори **Intel** 8088/8086 могли да адресирају мегабајтну меморију (2^{20} бајта) јер су имали адресну магистралу

са 20 линија, али њихов адресни операнд има само 16 бита, односно допушта постојање само 65536_{10} (2^{16}) могућих адреса.

Физичка адреса локације дужине 20 бита подељена је на две логичке 16–битне адресе. Само једна логичка адреса се налази у пољу операнда у инструкцији и назива се **померај**, **раздешеност** (**offset**), и то инструкцију чини краћом (штеди се меморијски простор), док се други део логичке адресе налази у једном регистру који се зове сегментни регистар.

Ови микропроцесори имају четири 16–битна сегмента регистра:

- **Code Segment (CS) register**
- **Data Segment (DS) register**
- **Stack Segment (SS) register**
- **Extra Segment (ES) register**

Дакле, оперативна меморија је подељена на четири дела, сегмента. Из самог имена се види чему служе ови регистри. Наиме **CS** регистар садржи почетну адресу сегмента меморије у коме се налази програм, **DS** и **ES** садрже почетне адресе сегмента са подацима, а **SS** садржи почетну адресу сегмента у којем се налази стек меморија. На слици 5.26 приказан је један пример поделе меморије на сегменте.

Физичка адреса податка се израчунава сабирањем помераја који се налази у инструкцији и садржаја сегмент регистра који је продужен за 4 бита са десне стране. Садржај сегмент регистра је $B2C9_{16}$, а њему одговара адреса $B2C90_{16}$. Како поље операнда инструкције садржи померај 123_{16}, ефективна адреса је збиру адресе сегмента и помераја: $B2C90_{16} + 123_{16} = \mathbf{B2DB3}_{16}$.

Адресирање помоћу сегмент регистра може се комбиновати са индексираним, слика 5.26. Адреса податка се тада израчунава на следећи начин. Адреса (1500_{16}) из поља операнда сабира се са садржајем индекс регистра (100_{16}), и добијамо померај (1600_{16}). Померај (1600_{16}) се сабере са адресом из сегмент регистра ($B2CDO_{16}$) и добија се ефективна адреса податка ($B4290_{16}$), који се затим преноси у акумулатор

Адреса следеће инструкције се добија сабирањем садржаја CS регистра са садржајем бројача наредби (PC), што би у овом примеру значило да се на крају фазе прибављања инструкције 50 израчунава стварна адреса инструкције 51, а то је: $10000_{16} + 51_{16} = 10051_{16}$. Адреса врха стека се добија сабирањем садржаја стек сегмент регистра (SS) и показивача стека (SP).

Главна разлика између адресирања сегмент регистрима и базног адресирања је у томе што садржаје сегмент регистра може да мења кориснички програм, док садржаје базних регистра може да мења само оперативни систем.

стварна локација	сегмент регистар	релативна адреса у сегменту		
00000			/ ?	(почетна локација меморије)
. . .				
. . .	**CS**	**PC**		(почетак програмског сегмента)
10000	1000	0000	/ почетак програма	
. . .				
10050	1000	0050	/ пренеси у акумулатор податак са адресе 123	
10051	1000	0051	/ сабери (акумулатор) са податком на локацији	
. . .				1500 + (индекс регистар)
10123	1000	0123	/ инструкција програма	
. . .				
1FFFF	1000	FFFF	/ . . .	(крај програмског сегмента)
. . .				
. . .	**(DS, ES)**	**off set**		
B2C90	B2C9	0000	/	(почетак сегмента података)
. . .				
B2DB3	B2C9	0123	/ податак	(из инструкције 50)
. . .				. . .
B4290	B2C9	1600	/ податак	(из инструкције 51)
. . .				
C2C8F	B2C9	FFFF	/	(крај сегмента података)

Слика 5.26. Адресирање наредби и података помоћу сегмент регистара

5.4.2. АДРЕСИРАЊЕ УЛАЗНО–ИЗЛАЗНОГ ПРОСТОРА

Обављање улазно–излазног преноса података је процес који се у многоме разликује од преноса података у и из меморије. Код рачунара са меморијски шемираним **U/I** простором, за обављање улаза и излаза могу се користити све наредбе и начини адресирања као код приступа меморији. Међутим, велики рачунари, по правилу, имају потпуно одвојене адресне просторе за меморију и за улаз и излаз.

Иако се **U/I** пренос обично обавља уз посредовање оперативног система, готово сви рачунари имају у списку наредби и наредбе за улаз и излаз. Код ових наредби примењује се директно адресирање, при чему је поље адресе обично дужине један или два бајта, односно може се директно адресирати до 256 или до 65536 уређаја. Сем директног адресирања, користи се и адресирање уз помоћ неког од регистара у процесору у који се записује адреса улазног или излазног уређаја.

5.5. ЗАКЉУЧАК

Централни процесор у свом саставу садржи читав низ регистара који се могу користити у унапред одређене сврхе или према потреби, затим аритметичко–логичку јединицу чији је задатак да изврши елементарне обраде над подацима и контролно–управљачку јединицу, која обезбеђује временске и управљачке сигнале неопходне за рад свих делова рачунарског система.

Логичке мреже у саставу централног процесора могу да изврше разне врсте машинских инструкција, које можемо сврстати у три групе: аритметичко–логичке, инструкције за пренос података и инструкције за контролу тока програма. Инструкције за контролу тока програма омогућавају одступање од природног редоследа извршавања програма, а инструкције за обраду прекида допуштају да се прекине извођење једног програма и настави се са извршавањем инструкција другог програма. Механизам прекида је основа вишепрограмског режима рада рачунара.

Инструкције преноса података између меморије и централног процесора у великој мери одређују снагу једног рачунара. Приступање меморијским локацијама врши се на бази садржаја појединих адресних регистара у саставу централног процесора, који се користе или непосредно, или се на бази њих и операнада у инструкцијама, на разне начине израчунавају стварне, физичке адресе података. Сви ови поступци заједно се називају начини адресирања.

Сваки рачунар има свој скуп начина адресирања који морају бити подржани одговарајућим хардвером и одговарајућим решењима у системским програмима. Најчешће коришћени начини адресирања меморијских локација су: директно, непосредно, релативно, базно, сегментно, индексно и индиректно.

Док је за адресирање меморијских локација на располагању неколико разних начина за адресирање, улазно–излазне операције обично спадају у скуп привилегованих инструкција и као такве нису доступне корисничким програмима, већ се одвијају под контролом оперативног система.

5.6. ПИТАЊА

1. У чему се разликују улоге појединих регистара у саставу централног процесора?
2. Које су разлике између корисничких и привилегованих инструкција и њима одговарајућих режима рада?
3. Које врсте програмских наредби постоје?
4. Описати неколико врста унарних операција које се користе у рачунарима.
5. Где се и како користе поједине логичке операције?
6. У чему се разликују инструкције гранања и прескока?
7. Објаснити разлике у начинима приступања меморији и улазно–излазним уређајима.
8. Када је програму потребна мала константа који се начин адресирања користи?

9. Да ли се програми који користе апсолутне адресе могу смештати на разне почетне локације у меморији?
10. Који начин адресирања је погодан за рад са низовима података?
11. Да ли је могуће померање програма у меморији у време извршавања?
12. Колико сегмент регистара имају процесори **Intel** (8086 и надаље), и чему они служе?
13. Ко поставља садржаје базних, а ко садржаје сегмент регистара?

5.7. КЉУЧНЕ РЕЧИ

- акумулатор (**accumulator**)
- апсолутно адресирање (**absolute addressing**)
- аритметички померај (**arithmetic shift**)
- аритметичке инструкције (**arithmetic instructions**)
- базни регистар (**base register**)
- базно адресирање (**base addressing**)
- безусловни скок (**unconditional branch, unconditional jump**)
- бинарне операције (**diadic operations**)
- бројач наредби (**instruction counter**)
- декрементирање (**decrement**)
- директно адресирање (**direct addressing**)
- ефективна адреса (**efective address**)
- гранање (**branch)**
- имплицитно адресирање (**implied addressing**)
- индексно адресирање (**index addressing**)
- инкрементирање (**increment**)
- инструкције преноса података (**data transfer instructions**)
- контролно–управљачке инструкције (**control instructions**)
- логичке инструкције (**logical instructions**)
- логички померај (**logical shift**)
- непосредно адресирање (**immediate addressing**)
- потпрограм (**subroutine**)
- прекид (**interrupt**)
- прескок (**skip**)
- програмски сегмент (**program segment**)
- реч стања програма (**program status word**)
- регистар стања (**status register**)
- регистарске наредбе (**register-to-register instructions**)
- релативно адресирање (**relative addressing**)
- релокација (**relocation**)
- сегмент регистри (**segment registers**)
- сегмент података **(data segment**)
- скок (**jump**)
- стек сегмент (**stack segment**)
- унарне операције (**monadic operations**)
- условни скок (**conditional branch**, **conditional jump**)

6. ПЕРИФЕРИЈСКЕ ЈЕДИНИЦЕ

Начин рада централне процесорске јединице је само један сегмент рада рачунарског система. Због тога треба размотрити и опрему која се користи за уношење и издавање података који се обрађују у рачунару. Ови уређаји се обично називају периферијским јединицама, јер се налазе изван рачунара, тј. централне јединице. Овој категорији уређаја припадају улазни уређаји, као што су: тастатура, миш, светлосна оловка, излазни уређаји попут дисплеја, екрана, принтера и плотера и меморијски уређаји као што су магнетни дискови, траке и оптички дискови, холографске меморије.

Подаци који се користе у периферијским јединицама често немају исти облик као њихов еквивалент у рачунару и зато се морају претварати из једног облика у други. Сем тога ови уређаји су знатно спорији од оперативне меморије или централног процесора, па то додатно компликује пренос података између њих.

6.1. НАЧИНИ ПРЕНОСА УЛАЗНО - ИЗЛАЗНИХ ПОДАТАКА

Пренос података између дигиталних кола, која чине саставни део рачунара и дигиталних логичких кола ван рачунара, обавља улазно–излазни подсистем рачунара. Уређаји за комуникацију између рачунара и периферијских јединица (U/I канали, контролери, периферијски процесори и сл.) остварују четири основне функције:

– *прихватање и прилагођавање (**buffering**),*
– *декодовање адресе, односно избор уређаја,*
– *декодовање команди,*
– *временско вођење и управљање.*

Прихватањем и прилагођавањем врши се синхронизација преноса података између рачунара и спољашњег уређаја. Синхронизација је потребна због неједнаких брзина рада U/I уређаја и рачунара.

Врло често је на један U/I подсистем прикључено више периферијских јединица, па се избор једног од њих обавља на основу адресе. Ову адресу генерише **CPU** и шаље је уређајима. Најчешће сами **U/I** канали врше, поред преноса, и делимичну обраду података, нпр. контролу парности, кодирање итд., па се те функције морају поближе дефинисати.

Све активности у рачунару, и ван њега, захтевају временско вођење (**timing**) и скуп одговарајућих управљачких сигнала. Један део овог посла преузима улазно – излазни подсистем да би се растеретио **CPU**.

Начини преноса података (информација) између рачунара и спољне логике могу се разврстати у три групе:

– *програмирани или програмски* **U/I** *пренос,*

– *прекидни* **U/I** *пренос (***interrupt***),* и

– *директан приступ меморији (***Direct Memory Access, DMA***).*

Програмски или програмирани **U/I** пренос података обавља се под контролом програма који се управо обрађује, тј. под контролом **CPU**.

Код прекидног **U/I** преноса података, спољашњи уређај (најчешће преко неког **U/I** канала или контролера) захтева од рачунара (тј. од **CPU**) да прекине извођење текућег програма, и посвети пажњу преносу података.

У оба претходна случаја пренос података се обавља уз учешће централног процесора, што у знатној мери успорава његов рад. Могуће је реализовати и пренос података без учешћа **CPU** уз помоћ, такозваног **директног приступа меморији**, када меморијске адресе и управљачке сигнале не генерише процесор већ један други специјализовани хардвер (чип) који се зове **DMA** контролер.

6.1.1. ПРОГРАМИРАНИ U/I ПРЕНОС

Већ смо напоменули да постоје две врсте рачунара. Код једних (обично великих рачунара), меморија и **U/I** међусклопови не користе заједничку магистралу података и адреса. Самим тим постоје одвојене инструкције за приступ меморији и U/I међусклоповима.

Постоји више разних врста инструкција **U/I** преноса података, на пример:

– *пренос података између регистара у процесору и регистара у* **U/I** *међусклоповима,*

– *пренос података између меморије и регистара у* **U/I** *међусклоповима,*

– *инструкције за испитивање стања периферијских уређаја и*

– *контролно–управљачке инструкције.*

Ове последње две врсте инструкција су неопходне јер постоји посебан хардвер за управљање **U/I** преносом, тј. постоје разне врсте **U/I** међусклопова (контролери, канали, улазно–излазни процесори и слично).

Друга врста, микро и мини рачунари, најчешће имају такву организацију да и меморија и **U/I** јединице користе исте магистрале података и адреса. Због тога се на магистрали адреса не разликују адресе меморијских локација од адреса **U/I** међусклопова. Већ смо поменули да тада постоје два начина организовања **U/I** адресног простора.

– *меморијски пресликани,*

– *издвојени, изоловани* ***U/I*** *простор.*

Код рачунара са меморијски пресликаним **U/I** простором, све инструкције за пренос података у и из меморије, као и сви поменути начини адресирања стоје нам на располагању. Код рачунара са издвојеним **U/I** простором, поступак преноса је исти као код великих рачунара, јер постоје посебне улазно – излазне инструкције, које у свом извршавању генеришу посебне улазно–излазне управљачке сигнале, који су доступни само **U/I** међусклоповима, а не и меморији.

Улаз и излаз података увек прати низ проблема, јер он никада не представља просто слање или пријем бајта података са уређаја. Слање податка на диск захтева тражење тачно одређеног цилиндра, налажење чистог (слободног) простора, лоцирање сектора, кодирање, додавање бита парности, па се тек на крају врши записивање. Читање податка са тастатуре захтева да се најпре изврши конверзија сигнала добијеног притиском на дирку (тастер) у **ASCII** или **EBCDIC** кодове.

Разлике у брзини преноса података улазно–излазних уређаја и процесора представљају додатни проблем. Због свега овога, изузев код неких микрорачунара, операције улаза и излаза се увек обављају уз помоћ оперативног система.

Програмирани **U/I** пренос подразумева извршавање U/I инструкције када током обраде програма дође на ред. Централни процесор тада шаље **U/I** управљачке сигнале ка некој периферији, и чека да та јединица заврши свој посао. Тек након што је улазно–излазна операција у целости обављена, **CPU** може да настави са обрадом следеће инструкције у програму.

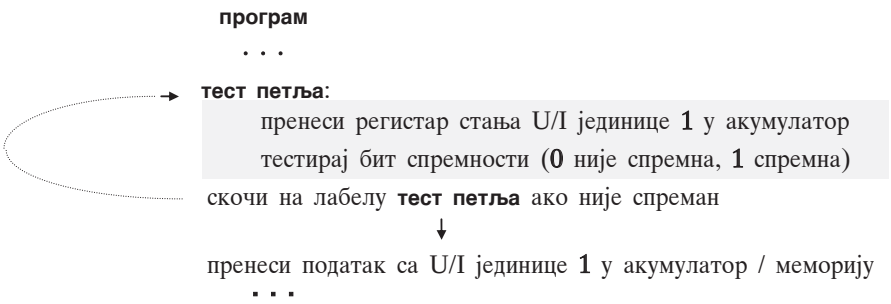
Ово је такозвано синхроно процесирање, а користи га још увек већина микрорачунара. Главна мана програмираног **U/I** преноса података јесте неефикасност система јер процесор остаје докон (**idle**) док траје обављање ове активности.

Постоје две врсте програмираног **U/I** преноса: безусловни и условни, На сликама 6.1. и 6.2. приказани су примери реализације програмираног **U/I** преноса за микропроцесор **Intel 8080**. Безусловни пренос је једноставнији и

бржи, али је непоуздан. Он се користи само онда када је **U/I** јединица увек спремна да обави пренос, и када унапред знамо када треба обавири неку улазну или излазну операцију.

```
      програм
      . . .
IN 1      пренос података са U/I јединице 1 у акумулатор (улаз)
      . . .
OUT 6     пренос података из акумулатора у U/I јединицу 6 (излаз)
      . . .
```

Слика 6.1. Безусловни програмирани U/I пренос



Слика 6.2. Условни програмирани U/I пренос

Условни пренос је сигурнији јер се најпре тестира стање **U/I** јединице, а пренос се обавља тек када је **U/I** јединица спремна. Но, чекање да уређај буде спреман додатно успорава рад процесора, тј. рачунара.

Програмирани **U/I** пренос је могуће користити само у случајевима када унапред знамо када тачно треба опслужити неку улазну или излазну јединицу. Нажалост, у пракси је то неизводљиво, па овакав пренос увек има за последицу велико расипање рачунарског времена због чекања на спремност уређаја и чекања да уређај обави пренос. То је очигледно, јер је већина периферијских уређаја знатно спорија од процесора. Многе периферије нису чак само електронске, већ и електромеханичке, па је њихова брзина мања од брзине процесора и милион пута.

У случају када постоји више спољашњих уређаја који су повезани преко једног U/I међусклопа, потребно је додатно време за периодично прозивање тих уређаја да би се утврдило да ли им треба опслуживање. Тај поступак назива се **прозивка** (**polling**). Да би се избегло непотребно губљење времена на испитивање стања уређаја и његове спремности за пренос користи се техника прекида и техника директног приступа меморији.

6.1.2. ДИРЕКТАН ПРИСТУП МЕМОРИЈИ

Овај метод **U/I** преноса ослобађа процесор терета контроле **U/I** преноса података. Помоћу директног приступа остварује се брз пренос између примарне меморије и спољног света и периферијских јединица без контроле од стране централног процесора, и без утицаја на садржаје регистара у **CPU**. Обзиром да се тај пренос врши без програмског управљања, границе брзине преноса зависе од карактеристика дигиталних кола која контролишу пренос. Координацију свих активности у току преноса преузима посебан хардвер који се зове **DMA** контролер, слика 6.3. У овом поступку ток података, успостављен између меморије и спољашњег логичког склопа, заобилази процесор, односно **DMA** има издвојен канал за пренос.

Неки рачунари, нарочито микрорачунари, немају могућност издвајања посебног **DMA** канала, па се у **DMA** преносу користи иста спољна магистрала коју користи и процесор (испрекидана линија на слици 6.3), и тада **DMA** мора послати сигнал процесору да му препусти коришћење магистрале. Да би се остварио директан приступ меморији, **DMA** контролер мора обавити следеће функције:

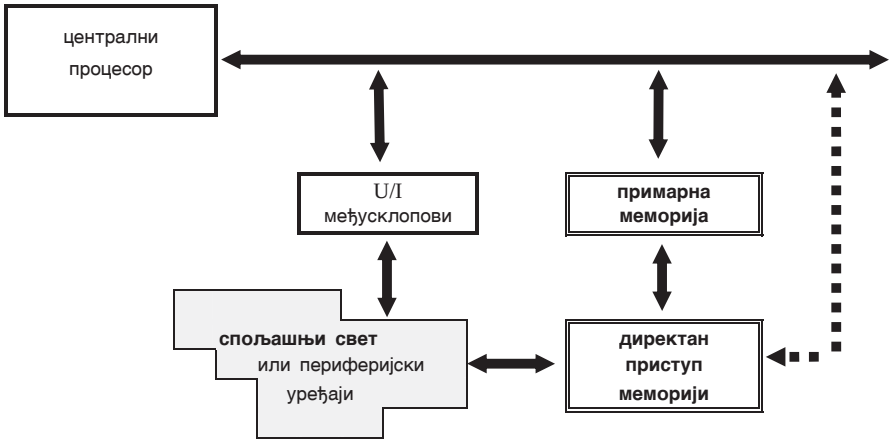
– *управљати адресном магистралом у време* ***DMA*** *преноса,*

– *управљати магистралом података,*

– *генерисати адресе локација у меморији,*

– *бројати пренете податке,*

– *изабрати начин управљања, смер преноса итд.*



Слика 6.3. Шематски приказ директног приступа меморији (**DMA**)

DMA се најчешће користи за пренос велике количине података (блокови података). За покретање неке улазно–излазне операције коришћењем **DMA**, неопходно је извршити један програм који обавља следеће функције:

- *пуњење почетне адресе блока у меморији,*
- *пуњење бројача речи,*
- *пуњење инструкције која одређује функцију која ће бити извршена (**read**, **write**),*
- *слање инструкције за отпочињање преноса (нека **GO** команда).*

То, уствари, значи да у **DMA** контролеру постоје меморијски адресни регистар и регистар у који се уписује дужина блока података. Такође, постоји и бројач који броји пренесене податке, као и управљачка јединица која декодира инструкцију преноса и генерише одговарајуће управљачке сигнале. **DMA** се зауставља када је достигнута жељена дужина блока.

6.1.3. ПРЕКИДНИ U/I ПРЕНОС И ПЕРИФЕРИЈСКИ ПРОЦЕСОРИ

Протекло је доста времена од првог рачунара до момента када је коначно постигнуто да два програма могу да буду истовремено у меморији, тако да један програм може да користи **CPU**, док други програм чека да се заврши његова улазно–излазна операција. Ово је основна идеја која се крије иза појма мултипрограмирања.

Мултипрограмски системи имају у меморији, у исто време, инструкције од неколико програма. Док неки програм чека да се заврши његова **U/I** операција, процесор није докон, већ је слободан да изврши инструкције из другог програма. Резултат оваквог рада је већа ефикасност у коришћењу рачунарских ресурса у смислу преклапања обраде података у централном процесору и обављања **U/I** активности.

Улазно–излазни уређаји и контролери пројектовани су тако да имају дигитална кола и логику рада која допушта полуаутономно функционисање. Овакви уређаји имају потребу да врло мало комуницирају са процесором, најчешће само да га обавесте да желе да изврше пренос, или да су завршили задати део посла. Ова комуникација се обавља путем сигнала прекида, на линији прекида **INT**.

Када добије захтев за прекид **INT**, процесор суспендује (прекида) текући програм, и отпочиње извршавање специјалног програма за обраду прекида који се зове **анализатор прекида**. Најчешће се најпре до краја изврши текућа инструкција (која се управо извршавала у тренутку када је дошао сигнал за прекид), а затим се прекида извођење програма.

Главни задатак анализатора прекида је да утврди који је уређај тражио прекид, да преда контролу одговарајућем програму који опслужује тај уређај, и касније да врати контролу главном програму који је прекинут.

Захтеви за **U/I** прекид могу настати било када, тј. асинхроно и тада се **U/I** уређајима омогућава директан приступ меморији, а ови уређаји настављају да раде полуаутономно.

Сада можемо да дамо једну поједностављену скицу догађаја који се дешавају од тренутка када нека периферијска јединица постави захтев за прекид:

1. *периферијска јединица поставља захтев за прекид,*
2. *централни процесор одмах (или по завршетку текуће инструкције) прекида извршавање текућег програма,*
3. *спречавају се нови сигнали за прекид (маскирање прекида),*
4. *обавештава се периферијска јединица да је њен сигнал за прекид препознат, и она као одговор мора да уклони свој захтев за прекид,*
5. *извршава се активност која је потребна да се опслужи прекид са те периферијске јединице, тј. извршава се програм којим се обавља пренос података између те периферијске јединице и меморије рачунара,*
6. *поново се допушта појава сигнала за прекид,*
7. *наставља се извршавање прекинутог програма.*

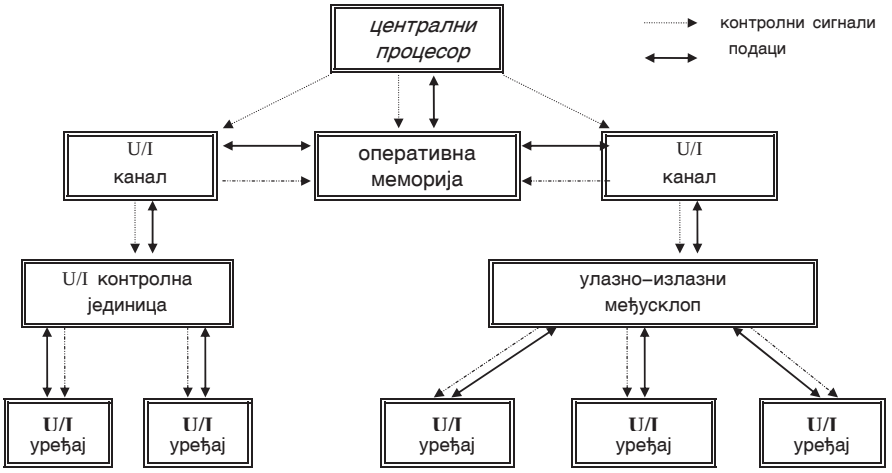
Постоји читав низ проблема који се морају разрешити да би прекидни систем функционисао ефикасно: како CPU може да препозна периферијски уређај који је тражио прекид? Обзиром да различити уређаји захтевају различите начине опслуживања, тј. различите програме, поставља се питање како ће CPU да сазна почетну адресу програма који треба да буде извршен? Да ли за време обраде једног прекида може доћи други сигнал за прекид, и шта у том случају да се ради? Шта да се ради у случају када два или више прекида дођу истовремено?

Велики рачунарски системи често користе специјалне рачунаре, који су предодређени да врше само улазно–излазне операције, али ослобађају **CPU** од спорих **бајт по бајт** улазно излазних активности. Ови се уређаји зову канали или периферијски процесори, и логично је што су смештени између меморије и **U/I** уређаја, слика 6.4. Канали имају вишеструку улогу у рачунарском систему:

– *прилагођења брзих оперативних меморија и спорих периферија,*
– *промене кодова и дужине (број битова) података,*
– *пренос података у већим блоковима, да би што ређе тражили приступ меморији.*

Канали, тј. периферијски процесори, имају придружену малу приватну меморију у коју записују: програм који извршавају, адресе у оперативној меморији којима приступају (почетне и крајње), информације о **U/I** јединици

итд.. Канал ради полуаутономно, и процесору се обраћа само кад изврши свој програм да би од њега добио нови задатак, или неопходан минимум информација о наставку текућих активности.



Слика 6.4. Пренос података помоћу канала између оперативне меморије и улазно–излазних уређаја

Неки канали могу опслуживати, у једном кратком интервалу времена, само једну брзу периферијску јединицу (диск, трака) и зову се селекторски канали, или више спорих уређаја (терминал, принтер) и тада су то **бајт мултиплексерски канали**. Код бајт мултиплексерских канала, њихов укупни капацитет преноса података, подељен је на више потканала. Сваки потканал има довољно снаге да подржи терминале, принтере, читаче кар–тица итд. Пројектовани су и канали који обједињују добре особине прет–ходна два типа. То су **блок мултиплексорски канали**, који могу да над–гледају неколико брзих јединица преносећи по један блок података једној па другој јединици и тако редом.

Треба напоменути да немају сви рачунари канале или периферијске процесоре. **VAX 11/780** рачунари имају два специјална U/I адаптера и две специјалне магистрале. **UNIBUS** адаптер има исту улогу као бајт мулти–плексерски канал, односно омогућава да неколико спорих уређаја приступи рачунару. Брзе периферије, као што су дискови и траке, су под контролом **MASSBUS** адаптера.

Микрорачунари имају за сваку периферијску јединицу посебан међусклоп, тј. контролер који надзире пренос података. Тако у састав микрорачунара улазе графички контролер за контролу видео дисплеја, диск контролер, контролер измењивог диска, контролер оптичког диска, серијски и

паралелни интерфејс итд. Серијски и паралелни интерфејс нису намењени одређеном уређају већ се на њих могу повезивати разни уређаји. Тако се на серијски интерфејс може повезати Fax-modem, миш или серијски штампач. Прикључивање на било који контролер захтева поштовање одређених процедура у раду, тј. поштовање одређених стандарда (протокола).

6.1.4. СТАНДАРДИ ЗА ПЕРИФЕРИЈСКЕ МЕЂУСКЛОПОВЕ

Приликом развоја уређаја који садрже управљачки део реализован помоћу процесора или, најчешће, микрорачунара потребно је посветити посебну пажњу структури спољне магистрале, јер она представља заједничку компоненту за међусобно повезивање многих компоненти и модула. Стандард магистрале је нужан фактор који се не сме занемарити. Он је један од главних услова за додавање нових склопова и модула у рачунарски систем. Стандард поједностављује процес конструкције уређаја, омогућује конструктору примену широког спектра компоненти и модула једноставним укључивањем (**plug–in**). На пример, низ произвођача нуди плочицу са меморијским модулом од 8MB. Уколико уређај има стандардну магистралу проблем проширења меморије се решава једноставним додавањем још једне плочице. У супротном су потребне модификације или на плочици или на магистрали.

Начин саобраћања на магистрали је по систему: водећи / пратећи (**master** / **slave**). На пример, микропроцесор са својим помоћним склоповима на матичној плочи је управљач (**bus master**), односно водећи, док су пратећи (**slave**) модули меморија, U/I међусклопови, управљачки склопови и слично. У рачунарским системима са више процесора потребан је "**магистрални арбитар**" (**bus arbiter**) – модул за додељивање права располагања магистралом. Погледајмо сада неке од познатијих стандарда за спољне магистрале:

Магистрала S-100 (**Altair/Imsai standard**) појавила се 1976. године и постала стандард захваљујући експанзији тржишта микропроцесора, и то микропроцесора намењених широком кругу поклоника технике. Магистрала S-100 се састоји од сто линија, има три нивоа напајања: +8V са регулатором напона за +5V; +16V са регулатором на +12V и –16V са регулатором на –12V. Ова магистрала подржава 8–мо битни и 16–то битни пренос података, и 14–то битно адресирање. Има две линије за прекид и допушта употребу векторских прекида. Један уређај (најчешће процесор) може бити управљач, а још 16 других уређаја могу привремено бити управљачи магистрале (**bus master**).

Предности магистрале S-100 су:

- *врло популарна и распрострањена, кориснику стоји на располагању мноштво модула и периферијских јединица,*
- *мноштво картица за експериментисање и стандардних међусклопова изведено је у стандарду S-100,*
- *мале димензије картице омогућују лаку реализацију сложених склопова,*

Недостаци стандардне магистрале S-100:

- *постоји одређени број недефинисаних линија,*
- *извори напона +8V, ±16V нису стандардни,*
- *нема довољно линија за масу (**GND**) и напајање.*

Магистрала S-100 је првенствено намењена рачунарским системима који се базирају на микропроцесорима Intel 8080, 8080A, 8085 и Z-80, док су за коришћење микропроцесора M6800 потребне мале измене на магистрали.

Фирма Intel је развила магистралу **Multibus**, која је захваљујући утицају Intel–а на тржишту, постала једна од најчешће примењиваних. Предности магистрале Multibus:

- *груписане су адресна, управљачка магистрала и магистрала података, што омогућује лакше испитивање и отклањање грешки,*
- *садржи додатне линије за масу и напајање (за уређаје којима су потребне веће струје),*
- *постоји велики број периферних картица компатибилних са Multibus-ом.*

Магистрала Multibus има 86 основних линија (16 линија за податке, 20 линија за адресу, 25 за управљање, 20 за напајање (+5V, ±12 V, GND), а 5 је резервисано за будућу употребу) и још 60 додатних, које сам корисник може дефинисати према потреби. Недостаци магистрале Multibus су:

- *првенствено је намењена породици Intel 8080, 8085 и 80x86,*
- *неким модулима је потребан напон –10V,*
- *већа димензија картице.*

Магистрала **Pro–log** микрорачунара има 56 линија и стандардизоване кар–тице малих димензија. Предности стандарда Pro–log магистрале су:

- *картице су једне од најмањих,*
- *напајање ±5V и +12V,*
- *адресна магистрала и магистрала података следе једна за другом,*
- *свака картица има двоструке прикључке за масу и напајање са сваке стране картице.*

Недостаци стандарда Pro–log магистрале су:

- *мање секундардних испоручилаца модула,*
- *понекад потребне додатне картице за сложеније уређаје.*

Магистрала LSI–11 DEC (Digital Equipment Corporation) једноставна је и употребљава се за саобраћање између микропроцесора, меморије и периферијских слопова. Стандард LSI–11 омогућује употребу и укључивање у рачунарски систем ½ картице (36 линија), нормалне картице (72 линије), средње картице (108 линија) и 1½ картице (144 линије). Магистрала LSI–11 има мултиплексиране адресне линије са линијама података. Стандард је скоро искључиво намењен микрорачунарима DEC LSI–11 и то је један од највећих његових недостатака. Нормална картица има 72 контакта и, уствари, иста је као **Unibus** магистрала која садржи: 16 линија за податке, 18 линија за адресе, 22 линије за управљање и 16 линија за напајање (+5, GND). Ова магистрала се користила као унутрашња за рачунаре **PDP**–11 и као спољна за рачунаре **VAX**.

Развој фамилије микрорачунара базираних на микропроцесорима фирме **Intel** праћен је развојем одговарајућих типова магистрала. **IBM** је за **XT** рачунаре (са процесором 8088) дефинисао **ISA** магистралу засновану на индустријском стандарду (**Industryl Standard Arhitecture**) са картицама које имају 62 контакта:

- *осам линија за пренос података,*
- *двадесет линија за адресе,*
- *шест линија за контролу прекида,*
- *шест линија за DMA,*
- *осам линиј за напајање (маса, ±5V и +12V),*
- *седам управљачких линија за меморију и улаз/излаз,*
- *седам линија за управљање (такт, освежавање, ресет и сл.).*

Са појавом AT рачунара (процесор 286 и јачи) ова магистрала је била недовољно брза па је једно од решења било у примени EISA проширења постојеће магистрале којој се додаје још једна прикључница са 36 линија: осам за податке, осам за адресу, две за напајање и осамнаест за управљање (пет за меморију и улаз/излаз, пет за прекиде и осам за DMA).

Са појавом 32–битних и 64–битних процесора јављају се нове магистрале а данас се користи најчешће **PCI** (**Peripheral Component Interconnect**). За 32–битне рачунаре користе се прикључнице са 124 контакта, док се за 64–битне користе прикључнице са 188 контаката, што је омогућило пренос од 528 MB/sec при такту од 66 MHz. Данашње магистрале имају такт преко 300 MHz.

Прикључење програмабилних и интелигентних инструмената на рачунар или микрорачунар тражи такође примену одређеног стандарда. У ту сврху развијена је стандардна магистрала IEEE 488 која се састоји од 16 сигналних линија:

- *пет управљачких линија (IFC, ATN, EOI, REN, SRQ).*
- *три линије за руковање преносном (**handshake**) (DAV, NFRD, NDAC),*
- *осам линија података DI01–DI08.*

Сваки инструмент прикључен на магистралу мора имати најмање једну од ових функција:

- *слушалац – инструмент способан за примање података,*
- *говорник – инструмент способан за слање података,*
- *управљач – који омогућује и онемогућује говорнике и слушаоце.*

На пример, микропроцесор може да садржи све три функције; функцију слушаоца може имати штампач или дигитално–програмибилни извор напона, а функцију говорника дигитални волтметар или бројило. Стандард IEEE прописује и протокол саобраћања између јединица прикључених на магистрале.

RS–232 C је серијски међусклоп (interface) погодан за пренос података на велика растојања. Овај међусклоп захтева само два проводника што има следеће предности:

- *цена кабла и број линијских појачавача је знатно мања него код веза са више жица,*
- *овај интерфејс омогућава да се у преносу података користи комерцијална комуникациона опрема (као што је, на пример већ постојећа телефонска линија).*

Код овог преноса податак се преноси бит по бит. По конвенцији, најпре се преноси LSB. То значи да је за пренос једног бајта потребно осам пута више времена него при паралелном преносу, али за највећи број примена брзине преноса које се могу остварити помоћу овог међусклопа су сасвим довољне. Веза између уређаја који користе серијски пренос, било асинхрони било синхрони, најчешће се остварује по препоруци (стандарду) V.24, а посебно када се користе за повезивање рачунара са модемом. У практичној употреби су брзине преноса: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 33600 и 56000 **bps** (бита у секунди, **bit per second**). Почев од 1996. године сви рачунари поред RS232, користе *универзалну серијску магистралу* **USB** (**Universal serial Bus**) која оногућава два нивоа брзине преноса: од 12 мегабита/секунди (**Mb/s**) и један подканал са 1,5 **Mb/s**. Данас готово све врсте уређаја подржавају ову магистралу.

У предходном тексту било је претпостављено да постоји само један "господар" на магистрали и да је то централни процесор. У реалним системима, улазно/излазни склопови често имају улогу "господара", Читају и уписију у меморију и шаљу сигнале прекида. Копроцесори такође могу имати потребу да буду "господари" на магистрали. Намеће се питање: "Шта се дешава ако два или више уређаја захтевају да буду "господари" у исто време?". Одговор је да је потребан механизам **арбитрације магистрале** (**bus arbitration**). Механизми арбитрације могу бити централизовани или децентрализовани. У случају централизоване арбитрације постоји један посебан уређај – арбитар магистрале који одређује следећег "господара" на магистрали. Многи микропроцесори имају арбитра који је уграђен у чип централног процесора, мада је у минирачунарским системима он понекад одвојен уређај. Код децентрализоване арбитраже питање управљања магистралом решава се линијама са различитим нивоима приоритета, или као код Multibus система са лепезастим уланчавањем при чему уређај нејвећег приоритета блокира захтеве уређаја са нижим приоритетом.

Код већине система централни процесор се такође мора такмичити за контролу над магистралом, али он има најнижи приоритет и добија магистралу само када ниједан други уређај то не захтева. Идеја је да централни процесор увек може да чека, а да улазно/излазни уређаји често морају брзо да добију магистралу на располагање или ће у супротном изгубити улазне податке. Дискови који ротирају великим брзинама такође не могу да чекају.

6.2. УРЕЂАЈИ ЗА УНОС И ИЗДАВАЊЕ ПОДАТАКА

Обрада података се одвија, најопштије посматрано, у четири фазе: унос података, прихват и меморисање података, обрада података у ужем смислу и издавање резултата обраде. Свака фаза обраде подразумева употребу

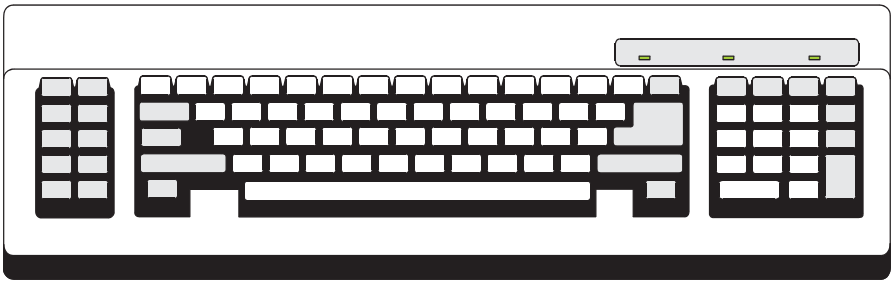
одређених специјализованих уређаја посебно пројектованих да обављају само одређене операције.

Без обзира на претходну поделу, поред улазних и излазних уређаја постоји и велики број уређаја који обављају обе ове функције. У групу уређаја за унос података и издавање података и резултата обраде спадају разни периферијски уређаји чији су основни задаци унос података у рачунарски систем и издавање резултата. По својој природи ови уређаји могу бити типични улазни (тастатура, оптичка оловка, миш, итд.), типични излазни уређаји (видео дисплеји, штампачи, плотери, итд.), и уређаји који могу да обављају и једну и другу функцију.

6.2.1. ТАСТАТУРЕ

На почетку ере рачунара главни облик уноса података биле су бушене картице. Но, данас се највећи проценат рачунарских програма и података уноси преко тастатура које су саставни део терминала или микрорачунара (слика 6.5.).

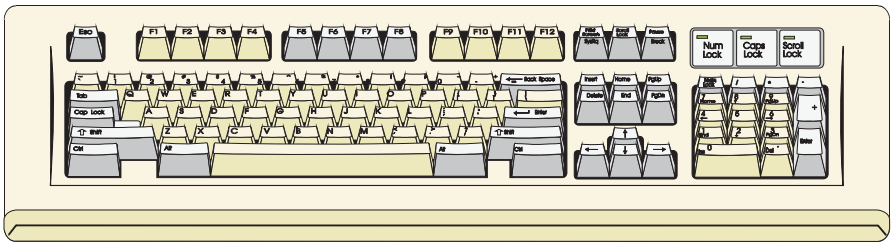
Притиском на било који тастер генерише се сигнал, тј. промена напона, која се детектује помоћу дигиталног кола који се зове енкодер.



Слика 6.5. Типична тастатура

Задатак енкодера је да конвертује сигнал који долази са неког од тастера у форму која је прихватљива за рачунар, тј. у низ бинарних цифара, у **ASCII** или **EBCDIC** коду или један специјални број придружен том тастеру. Притиском на тастер **X** на излазу енкодера се јавља број 10100111_2 у **EBCDIC** коду или 01111000_2 у **ASCII** коду.

Тастатуре (слика 6.5. и 6.6.) поред уобичајених знакова који представљају бројеве (0,1,...,9), слова азбуке (а, б, ... ,ш, А, Б, ... ,Ш), симбола операција (+, –, >, /) и неких посебних знакова ($, !, ., ; итд), имају и специјалне тастере који мењају значење попут других тастера. Најчешће коришћени су: **Shift**, **Control** (**Ctrl**) и **Alternate** (**Alt**).



Слика 6.6. Тастатура **IBM-PC** рачунара

Притиском на дугме **Shift**, на пример, истовремено са притиском на дугме **s** добија се код који одговара за велико слово **S**. Ако истовремено са притиском на тастер са словом **x** притиснемо и тастер **Ctrl**, енкодер шаље ка рачунару специјални знак који се зове **Control-x**. Слично се догађа и при истовременом притиску на тастере **Alt** и **x** .

За тастатуре код **IBM-PC** компатибилних рачунарских ситема (слика 6.6), енкодер преноси такозвани **SCAN CODE. Scan code** указује који тастер, или тастери су притиснути, али не даје никакав специјални код.

На пример, тастер **m** има **scan code** 50_{16}. Педесет није **ASCII** код за **m**, већ је то просто број који је придружен лабели, односно обележју **m**. То значи да у рачунару постоји специјални програм који се зове **транслатор тастатуре** који придружује значење добијеном **scan** коду. Овакав поступак се понекад зове **SOFT KEYS**, и ствара могућност да се промени значење тастера на тастатури. То значи да једна иста тастатура (**hardware**) може да се користи и у Русији и у USA и у Југославији, тако да садржи азбуку и распоред тастера који је типичан за сваку од ових земаља. То се постиже једноставном изменом програма који интерпретира **scan** кодове.

Обзиром на распрострањеност персоналних микрорачунара, описаћемо детаљније употребу појединих тастера при раду под оперативним системом **DOS**, који се најчешће користи код ових рачунара. Други оперативни системи, као и разни програми додељују тастерима другачије функције.

Тастатуре за **IBM PC** рачунаре се производе са различитим распоредом тастера, према стандардима појединих земаља. Без обзира који је распоред тастера на тастатури, она се може помоћу програма из **DOS**-а прилагодити да ради по било ком стандарду.

Треба, пре свега, идентификовати групе тастера. Већи део тастатуре заузимају слова, изнад њих су бројеви и специјални знаци (као на писаћој машини). Изнад слова су 12 функцијских тастера обележених са **F1, F2, ... , F12**. Ове тастере користе програми на различите начине, најчешће за позивање разних стандардних функција. Што се тиче оперативног система

DOS, неки од функцијских тастера користе се у току исправљања командних линија.

Са десне стране је нумеричка тастатура која се састоји од 17 тастера. Нумерички тастери могу да се користе и за контролу рачунара: на њима су исцртане стрелице, односно написане речи попут **Home, End, Insert, Del, PgUp**, итд. Између основне и нумеричке тастатуре налазе се три групе контролних тастера. Прва група се састоји од три тастера на којима пише **Print Screen / SysRq, Scroll Lock** и **Pause / Break**, друга од шест тастера на којима је исписано **Insert, Delete, Home, End, PageUp** и **PageDown**, а трећа од четири тастера на којима су исцртане стрелице:

Тастатура се са рачунаром најчешће повезује помоћу кабла, а могуће су разне опције. Ако је тастатура директно повезана са рачунаром (као код персоналних рачунара), енкодер може да шаље све битове истовремено. Овај поступак је познат као паралелни пренос, где за сваки бит постоји посебна линија, тј. 8 линија за бајт. Паралелни пренос се обично користи када су растојања мала и када имамо на располагању довољно линија. Када то није случај подаци се шаљу **бит по бит**, тј. серијски. Серијски пренос тражи постојање једног регистра који може да прихвати свих 8 бита истовремено и затим да их један по један шаље на комуникациону линију.

Овакав редни пренос штеди линије потребне за пренос података, али знатно успорава пренос, јер се један бајт преноси у осам интервала времена. Повезивање било ког уређаја на рачунар захтева поштовање одређених правила понашања, односно протокола Тако се, рецимо, при реализацији серијске везе, мора поштовати одговарајући стандард уз примену одговарајућег интерфејса, а један од њих је и **RS 232**.

6.2.2. ТАБЛА ПОДАТАКА

Велики број корисника рачунара не може, или неће, да куца на тастатури. За помоћ таквим корисницима развијен је низ уређаја који омогућавају другачији унос података, као што су: светлосне оловке, табле, палице (**joysticks**), екрани осетљиви на додир (**touch**) и мишеви. Табла је у суштини равна површина преко које се повлачи "перо", тј. "пенкало", и та површина је осетљива на "додир" са "пенкалом". Ове табле обично користе светлост, ултразвук или притисак да детектују присуство прста, оловке или неког другог инструмента. Табла може да шаље рачунару вертикалну и хоризонталну координату "пенкала" непрекидно, тј. у сваком тренутку времена, или пак само онда када је притиснут неки тастер. Из претходног описа је

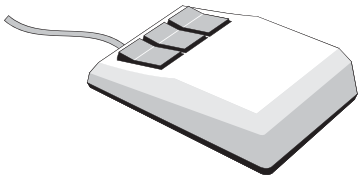
очигледно да рачунар од табле добија два (или више сигнала). На бази тих информација може се помоћу програма у рачунару одредити позиција додира пера на површини табле.

6.2.3. СВЕТЛОСНА ОЛОВКА

Светлосна оловка је цилиндрични уређај који личи на оловку. Када је њен врх ослоњен на екран, светлост која долази са екрана детектује се помоћу фотоћелије у оловци. Фотоћелија генерише струју која одлази у контролер који пореди сигнал из оловке са сигналима који долазе са екрана и на основу тога утврђује где се налази оловка, тј. одређује њен положај. Када се одреди положај оловке, контролер шаље информацију у рачунар. Један програм који прима тај податак може да пошаље специјални карактер на указану тачку екрана, или да придружи позицији на екрану неку команду и изазове неку акцију.

6.2.4. МИШ

Миш (**mouse**) је једна од најраспрострањенијих варијација табле и пера, а једна од могућих изведби дата је на слици 6.7. Миш се држи једном руком и креће се по равној површини. Правац и смер померања се детектују било ротирајућом куглом унутар миша, било фотоћелијом. Миш са фотоћелијом нема покретних делова, али захтева постојање специјалне површине са означеним вертикалним и хоризонталним линијама.



Слика 6.7. **Genius** миш

На мишу се налази једно, два или три дугмета, и свако од њих шаље другачији сигнал који се интерпретира помоћу једног програма у рачунару. Овај програм најчешће контролише положај једне врсте маркера на екрану, и показује кориснику текући "положај" миша. Маркери (показивачи) могу имати облик стрелице, крста или било који други и зову се курсори.

6.2.5. JOY STICK

Joy stick има вертикалну ручицу која се може померати лево или десно и напред или назад. У састав **joy stick**–а улазе и специјални електронски

склопови који могу да мере те помераје у односу на средишњи положај. Те информације се шаљу у рачунар који их даље обрађује помоћу специјалних програма.

6.2.6. РОТИРАЈУЋА КУГЛА

Ротирајућа кугла **(track ball)** садржи куглу која може да се слободно креће у било ком смеру и правцу. Корисник ротира куглу, а електроника детектује смер и брзину ротације. Ови подаци се шаљу рачунару, тј. опет неком програму који их даље обрађује. Ротирајуће кугле се користе као саставни део неких врста мишева.

6.2.7. ОПТИЧКИ ЧИТАЧИ

Оптички читачи раде на принципу емитовања светлости на површину са које се чита, а потом се одбијена светлост прихвата и обрађује. Светле површине имају велики коефицијент рефлексије а тамне мали. Посебна дигитална мрежа врши анализу добијеног рефлектованог снопа и он се претвара у одговарајуће информације, које се затим обрађују помоћу програма. Има више врста оваквих уређаја.

6.2.8. УРЕЂАЈИ ЗА ОПТИЧКО ПРЕПОЗНАВАЊЕ КАРАКТЕРА

Оптички читач може да чита некодиране алфанумеричке знаке штампане помоћу писаће машине или неког механографског уређаја на папиру. Сем тога, може да чита и бројеве писане руком, према угледном примеру. Читање се врши помоћу снажног снопа светлости који се креће према знаку и догод наилази на бео папир одбија се од њега што изазива команду за даље кретање снопа светлости у истом смеру. Када светлосни зраци наиђу на црн знак, они се одбијају, што преко фотоћелија изазива команду за привремено враћање снопа, а потом поновно наступање снопа према знаку, све док се не обиђе цео знак. Пре испитивања који је знак у питању, млаз прво испитује његову величину, која треба да буде једна од стандардних. Прочитани знак се упоређује са знацима који оптички читач има у својој меморији. Ако постоји сличност, знак се препозна, и шаље се даље, а уколико знак није препознат, детектује се грешка.

Оно што је извесно, један од основних облика комуникације човека са рачунаром, већ крајем овог века, биће писање помоћу пластичног или металног уређаја који је налик на пенкало. Овај начин биће нарочито погодан за преносне (**portable**), ручне рачунаре. Други, доминантан начин комуникације човека са рачунаром биће базиран на препознавању говора, тј. човек ће комуницирати са машином путем говора. У овој области већ постоје одређени резултати, за сада углавном у области препознавања

речи. Значајнији резултати у препознавању реченица очекују се почетком овог миленијума.

6.2.9. ЧИТАЧ БАР–КОДА

Бар–код (**bar-code**) служи за представљање нумеричких података на спе–цијално кодирани начин. Постоје две значајне примене бар–кода: за шифрирање артикала у трговини и представљање интернационалног броја књига у компјутеризованим библиотекама. У оба случаја се бар–код чита помоћу специјалног ручног пера са светлосним или ласерским зрацима. Стандардни европски бар–код, тзв. **EAN** садржи 13 цифара – 2 или 3 за шифру државе, а остале за шифру произвођача, шифру производа и 1 за контролни број. У меморији рачунара смештене су цене свих артикала. Бар–код се шаље у рачунар, где га прихвата програм који обрађује податке, и шаље цену продате робе терминалу који штампа рачун за купца.

6.2.10. ВИДЕО ДИСПЛЕЈ – ЕКРАН

Као што су папирне картице дуго биле главни извор података, тако су и штампани извештаји били главни облик излаза података. Иако су штампачи и даље врло значајни, основни облик приказивања резултата обраде у многим применама данас су видео дисплеји.

Данас се користе три врсте видео дисплеја:

– *катодне цеви монохроматске и у боји (**Cathode Ray Tube, CRT**),*

– *дисплеји са течним кристалом (**Liquid Cristal Display, LCD**),*

– *плазма екрани (**Plasma Display**).*

Битан фактор који у великој мери одређује квалитет свих врста дисплеја јесте резолуција, тј. могућност да разликујемо детаље на слици, па се за ове три врсте дисплеја може закључити следеће.

Катодне цеви (**CRT**) имају добру резолуцију, али су гломазне и троше ве–лику количину електричне енергије. Ови дисплеји ће у будућности користити **HDTV** (**high-definition television**) поступак, који ће омогућити резолуцију већу од 1920 **x** 1080 **pixel**–а. Два главна проблема код **CRT**–а остају: драстично се повећавају дубина и тежина са повећањем екрана.

Дисплеји са течним кристалом (**LCD**), су лаки, танки и мало троше, али имају малу резолуцију и треба им спољни извор светлости да би слика била видљива. Наиме, **LCD** материјали се налазе у сендвичу између две стаклене плоче, рефлектују више светлости када су њихови молекули ус–мерени дуж једне осе него када су усмерени дуж друге осе. Усмеравање се врши помоћу електричног поља. Али, да би уопште рефлектовали, светлост мора доћи споља. Данас се углавном користе код **portabl** рачунара (**note-**

book), али се предвиђа да ће почетком следећег века они постати стандард.

Плазма дисплеји, у сендвичу између две стаклене равни имају јонизовани гас и мрежу хоризонталних и вертикалних линија. Пропуштањем мале струје кроз ове линије у пресечним тачкама се мењају својства гаса. Ови дисплеји имају одличну резолуцију и немају проблема са видљивошћу.

За управљање радом дисплеја користе се информације које се обично смештају у тзв. **refresh** меморију. Ова меморија може бити део јединице видео дисплеја, или специјално поље у меморији рачунара, а у њој су смештени кодови података који се управо приказују. Свака врста дисплеја има одговарајући контролер, тј. електронску опрему која контролише његов рад, обезбеђује потребну аутономност у раду и преко које је остварена веза са рачунаром.

Главне потешкоће у коришћењу видео дисплеја су: ограничена количина података који се могу видети у неком тренутку времена (типично 24–25 линија са 80 карактера), као и то што се подаци не могу задржати трајно или за будућу примену, већ су привременог карактера. Оба ова проблема се превазилазе коришћењем штампача као излазних уређаја.

6.2.11. ШТАМПАЧИ

Скоро сви рачунарски системи имају потребу за издавањем резултата обраде у писаној форми, тј. посредством штампача. Постоји много различитих врста штампача који се међусобом разликују по брзини рада, квалитету и цени, и то су све битни параметри при избору штампача у рачунарском систему. Велики рачунарски системи често у свом саставу имају неколико штампача. Штампач је излазна јединица која омогућава најбрже исписивање излазних информација на папиру. Ово је, без сумње, најчешће коришћена излазна јединица, што је и разумљиво с обзиром на то да је писани облик документације још увек врло присутан, а често и неопходан у пракси пословних система. Према конструкцији и принципу рада, постоји више врста штампача. У основи их делимо на две групе: електромеханичке и физичко–хемијске.

Електромеханички су они који утискују знаке у папир механичким ударцем, (ударом чекића, иглице и слично). Физичко–хемијски утискују знакове на основу електрохемијских или фотоелектричних реакција.

Код великих рачунарских система, штампачи нису под директном контролом централне јединице и корисничког програма, већ су под контролом оперативног система и периферијских процесора. У изузетним случајевима, информације за штампање се привремено меморишу у неку спољну меморију, а касније се аутоматски шаљу у штампач.

Командни део штампача има више тастера за укључивање и проверу рада штампача, као и индикаторе за детекцију неправилних стања.

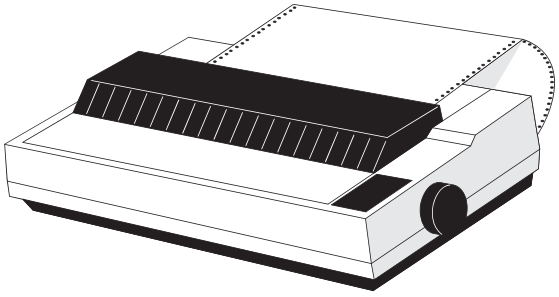
Линијски штампач са добошем у механизаму за штампање има следеће делове: добош са уграviраним знацима, ред чекића, траку, вођице и још неке делове. Између добоша и чекића налазе се папир и трака. Добош по целом свом обиму има уграviране стандардне знаке (64 знака). У једном реду су уграviрана само слова **A**, а у следећем реду слова **B**, тако да се обично на целом добошу налазе два низа редова свих знакова. Наспрам добоша по дужини се налази (за сваку позицију) ред чекића који ударају о папир у моменту када у том реду треба да се штампа на одређеном месту потребан знак. У једном моменту када је ред **A** добоша наспрам чекића, штампају се сва слова **A** која се налазе у реду који треба штампати, затим се штампају сва потребна слова **B**, па сва слова **C** итд.

Активирање појединих чекића у одређеном моменту постиже се побуђивањем одговарајућих електромагнета који преко својих опруга ослобађају чекиће. Када у компаратору дође до слагања знакова из детектора и бафер–меморије на одређеним позицијама, шаљу се импулси за активирање потребних електромагнета који опуштају опруге својих чекића, а ови ударају о папир.

Систем за транспорт папира састоји се обично од четири транспортера у чије зупце се намешта папир. Један пар транспортера се може прилагодити ширини папира. У систем за транспорт спада и неколико микропрекидача за детекцију стања папира.

За разлику од линијског штампача који штампа ред по ред, **серијски штампач** штампа знак по знак, а служи за штампање мањег броја кратких извештаја за које се не тражи велика брзина штампања.

Серијски штампачи могу бити матрични и са лепезом. Матрични штампач, слика 6.8., има главу за штампање која се помера дуж целог реда на папиру захваљујући свом покретном носачу. Осим механичких, матрични штампач садржи и бројне електронске делове.



Слика 6.8. Матрични штампач

Добре особине матричних штампача су: ниска цена, мале димензије и могућност формирања нестандардних знакова, укључујући штампање цртежа. Главна мана је што квалитет штампања није висок, јер знаци нису компактни већ се састоје од скупа тачака. Постоје деветопински и двадесет–четворопински матрични штампачи.

Серијски штампач са лепезом има компактне знаке уграviране на ободу једног котура које се обрће. Квалитет штампања му је висок, али је брзина мања него брзина матричних штампача. Главна мана је немогућност проширења стандардног сета знакова.

Постоји више врста штампача који раде без механичког додира, па свој рад заснивају на различитим физичко–хемијским принципима (електростатичком, термичком и др.).

Термички штампачи су серијски матрични штампачи. Имају главу са минијатурним отпорницима који су распоређени у облику матрице; отпорници на одређеним местима греју термоосетљиви папир на којем се јављају тамне тачкице чијим се комбинацијама стварају знаци. Термички штампачи захтевају скупи термоосетљиви папир и не могу истовремено да штампају више копија.

Ласерски штампачи су најбржи штампачи без механичког додира, слика 6.9. Светлосно осетљиви фотопроводни слој се налази на ротирајућем добошу. У току рада добош се излаже ласерским зрацима који су модулисани потребним знацима за штампање. Специјални суви прах пребацује наелектрисање добоша на претходно загрејан обичан папир. Ови штампачи се називају и странични, јер штампају целу страну истовремено, и то врло квалитетно и брзо. Што се тиче кућних рачунара, сматра се да ће због својих предности ови штампачи доминирати за црно–белу штампу.

Слика 6.9. Ласерски штампач

Штампачи са млазом мастила (**ink-jet**) су у новије време усавршени, тако да су им повећани брзина и квалитет штампања, док им је цена релативно ниска. Квалитет штампе је лошији него код ласерског штампача, али сматра се да ће они практично доминирати у области штампе у боји (пре свега када говоримо о кућним рачунарима).

Заједничка предност свих штампача, који раде без механичког додира, је у томе што раде брже, квалитетније и поузданије од јефтиних механичких штампача, а нису сувише скупи као квалитетни механички штампачи.

6.2.12. КООРДИНАТНИ ЦРТАЧИ – ПЛОТЕРИ

Ова врста уређаја служи као спона компјутера и графичке информације, и то у смислу израде графичких или текстуалних приказа, а на темељу дигиталних података. У основи постоје две врсте плотера, и то:

– ***плотери са обртним ваљком*** *за цртање на непрекидном папиру, и*

– ***хоризонтални плотери*** *за цртање на појединачним листовима.*

Код **плотера са обртним ваљком**, кретањем главе за цртање управљају сигнали који се доводе из рачунара. Кретањем ваљка се остварује померање папира, а њиме се такође управља из рачунара. Обе ове врсте кретања се остварују помоћу малих корака.

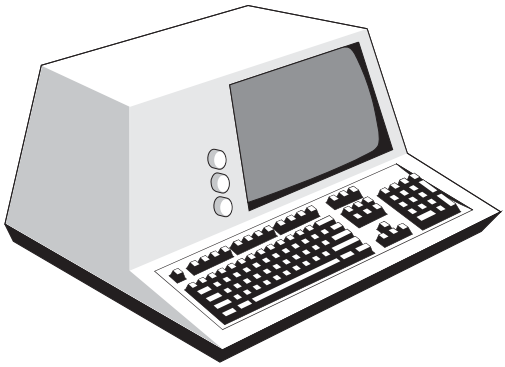
Хоризонтални плотер служи за цртање кривих линија на посебним листовима помоћу пера које може да се креће дуж **x**–осе и дуж **y**–осе. Кретање пера се остварује посебним сервомеханизмима. Захваљујући добрим перформансама (већој брзини и резолуцији) које поседују, хоризонтални плотери се све више користе за израду свих врста техничких цртежа.

6.2.13. ТЕРМИНАЛИ

Многи модерни рачунарски системи користе терминале као своју основну улазно–излазну јединицу. Терминали комбинују екран (**CRT, LCD**, плазма, итд.) као излазни уређај, са једном или више улазних јединица као што су: миш, тастатура, светлосна оловка, итд. Постоје и терминали који комбинују штампач као излазни уређај са једном или више улазних јединица. На слици 6.10 приказан је терминал који комбинује тастатуру са монитором.

Терминали се физички могу налазити близу централне јединице (у склопу рачунарског система) или пак могу бити врло далеко, на пример на радном месту корисника.
То значи да се између терминала и централног рачунара по правилу налази неки комуникациони систем, преко којег се преносе подаци. За пренос по–

датака се могу користити разне врсте комуникационих система, али се најчешће користе постојеће телефонске линије везе. Овакве комуникације се остварују посредством специјалног хардвера који се назива **модем**.



Слика 6.10. Терминал са монитором и тастатуром

6.3. СЕКУНДАРНЕ МЕМОРИЈЕ

Рачунарски системи поред типично улазних и типично излазних уређаја, користе, такође, и уређаје који имају обе ове улоге. Ови уређаји (магнетне траке, магнетни и оптички дискови), имају способност да сачувају излазни податак и да га учине расположивим за улаз (унос), касније, када је то потребно. Резултат употребе ових меморија је дуготрајно памћење података. Дуготрајне меморије се још зову и секундарне меморије, али истовремено то су и меморије врло великог капацитета, односно масовне меморије. Ове меморије омогућавају да се подаци избаце из главне, оперативне меморије рачунара, а да ипак остану доступни програму када је то потребно.

На слици 6.11. дат је преглед карактеристика различитих технологија из 1986. године које се користе за израду меморија (унутрашњих и спољних). Можемо уочити да је цена по јединици запамћене информације обрнуто сразмерна брзини приступа. Значи, брзе меморије су скупе и имају мали капацитет.

Али, у последњих 10 година дошло је до изузетног развоја технологије и пада цена, захваљујући пре свега масовној употреби кућних и личних рачунара. Тако се, рецимо данас, може купити 2 GB RAM–а за 80$, односно око 10^{-8} $ / биту док је цена диска 10^{-11} $ / биту.

Међутим, подаци на секундарним меморијама битно се разликују од оних у главној меморији, пре свега по начину организовања, начину адресирања односно идентификације, начину коришћења и начину приступа.

технологија	цена $ / биту	време приступа sec	начин приступа	врсте приступа	особине	врста материјала
биполарни полупровод.	10^{-1}	10^{-8}	случајан	упис / читање	NDRO, избрисива	електронска
метал–оксид полупровод.	10^{-2}	10^{-7}	случајан	упис / читање	DRO или NDRO избрисиве	електронска
феритна језгра	10^{-2}	10^{-6}	случајан	упис / читање	DRO избрисиве	магнетска
магнетни дискови	10^{-4}	10^{-2}	случајан полуслучајан	упис / читање	NDRO избрисиве	магнетска
магнетне траке	10^{-5}	10^{-1}	серијски	упис / читање	NDRO избрисиве	магнетска
папирне траке и карте	10^{-6}	10	серијски	само читање	NDRO избрисиве	механичка

DRO destructive readout
NDRO non-destructive readout

Слика 6.11. Упоредне карактеристике неких технологија за израду меморија

6.3.1. ОРГАНИЗАЦИОНЕ ЈЕДИНИЦЕ ПОДАТАКА

Мали број података није неопходно организовати, и са њима се може лако управљати. Организација података олакшава управљање великом количином података и њихово коришћење. Она подразумева груписање података на погодан начин у организационе целине, при чему треба уочити разлику између самих организационих јединица података и њиховог информационог садржаја.

Најмања организациона јединица података је "**бит**" (**bit, binary digit**, односно бинарна цифра **0** или **1**). Један бит је истовремено најмања количина информација која се може изразити. Код већине рачунара, и програмских језика, бит се не може адресирати, па се не може самостално ни користити. Један бит се записује у једној ћелији.

Најмања адресибилна јединица података, код микро рачунара и многих великих рачунара, којој се може директно приступити када је она у меморији рачунарског система јесте "**бајт**" (**byte**). Један бајт се састоји од осам бита. Но, постоје и рачунари код којих је бајт низ од шест нула и јединица, а постоје и рачунари који уопште не користе бајт као организациону јединицу меморије, већ је код њих најмања организациона јединица меморије **једна меморијска реч.**

Један бајт, као организациона јединица података, може да представља једну или две цифре BCD броја, или једно слово или један знак. Са осам бита рачунар може кодирати 2^8 = 256 различитих знакова, односно "симбола". Дакле, симбол је логичка организациона јединица података, а бајт можемо схватити као меморијски простор у који се може сместити један

знак, односно бајт је физичка (меморијска) организациона јединица по–датака.

Најчешће се један, два или више знакова групишу тако да чине већу логичку организациону целину коју називамо **елементарни** или **скаларни податак**, коме одговара физички простор за меморисање који зовемо **поље**. Једно поље се састоји од једне или више меморијских речи. Податак, и одговарајуће поље означавају се симболичким именом података, односно, називом поља. Морамо строго правити разлику између имена податка (поља) и вредности податка, тј. садржаја поља.

Садржај поља могу бити било какви подаци, бројни (нумерички), азбучни или алфанумерички, а најчешће су операнди над којима се извршавају различите обраде. Могући садржај поља одређује његову врсту (разне врсте нумеричких и алфанумеричких поља). Број меморијских локација у пољу одређује његову дужину, а она се одређује према највећем податку који је потребно регистровати у пољу одређене врсте.

Понекад се обрада може вршити над више елементарних података исто–времено. Таква логичка организациона јединица података, састављена од два или више поља зове се **групно поље**. Групно поље се још назива и **сегмент**. Групно поље, такође, има своје име; на пример, **дан**, **месец**, **година** и **дан-у-години**, чине структуру са именом **датум**. По потреби могу се узимати садржаји сваког поља понаособ: **дан**, **месец**, **година** и **дан-у-години**, или садржај читавог групног поља – **датум** (сва четири поља ис–товремено).

Једно или више групних поља, која су заједно доступна за обраду у рачунарском систему, чине логичку организациону целину која се зове **запис**. Запис се још зове и **логички слог**. Према томе, слог је скуп поља чији се садржај односи на један појам, појаву или догађај. Дужина слога одређује се бројем и дужином поља. Слогови могу бити фиксне и промен–љиве дужине. Један или више логичких слогова записују се на један физички простор, део секундарне меморије који се зове **физички запис** или **блок**. Структуру слога одређују: број, врста и дужина поља. Обзиром да се један исти податак у рачунару може приказати на више разних начина, приликом формирања структуре записа за свако поље треба користити најпогоднији облик са аспекта обраде. На пример, ако се поље користи претежно за рачунање, податак треба да буде регистрован у облику броја (бинарног или **BCD**), а ако је намењено за формирање извештаја податак треба да буде у облику стринга (низа знакова).

Ради обраде, потребно је да се сваки запис може идентификовати, тј. разликовати међу осталим записима. Као средство за препознавање слога служе једно или више поља у слогу, и то је **идентификатор** или **кључ.** Слог може имати више од једног кључа, и тада је један примарни, а остали

су секундарни. Вредности примарних кључева за сваки запис морају бити различите тако да примарни кључ једнозначно идентификује слог. Секундарним кључем издваја се група слогова, односно сви објекти или појаве који поседују исто одабрано својство, јер он представља обележје чија се вредност може понављати у већем броју слогова исте врсте.

На пример, за неког радника примарни кључ је матични број радника (шифра радника) и његова вредност указује увек на једног конкретног радника, а секундарни кључ може бити квалификација (одређује групу радника са одређеном школском спремом), радни стаж, итд.

Слогови се физички групишу у блокове, а логички се групишу у датотеке и записују се на секундарним меморијама, а њихова основна улога је чување података. Облик у којем се слог физички меморише може бити различит и зависи од начина његовог коришћења, са циљем да се рационалније користи меморијски простор и што брже и лакше приступа подацима.

Како су секундарне меморије релативно споре, то се у оперативну меморију рачунара, или обратно, при једном обраћању централног процесора периферном уређају, преноси често више логичких слогова који чине блок података или физички слог.

Блок података је основна јединица података која се чита или записује једном улазно–излазном операцијом рачунара. Број логичких записа у једном блоку зове се фактор груписања. Када је фактор груписања једнак јединици, онда у једном физичком слогу има само једнан логички слог.

Ако је коефицијент груписања три, три логичка слога чине физички слог или блок. Између физичких блокова постоји празан простор ради раздвајања, а код магнетне траке он постоји и ради покретања и заустављања. То је међублок или међузаписна празнина (**interblock, interrecord gap**). Пожељно је да овај празан простор буде што мањи, јер се на њему не могу чувати подаци.

Физички слог, осим података које садрже логички слогови, може да садржи и контролне податке. Контролне податке придодаје оперативни систем рачунара и они су, у том случају, неопходни за исправно меморисање и проналажење физичких слогова. Садржај и обим контролних података зависе не само од врсте периферијског меморијског уређаја (диск, трака итд.), већ и од оперативног система. Логички слогови се не морају увек груписати у блокове, али се груписањем смањује учестаност комуницирања централног процесора и спољних меморија, па се може знатно повећати брзина извршавања програма. Ефекти су утолико већи уколико је коефицијент груписања већи.

С друге стране, величина физичког слога ограничена је капацитетима унутрашње меморије и прихватне меморије рачунара (међумеморија),

величином логичких слогова као и својствима периферијског уређаја за чување података.

Још већа организациона јединица података је датотека. Датотека се састоји из слогова и садржи по неком критеријуму сродне податке. Зато можемо казати да је датотека скуп сродних података који се обрађују и чувају под заједничким именом. Име једнозначно одређује датотеку и служи за приступање датотеци и њено проналажење. Величина датотеке зависи од броја и величине логичких записа. Записи једне датотеке су обично исте врсте,. тј. имају исти број поља која се не могу разликовати по структури већ само по садржају.

Када се више логички повезаних датотека организује, и на неки начин међу собом повеже, добијамо **базу података**.

Датотеке се меморишу и чувају на масовним меморијама, и само повремено се копирају у главну меморију ради обраде. Секундарне меморије представљају разни уређаји базирани на разним медијумима за меморисање података.

6.4. СЕКУНДАРНИ МЕМОРИЈСКИ УРЕЂАЈИ

До појаве електронских цифарских рачунара, основни медијум за дуготрајно меморисање података биле су папирне траке. Но, врло брзо су почели да их потискују магнетни меморијски материјали (магнетне траке, магнетни дискови), док су данас у експанзији оптички дискови.

Секундарне меморије се деле према начину приступа подацима на две групе уређаја: секвенцијалне или серијске меморије и меморије са директним приступом.

6.4.1. ЈЕДИНИЦЕ СА СЕКВЕНЦИЈАЛНИМ ПРИСТУПОМ

Секвенцијалне меморије имају особину да је време приступа подацима зависно од положаја података на меморијском медијуму. У ову групу уређаја спадају две врсте меморија: са покретним магнетним материјалом (разне врсте магнетних трака) и магнетне меморије без покретних делова (велики померачки регистри реализовани помоћу магнетних мехурића).

6.4.1.1 Магнетне траке

Магнетне траке се користе од настанка електронских рачунара. Основне предности траке су ниска цена и велики меморијски капацитет, а главна им је мана у томе што се трака мора обрађивати секвенцијално, односно по

редоследу записивања на траци. Класична примена трака је: чување резервне копије датотека **(backup**), преношење датотека и евиденција трансакција.

Постојање резервне копије је погодно ако се шта деси са оригиналном датотеком (подаци се могу обновити са траке). Датотеке се могу копирати на траку, а затим се трака преноси на други рачунарски систем. Евидентирање трансакција се састоји у томе да се на траци бележе: претходна верзија датотеке, запис свих учињених трансакција и нови оригинал, што даје могућност обнављања записа и враћање стања пре последње обраде.

Магнетне траке данас, углавном, служе за снимање група од девет битова, познато као снимање на девет стаза. Подаци који се записују на траку су типа бајта, или карактера и иза једног следи низ других података записаних секвенцијално, слика 6.12. Осам битова који кодирају сваки карактер записани су паралелно широм траке. Упоредо са њима уписује се девети бит, бит вертикалне парности, који служи за контролу исправности записа податка.

стаза парности	1 1 0 0 1 1 0 0 0 1 0 1 1 1 1
стаза 8	1 1 0 0 1 0 1 0 0 1 0 0 0 0 0
стаза 7	1 1 1 1 1 0 0 1 0 1 0 1 0 1 1
стаза 6	0 0 0 0 1 1 1 0 0 1 1 0 0 1 1
стаза 5	0 1 0 1 0 1 1 1 1 0 0 0 0 1 0
стаза 4	1 1 1 1 0 0 1 1 1 1 1 1 1 1 1
стаза 3	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
стаза 2	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
стаза 1	1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

Слика 6.12. Меморисање података на траци са девет стаза

Користе се два система парности: парна и непарна парност. Код парне парности свака група од девет бита мора имати паран број јединица. Систем непарне парности захтева непаран број јединица. Приликом уписа сваког карактера на траку систем броји јединице и у девети бит додаје се потребан бит да би био задовољен услов парности. На слици 6.12 примењен је услов парне парности.

Парност се користи за проверу исправности прочитаних података са магнетних медија. Након читања кода са траке, систем броји број јединичних бита и тестира парност. Ако је нађен погрешан број бита 1 јавља се грешка парности, па треба поново прочитати те исте податке.

Парност помаже у откривању грешке у једном биту, али је немоћна код грешке у два бита (у парном броју бита). Једно од решења овог проблема је

коришћење другог бита парности записа дуж траке, тј. лонгитудинална провера парности.

На слици 6.13 дато је стање једног блока података. У **EBCDIC** коду је записана реч **COMPUTER** са непарном парношћу. На слици 6.14 показан је ефекат грешке у једном биту, и поступак корекције.

	C O M P U T E R	*битови подужне парности*
стаза парности →	1 0 1 1 1 0 1 0	0
стаза 8	1 1 1 1 1 1 1 1	1
стаза 7	1 1 1 1 1 1 1 1	1
стаза 6	0 0 0 0 1 1 0 0	1
стаза 5	0 1 1 1 0 0 0 1	1
стаза 4	0 0 0 0 0 0 0 1	0
стаза 3	0 1 1 1 1 0 1 0	0
стаза 2	1 1 0 1 0 1 0 0	1
стаза 1	1 0 0 1 0 1 1 1	0

Слика 6.13. Потврда вертикалне и подужне (лонгитудиналне) парности

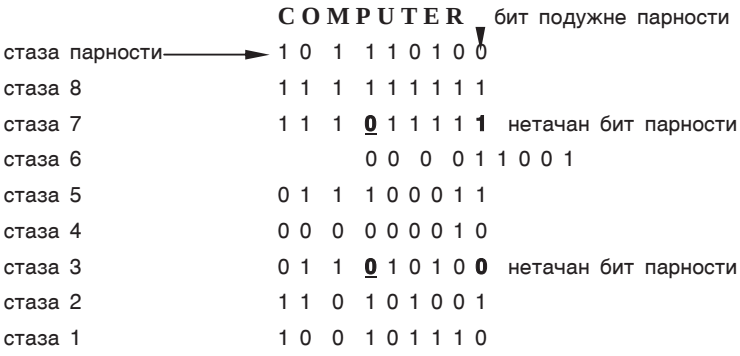
У запису на слици 6.14 у свакој колони и врсти записа има непаран број јединица. У читању је направљена грешка у слову **M**, уместо 001010111 прочитано је **1**01010111. При провери вертикалне парности одмах је детектована грешка у колони 3 (има 6 јединица).

некоректан бит парности →	**C O M P U T E R**	*битови подужне парности*
стаза парности	1 0 **1** 1 1 0 1 0	0
стаза 8	1 1 1 1 1 1 1 1	1
стаза 7	1 1 1 1 1 1 1 1	1
стаза 6	0 0 0 0 1 1 0 0	1
стаза 5	0 1 1 1 0 0 0 1	1
стаза 4	0 0 0 0 0 0 0 1	0
стаза 3	0 1 1 1 1 0 1 0	0
стаза 2	1 1 0 1 0 1 0 0	1
стаза 1	1 0 **1** 1 0 1 1 1	**0** ← ***некоректан бит подужне***

парности

Слика 6.14. Детекција и корекција грешке у једном биту

При провери хоризонталне парности детектована је грешка у 1. стази, где се сада налази 6 јединица. Значи да и врста и колона имају погрешну парност, а грешка је настала у пресеку колоне и врсте и може се кориговати.



Слика 6.15 Детекција двоструке грешке у једном знаку (по вертикали)

Ако би настале две грешке у истој колони, слика 6.15 (или истој врсти), оне се могу само детектовати, али не и кориговати.

Грешке могу настати и тако да остану неоткривене код овог система парности, па су развијене и неке друге, знатно сложеније технике контроле парности.

6.4.1.1.1 ФАКТОР БЛОКИРАЊА ТРАКЕ

Број знакова који се могу уписати на једном инчу траке зове се густина записивања (**density**). Прве траке су имале 800 бајта по инчу (800 **BPI, bytes per inch**), или 800 **CPI** (**characters per inch**). Са развојем технологије та се граница померала и сада су у употреби траке 1600 **BPI**, а има и оних са 6250 **BPI**.

Густина и брзина траке су два главна фактора који одређују брзину преноса података између траке и рачунара. Траке које се крећу брзином 10 инча у секунди са густином 800 BPI имају брзину преноса од 62,500 бајта у секунди. У стварности су брзине траке знатно веће (до 200 инча у секунди), па су веће и брзине преноса, типично од 200 до 1500 Кбајта у секунди.

Још једна чињеница битно одређује употребу траке. Наиме, траке се крећу само у току уписа или читања блока података, а затим се заустављају и чекају следећу инструкцију. Такође треба напоменути да се подаци са траке читају и на њу записују само у једном смеру кретања, и при томе линијска брзина кретања мора бити равномерна (константна).

Уређаји за покретање траке имају велику брзину како окретања тако и покретања и заустављања. Дакле, да би се што брже пронашао податак, трака се мора што брже кретати. Због тога је корисно да колутови за на–

мотавање траке имају што мању инерцију, односно масу. Времена покретања и заустављања могу се свести на величине реда једне милисекунде. Брзине трака испод магнетних глава су различите. Бржи системи су и скупљи. Уз брзину за коришћење траке постоји и брзина искључиво за премотавање.

Посматрајмо програм који треба да прочита блок од 80 карактера (1 линија) са тастатуре и смести на траку. После сваке линије, трака се зауставља и чека да оператер укуца следећи ред. Покретање и заустављање траке захтева неко време. Како тада брзина траке није константна, ово време се не може користити, јер би растојања између појединих битова била неједнака и не би било могуће користити те податке.

Током овог времена на једном делу траке настала је празнина (**gap**) тј. међублок или међузаписна празнина (**interblock, interrecord gap**), и она се не може користити за смештање података. Овај део траке је обично дужине 0.3 до 0.95 инча, зависно од модела.

Ако бисмо користили траку од 1600 BPI са брзином 10 инча у секунди, за запис једног реда треба нам 0.05 инча. Ако је међублоковска празнина 0.75 инча онда је искоришћено само 6% траке за запис (корисна површина), а 94% је изгубљено у празнинама. Код траке од 6250 BPI тај однос би био још гори.

Једино решење да се смањи расипање меморије (траке) јесте коришћење већих блокова. Све што треба записати на траку, смешта се најпре у један посебан део меморије, међумеморију (**buffer**), а тек онда уписује на траку. Упис на траку се врши тек онда када је међумеморија пуна.

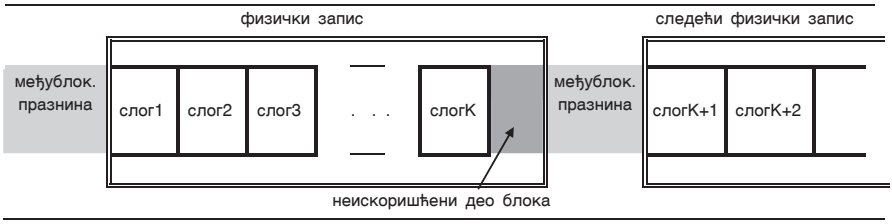
Када програм пошаље команду за упис на траку, један програм из оперативног система **IOCS (Input / Output Control System)**, усмерава податке у одговарајући део међумеморије. Програм **IOCS** памти број записа у међумеморији, и тек када је она пуна, овај програм записује садржај читаве међумеморије на траку као један запис.

Ако једну линију посматрамо као једну логичку целину, логички запис, у делу траке од 1 инча са 800 **BPI** (физички блок), може се записати 100 линија. Значи физички запис садржи 100 логичких записа.

Овакав начин уписа података повећава проценат искоришћености траке са 12% на 57%. Процес груписања више логичких записа у један физички запис познат је под именом блокирање, а број логичких записа (K) у једном физичком блоку зове се фактор блокирања, слика 6.16.

Када чита податке са траке, програм мора знати величину блока и фактор блокирања, и мора читати читав блок, а не само логички запис који му тренутно треба, иначе ће у противном доћи до губљења података. Да би се

то што лакше постигло, оперативни систем на почетку сваког блока (не само на траци већ и на диску) уписује неке контролне податке. Они помажу да се физичка дреса података на медијуму одреди на бази неког логичког кључа или дела самог податка.



Слика 6.16. Однос логичког записа–слога и физичког записа–блока

Трака се намотава на колутове који се могу скидати и поново постављати на уређаје за читање и писање. На тај начин се може сместити валики број података, ограничен само складишним простором корисника. Због тога се велике количине различитих статистичких и других података, које су потребне у различитим временским периодима, обично смештају на магнетне траке. Према томе, на магнетним тракама се чувају они подаци којих има много, а не морају бити доступни сувише брзо. Чување података обавља се уз врло ниску цену.

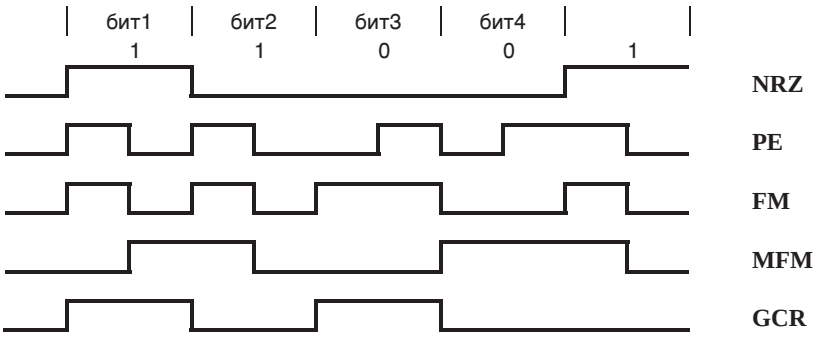
6.4.1.2 Кодирање података

Дигитални записи имају велику густину паковања, па је основни проблем како обезбедити промену магнетног поља да би записали податак, јер је директан запис немогућ. Дакле, мора се вршити кодирање тако да се и нуле и јединице могу препознавати у низу цифара.

На слици 6.17. приказано је неколико начина кодирања. Временски интервал трајања једног бита зове се ћелија.

NRZ1 кодира промену магнетног флукса сваки пут кад се појави бит један. Нуле су записане без промене флукса, па се јавља проблем одређивања почетка бита, када наиђе више нула једна за другом. Решење је довођење спољног такт сигнала који ће указивати на почетак сваког бита.

Траке са великом густином паковања користе фазно или Манчестер кодирање (**PE**). Овде се флукс мења на средини ћелије сваког бита. Промена са високог на ниски ниво означава јединицу, а са ниског на високи ниво означава нулу. Овде није потребан спољни такт сигнал.



Слика 6.17. Уобичајене шеме дигиталног кодирања

Многе дискете користе метод фреквентне модулације **(FM**). Овде се флукс мења на почетку сваке ћелије бита и додатно на средини сваке ћелије у којој је јединица.

За дискете са двоструком густином записивања користи се модификована фреквентна модулација **(MFM**), где се флукс мења на средини ћелије у којој је јединица, али нема обавезних промена на почетку ћелије, сем кад наиђу две нуле заредом.

Најновији начини кодирања примењују се над групама битова тако да се они трансформишу у нову групу битова у којој нема комбинација са две нуле у низу. Један начин је **GCR** (**Group-Coded Recording**) који користи пет битова за приказивање групе од четири бита. Како је $2^5 = 32$ то има двоструко више кодних комбинација, а користе се само оне где нема више од две нуле једна до друге у низу, којих има 17. Сувишна комбинација је 11111, и она се користи за синхронизацију.

Највећи недостатак траке је у томе што она има непромењиву секвенцијалну природу, па ако после 10. записа треба да обради запис 1000 мора се прећи преко свих записа од 10 до 1000. Секвенцијалност траке захтева сортирање записа пре почетка обраде, и тада не би било премотавања унапред и уназад. Но, ажурирање записа на траци тражи копирање на другу траку, јер упис на исто место, без премотавања уназад, није могућ. Да би се превазишли ови проблеми, у обради података се често захтева приступ било ком запису, у било које време, тј. директни, односно, случајни приступ.

6.4.1.3 Стримери и касете

Магнетне траке са девет стаза које се користе у великим системима су сувише скупе за мале микрорачунарске системе, па се у њима обично користе стримери и касете. Стримери (**streaming tape**) се не могу зауста–

вити и покренути на дужину уобичајених међублоковских простора. Они се заустављају споро, затим се премотају уназад довољно далеко да се могу поново покренути и достићи брзину до следећег корисног блока. Стримери су ипак врло корисни за упис врло великих блокова, па се користе за смештање резервних копија, или копирање свих датотека са диска. У овим условима стример ради континуално и ефикасно.

Касете се користе у врло јефтиним микрорачунарским системима. Неки чак користе најобичније аудио касете, док други користе специјалне касете. Имају густину паковања до 1600 BPI, али запис је серијски, карактер је низ од 8 бита (као и стримери и оне имају једну стазу, тј. једну уписно–читајућу главу), а брзина је до 15 инча у секунди.

6.4.2. ЈЕДИНИЦЕ СА ДИРЕКТНИМ ПРИСТУПОМ

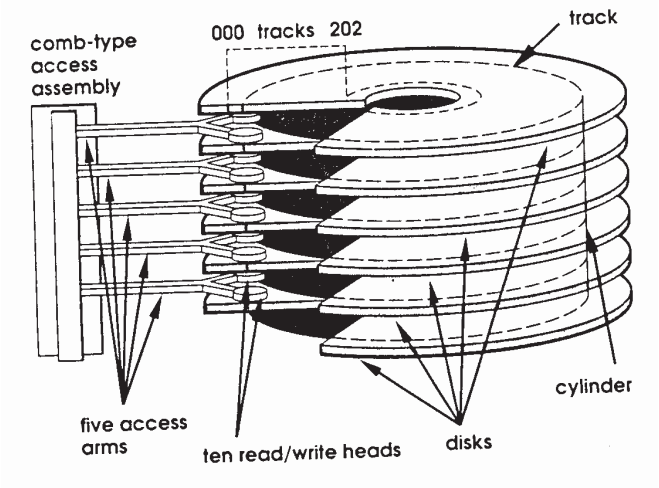
Ове јединице допуштају директан приступ једном запису, његово ажурирање и поновно записивање на његово претходно место. Овим записима се може приступити на бази кључа или задавањем физичке адресе. Кључ је посебни податак, или део записа, који омогућава брзу идентификацију жељеног записа. У ове уређаје спадају разне врсте тврдих магнетних дискова (**hard disks**), измењивих магнетних дискова (**floppy disks**) и неколико врста оптичких уређаја, оптичких дискова (**optical disks**).

6.4.2.1 Магнетни дискови

Диск спада у групу директно адресибилних меморијских јединица (**DASD, Direct Access Storage Device**), мада, истини за вољу, сви дискови имају по–лудиректни приступ. Наиме, директно се приступа површини и стази (путем позиционирања одређене главе на одређени цилиндар), а затим се подацима унутар цилиндра може приступити искључиво секвенцијално.

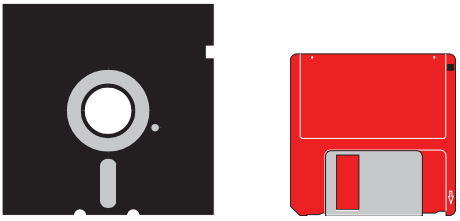
Механичка носива подлога код те врсте масовне меморије је у облику диска, па јој одатле и име. То може бити само један диск, ако се жели уписивати реалитовно мали број података, а може бити и вишеструки диск, слика 6.18.

Подаци се читају и уписују тако што дискови ротирају, а помоћу магнетних глава, тј. малих завојница кроз које тече струја, чита се и пише. При про–тицању струје кроз завојницу ствара се магнетно поље које магнетише површину диска на том месту и уписује јединицу. Магнетисање површине значи да одређени, врло мали делић површине намењен упису једног бита, достигне засићење, тј. максималну магнетну индукцију.



Слика 6.18. Тврди диск са 10 површина

Постоје две врсте магнетних дискова: неизмењиви (**hard**) и измењиви (**floppy**) дискови. Подлоге на које се наносе магнетни материјали за памћење могу бити чврсте или савитљиве, па према томе постоје чврсти (**hard**) дискови или савитљиви дискови (дискета, **floppy disk**), слика 6.19.



Слика 6.19. Дискете од **5.25** и **3.5** инча

Они се међу собом разликују по много параметара: по конструкцији, капацитету, брзини рада итд. Но, једна од најважнијих разлика је у томе што код тврдих дискова глава не остварује физички контакт са магнетном површином, а тврди диск се окреће непрекидно од тренутка укључења до искључења. Код измењивих дискова глава остварује физички контакт са површином диска, а диск се окреће само онда када се приступа подацима и након приступа се зауставља.Савитљиви дискови се употребљавају у микрорачунарима, јер они по величини, капацитету записа и цени више одговарају микрорачунарима, него велики чврсти дискови, али их из упо–

требе полако потискују други медијуми (оптички дискови). ористе се углав–ном за пренос података с једног на други рачунар. У микрорачунарима се употребљавају чврсти дискови, посебна верзија такозваних "винчестер" (**winchester**) дискова, која је по димензијама, цени и потреби за одржавањем прилагођена јефтиним микрорачунарима. Наиме, винчестер дискови су смештени у хериметички затворено кућиште са пречишћеним ваздухом, што омогућује да глава буде врло близу површине. Што је глава ближе површини, могућа је већа густина паковања података. Значи да ови дискови имају већи капацитет, а мање димензије и цену. При томе, пречишћени ваздух смањује и корозивно дејство околине. Али, ови модули нису погодни (лаки) за замену, тј. нису погодни као **off–line** меморије. Маг–нетни диск има једну или више кружних површина на које је нанет слој оксида феромагнетног материјала. Ова површина се окреће око осе, док се уписно–читајућа глава помера радијално ка оси ротације и од ње. Комби–новањем ова два кретања може се директно приступити свакој тачки површине. Тврди дискови се обрћу брзином 3600 обртаја у минути, док се флопи дискови обрћу обично 10 пута спорије. Уписно–читајућа глава, код тврдих дискова, плива на ваздушном јастуку који се јавља и као последица велике брзине обртања, док се глава код меких дискова у току рада спушта и ослања на површину, па предуга употреба има за последицу ме–ханичко оштећење диска или главе.

Тврди дискови се најчешће састоје од неколико плоча и неколико глава. Све главе се обично покрећу заједно, а код неких скупих система главе се деле у две групе које имају независно покретање. Подаци се памте на површини диска који је издељен на кружне прстенове, стазе (**track**). Стаза је логичко поље, јер све стазе, на свим меморијским површинама, на истом полупречнику, чине физичко поље звано цилиндар. Цилиндар је адресиби–лан, задавањем позиције 15, погон за покретање магнетних глава позиционира све главе изнад стазе 15 на свим дисковима.

Време потребно да се глава помери са једне стазе на другу назива се време тражења (**seek time**). Најкраће време за прелазак на суседну стазу креће се од **2,5–10 ms** код тврдих дискова до **90 ms** код измењивих. Просечно време позиционирања на ма коју позицију обично се, код дискова за главне рачунаре креће око **10 ms**, док се код микрорачунара креће од **10** до **20 ms**.

Но, након позиционирања главе подаци нису одмах на располагању, па треба сачекати да се диск окрене до места где се налазе жељени подаци. Просечно треба сачекати да се диск окрене за пола круга, што је при брзини до 3600 обртаја у минути равно 8**ms**. Овај временски интервал зове се ротационо кашњење.

За разлику од траке, где се један карактер (8 бита + парност) памти од–једном, на диску се подаци памте дуж стазе бит по бит, секвенцијално.

Брзина преноса података између диска и рачунара је функција густине паковања и угаоне брзине ротирања диска. Код тврдих дискова се креће од 600 **KB/sec** до 3 **MB/sec**, док је код меких дискова 100–200 **KB/sec**.

Као и код траке, подаци се на диск не смештају изоловано један по један, већ у већим организационим јединицама. Основни облик организовања и логичког повезивања података јесу датотеке. Датотеке се на дискове могу смештати на више начина. То може бити континуалан слободан простор довољне величине да се упише читава датотека, или се пак датотека подели на блокове, а затим се поједини блокови смештају на слободан простор на диску.

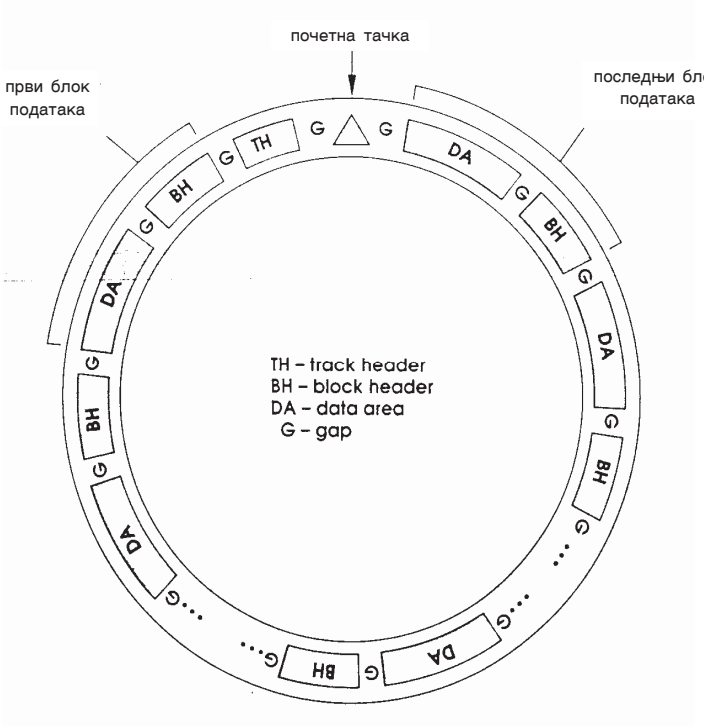
Један блок датотеке садржи једну или више логичких целина. И меморијски простор диска се, такође, дели на блокове, најчешће фиксне величине. Тада се за смештање датотеке не мора користити континуиран простор, а простор који датотека заузима постепено се повећава са порастом датотеке. Ово је такозвана архитектура са фиксним блоковима (**FBA**, **Fixed Block Architecture**).

Свака стаза је подељена на неколико блокова једнаке величине, који се зову **сектор**. Сектори су такође адресибилни. Сваки сектор се независно може доделити некој датотеци, наравно под условом да је слободан. Иначе адреса једног блока састоји се из три дела: **цилиндра, површине (глава), сектора**. Број сектора на стази варира и зависи од диска, али и од оперативних система, па рецимо **IBM-PC** имају 8 (прва верзија оперативног система, DOS 1.0), 9 или 15 сектора по траци. Понекад се код дискова великог капацитета организовано користе 2, 4 или 8 сектора, и они чине **грозд** (**cluster**).

Подела диска на секторе мора се извршити пре употребе диска, а процес који то обавља назива се форматирање (**formatting**). Информације о подели диска на секторе, као и низ других информација, уписују се на диск у једну табелу која се назива **контролни блок јединице**. У ову табелу се уписују и имена датотека, подаци о простору који је слободан и слично. Сличан контролни блок имају све улазно–излазне јединице.

Као што смо већ напоменули, подаци се на диск такође смештају у облику блокова, а не појединачно. На слици 6.20 је приказано како се, у општем случају, дуж једне стазе записују подаци. Има више разних приступа за организовање података на диску, а најчешће се користе **IBM**–ов **count–data format** и **count–key–data format**.

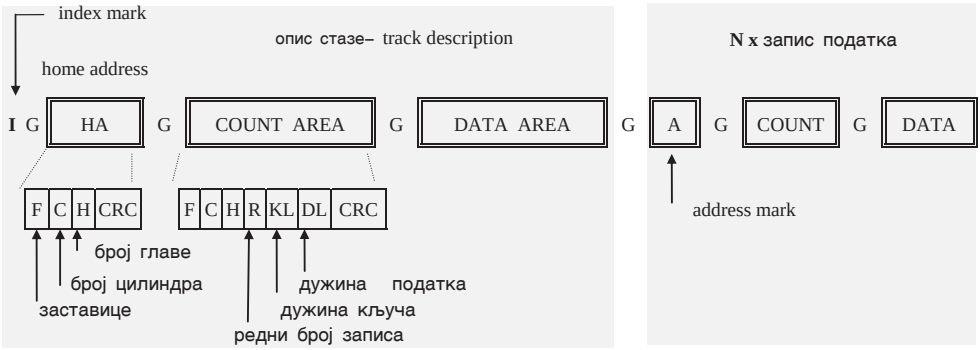
Поља означена са **G** представљају међублоковске празнине које раздвајају поједине делове стазе које садрже податке. Ови размаци су неопходни да би диск контролер (или рачунар) могли да испитају једну групу података и да одлуче шта даље да раде пре него што следећи блок података стигне испод уписно/читајуће главе.



Слика 6.20. Записивање података на диск

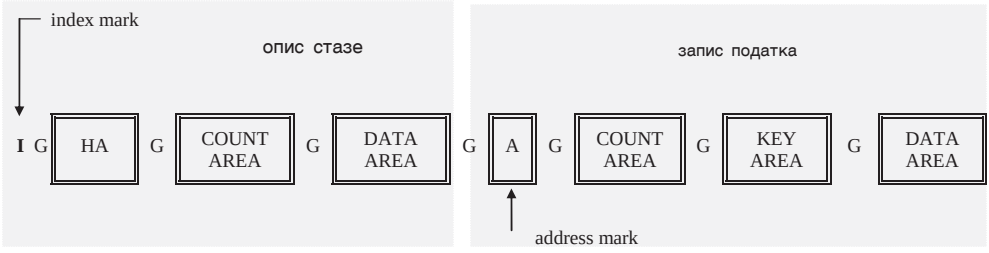
Свака стаза почиње једним заглављем стазе (**TH**) које се код **count-data** формата састоји из три дела: *индекса* (**index mark**), *кућне адресе* (**home address**, **HA**) и *описа стазе*. Поље **HA** садржи све потребне информације којима се идентификује стаза, а то су: **број цилиндра**, **број главе** и још два специјална поља, слика 6.21. Прво поље је **заставица** која указује на стање стазе (употребљива или неупотребљива), а друго поље се зове циклична провера (**Cyclic Check**, **CC**) или, чешће, додатна–редундантна циклична провера (**Cyclic Redundancy Check**, **CRC**).

Поље **Cyclic Redundancy Check** (**CRC**) служи за детекцију могуће грешке у подацима при њиховом записивању, читању или преносу. Користи се више разних техника за израчунавање **CRC**–а, а једна једноставна метода би била издвајање два најмање значајна бајта из збира бројева који се налазе у пољима **F**, **C** и **H**. Ако је у **F** број $F0_{16}$, у **H** број $0C_{16}$ а у **C** број $A6_{16}$, њихов збир је $1A2_{16}$, па ће у **CRC** бити записан број $\mathbf{A2_{16}}$. Као и парност, и **CRC** се израчунава при упису и записује се заједно са подацима. Када се подаци читају, **CRC** се поново израчунава и ако се разликује од оног који је добијен при упису сигнализира се грешка.



Слика 6.21. Опис записа на стази код **count-data format**–а

Count-Key-Data Format (слика 6.22) је врло сличан, само између поља са подацима и поља **count** постоји додатно **поље кључа** (**key area**). Ово поље обично садржи податак који сам корисник употребљава да би идентификовао дати запис. То може бити *име*, *идентификациони број*, *шифра производа* или неки други јединствени идентификатор. Употреба кључа омогућава лоцирање жељеног записа на диску помоћу кључа уместо помоћу **цилиндра**, **главе** и **редног броја записа** (или сектора).



Слика 6.22. **Count-key-data** диск формат

Да би се подаци исправно прочитали са диска, он мора да се окреће константном брзином. Промена брзине обртања може довести до погрешног читања, јер управљање диском се заснива на претпоставци да се поља битова смењују испод главе, неком унапред одређеном брзином. Но, код измењивих дискова (дискете) та брзина ипак није фиксна већ се креће унутар неких граница. Због тога се код дискета на сваком сектору налазе поља за синхронизацију која диск контролер препознаје, и на основу којих диск контролер одређује стварну брзину окретања диска и на основу ње коригује очитавање битова са диска.

Први хард диск је познати **IBM**–ов 305 **RAMAC** са 50 плоча пречника 24 инча, са капацитетом од 5 **MB**, густином од **2 килобајта по квадратном инчу** и ценом од 10 000$ по мегабајту. Заузимао је простор као два фрижидера. Данас дискови имају густину од **500 мегабита по квадратном инчу** и цену од **10 центи по мегабајту**. До краја века очекује се већа примена MR глава (magnetoresistive), а **IBM** је већ најавио дискове са густином од **5 гигабита по квадратном инчу.**

6.4.2.2 Оптичке меморије

Због свог огромног капацитета, механичке робустности и трајности података, оптички дискови (слика 6.23.) су у центру пажње развоја нових меморијских медија. Раде на принципу ласерског читања, а упис може такође бити ласерски или на неки други начин. На равну површину кружног облика наноси се танак филм материјала који служи као оптички медиј.

Слика 6.23. Оптички диск

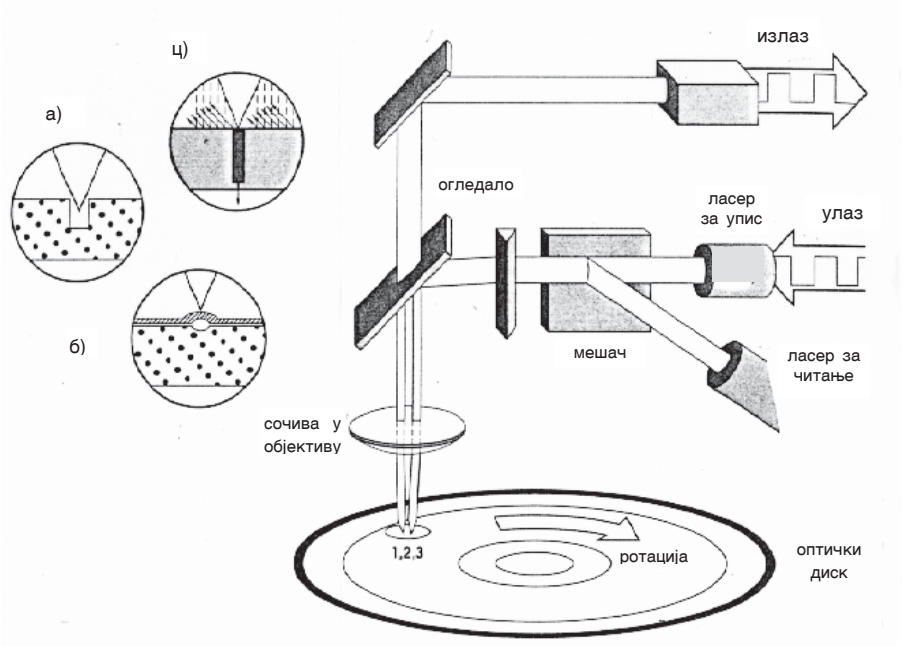
Данас су у употреби различити материјали на које се подаци могу уписати механички, термички или оптички (слика 6.24.). Код механичких записа на филму се налазе тањи и дебљи слојеви који, на разне начине, рефлектују ласерску светлост и симболизују нуле и јединице (слика 6.24. а)). Слична је употреба материјала са мехурићима који имају исту улогу као дебљи слојеви филма (слика 6.24. б)). Данас се много ради на примени материјала који под дејством топлоте мењају структуру из кристалне у амфорну и обрнуто, и при томе различито рефлектују поларизовану светлост зависно од стања (слика 6.24 ц)).

Разликујемо три врсте оптичких дискова:

1. CD-ROM (Compact Disc Read Only Memory)

2. WORM (Write Once Read Many)

3. ИЗБРИСИВИ (Erasable Media)



Слика 6.24. Оптички дискови користе једне ласере за упис информација на специјалну површину, а друге ласере за читање информација

Компакт дискови користе методе групног кодирања које конвертују осмо–битни податак у запис дужине четрнаест бита. Захваљујући томе могуће је користити врло ефикасне процедуре за детекцију грешака, знатно ефикасније него код магнетних медијума. Капацитет **CD-ROM** је неколико хиљада пута већи од капацитета измењивог диска. Но, главна мана му је још увек недовољна брзина преноса података.

WORM омогућују кориснику да једном упише податке, након чега се они могу само читати. Упис се врши трајним топљењем површине или прављењем мехурића, након чега измењена површина диска рефлектује ласерску светлост другачије него неизмењена. Ови оптички дискове се углавном користе за прављење трајних резервних копија и дуготрајно чување датотека.

Избрисиви оптички дискови користе магнетно–оптичке (**MO**) површине са материјалом који ротира поларизовану светлост. Угао ротације се подешава помоћу магнетног поља на медијуму. Ови материјали имају и особину да задржавају магнетизам, након што се загреју изнад такозване Киријеве тачке. Загревање се врши помоћу ласера, а иста та тачка се истовремено

излаже дејству магнетног поља које се производи помоћу магнетне главе за упис. Подаци се могу избрисати или променити поновним излагањем дејству снажног ласерског зрака и магнетног поља. Очекује се брз развој и велика примена ових оптичких дискова у најскорије време.

Будућност оптичких дискова је већ почела. Наиме, почетком 1998 појавили су се **DVD–RAM** оптички дискови са капацитетом од 2,6 **GB,** а данас су преко 18 **GB**. Први дискови су користили технологију са променом фазе (ротација поларизоване светлости), али се предвиђа да ће се убрзо појавити и такозвани **MO7 DVD** дискови са пречником од 120 **mm**. Који ће од ова два принципа превладати зависи од тога који од њих ће брже повећавати густину записа на површини. Предвиђа се да ће у скорој будућности ови дискови имати пречник од 8 **cm**. Шта ће се даље догађати у великој мери зависи од развоја ласера. Очекује се да ће они ићи у подручје зелене и плаве светлости, што ће омогућити даље смањење тачака на оптичким дисковима.

6.5. ЗАКЉУЧАК

Многи модерни системи користе терминале као примарну улазно–излазну јединицу. Терминал комбинује дисплеј са једном или више улазних јединица, као што су тастатура, миш, оптичка оловка итд.

Понекад се као излазна јединица код терминала користи принтер, који и даље представља једну од основних јединица у сваком рачунарском систему. Цене принтера се крећу од неколико стотина долара до неколико стотина хиљада долара, јер неки могу да одштампају 50 карактера у секунди, а неки, пак, неколико стотина страница у минути.

Рачунарски систем мора обезбедити памћење података на дуже време. Главна меморија није погодна за овај задатак јер је скупа, ограничене величине и избрисива након искључења напајања. Основни уређаји који обављају ову функцију су магнетне траке, дискови и оптички дискови. Без обзира на врсту уређаја, подаци се логички групишу и на медије се смештају организовани у датотеке.

Магнетна трака обезбеђује јефтино памћење великог броја података. Но, она има огромнуману, а то је предодређеност за секвенцијалну обраду. Уређаји типа диска омогућавају приступ било ком податку у било које време. С обзиром да се папирни медији данас врло мало користе, може се сматрати да су магнетне траке најјефтинија меморија, али и најспорија. Међутим, сматра се да оне имају блиставу будућност. Очекује се знатно повећање брзине преноса података, ускоро ће се појавити траке са 16 глава, а главна истраживања су усмерена ка томе да са аспекта оперативног система траке личе на диск (тј. да имају неку врсту директног приступа). Такође се ради на реализацији оптичких трака.

Магнетни дискови су бржи и скупљи, али ће још дуго бити основна врста масовне меморије. Могућности магнетних дискова ограничава такозвани, *суперпарамагметни ефекат* (губитак магнетизма на собној температури услед сувише малих димензија

магнета). Овај ефекат ограничава густину на диску на 50 до 100 гигабита по квадратном инчу.

Оптички системи су још увек у почетној фази примене, али се очекује њихов значајан продор јер имају велики капацитет (један CD-ROM има приближно GB), велику трајност чувања података (практично су вечни) и мање су осетљиви на утицаје спољашње средине (механичке, топлотне итд.). За оптичке дискове будућност је већ почела, тј. са појавом DVD–RAM дискова са 2,6 GB.

Оно што стварно представља будућност масовних меморија јесу холографске меморије које имају огромне могућности у погледу капацитета, а дефект у медијуму не изазива губљење података.

6.6. ПИТАЊА

1. Које врсте улазно–излазног преноса постоје?
2. Како се обавља програмирани улазно–излазни пренос?
3. Који је начин програмираног U/I преноса бржи, а који је поузданији?
4. Да ли и при директном приступу меморији централни процесор учествује у преносу података?
5. Које су предности прекидног улазно–излазног преноса?
6. Шта омогућавају периферијски процесори?
7. Које су типичне улазне јединице и која је њихова намена?
8. Шта је то scan code, и шта он омогућава?
9. Који ће облик комуникације између човека и машине у скорој будућности доминирати?
9. Које врсте видео дисплеја постоје и које су њихове особине?
10. Које врсте штампача постоје?
11. Зашто је квалитет штампе код матричних штампача лошији него код других врста штампача?
12. Врсте секундарних меморија.
13. Шта су логичке, а шта физичке организационе јединице података?
14. Како се врши контрола парности и корекција грешке?
15. Шта је фактор блокирања траке?
16. Шта је стример?
17. У чему се разликује запис података на траци и стримеру (и касети)?
18. У чему је разлика између приступа подацима на траци и диску?
19. Шта све сачињава физичку адресу блока података на диску?
20. Које су предности оптичких меморија у односу на магнетне?
21. Које врсте оптичких меморија постоје?

6.7. КЉУЧНЕ РЕЧИ

- архитектура са фиксним блоковима (**FBD, Fixed-Block-Architecture**)
- блок (**block**)
- блокирање (**blocking**)
- циклична провера података (**Cyclic Check**, **CC**)
- цилиндар (**cylinder**)
- датотека (**file**)
- директан приступ меморији (**Direct Memory Access**)
- диск **(hard disk**)
- дискета (**floppy disk**)
- дисплеј са течним кристалом **(LCD, Liquid Cristal Display**)
- **DRO destructive readout**
- фактор блокирања (**blocking factor)**
- фазно кодирање
- физички запис (physical record)
- форматирање (**formatting**)
- фреквентна модулација, **FM**
- групно кодирање
- густина (**density**)
- избрисиви медиј **(erasable media**)
- IEEE 488
- **joy stick**
- катодна цев (**Cathode Ray Tube**)
- компакт–диск, **CD-ROM** (**Compact Disk Read Only Memory**)
- контролер (**controler**)
- координатни цртач (**ploter**)
- кључ (**key**)
- ласерски штампач (**laser printer**)
- логички запис (**logical record**)
- лонгитудинална парност
- међублоковска празнина (**interrecord gap**)
- међумеморија (**buffer**)
- миш (**mouse**)
- модификована фреквентна модулација, **MFM**
- **Multibus**
- **NDRO Non-Destructive Readout**
- низ података, поље (**array**)
- оптички читачи (**optical readers**)
- оптичко препознавање карактера (**Optical Character Recognition**)
- паралелни пренос (**parallel transmisson**)
- парна, непарна парност (**even, odd parity**)
- парност (**parity**)
- периферијске јединице (**peripheral devices**)
- периферијски процесори (**peripheral processor**)
- плазма екран (**plasma screen**)
- празнина (**gap**)
- прекид (**interrupt**)
- програмирани пренос података
- редундантна циклична провера (**Cyclic Redundancy Check**, **CRC**)
- ротационо кашњење **(rotational delay**)
- ротирајуће кугле (**track balls**)
- **RS 232 C**
- **scan code**
- сектор (**sector**)
- серијски пренос (**serial transmission**)
- стаза **(track**)
- светлосна оловка (**light pen**)
- **S-100**
- табла података (**data tablet**)
- тастатура (**keyboard**)
- вертикална парност
- време тражења (**seek time**)
- **WORM (Write Once Read Many**)
- запис (**record**)
- логички запис, слог
- знак, симбол, карактер
- штампач (**printer**)
- штампач са мастилом (**ink jet printer**)

7. СТРУКТУРЕ ПОДАТАКА

Податак је у општем случају одређени запис о неком догађају, појави или карактеристици из околине коју називамо објективна стварност. Подаци који се обрађују у рачунарима представљају дискретне чињенице или запажања кодирана помоћу бинарних цифара: 0 и 1. Подаци уједно представљају и средства за изражавање информација. Тек када се подаци користе за доношење одређених одлука с циљем решавања неког проблема, односно тек онда када спознамо њихов смисао и када нам повећају знање они прерастају у информације. Подаци имају и одређена својства: тачност, време трајања, област важења и слично. Појмови податак, информација и знање су међу собом повезани, тако да дају одређену слику околине (слика 7.1.).

Слика 7.1. Однос података, информација и знања

Подаци се могу прикупљати, обрађивати, чувати, а може се мењати начин њиховог записивања и коришћења. Уколико се податак записан на неком медију не користи (и неће се никада користити) он престаје да буде податак.

Рачунарски системи користе у свом раду искључиво бинарне бројеве. У меморији рачунара подаци се смештају као низови битова који се могу груписати, па тако чине бајтове или речи. Појединачни бајтови или речи у меморији могу представљати број са или без знака, неки од специјалних кодова, инструкцију итд.

Проучавајући разне врсте меморијских уређаја видели смо да су рачунарски подаци смештени најчешће у групама, које се могу звати блокови, записи или сектори. Видели смо, такође, да се бројеви могу организовати тако да представљају код операције или операнде у инструкцијама. Без обзира шта бројеви представљају, бајтови и речи се у меморији рачунара организују на посебан начин тако да представљају **структуре података**.

Структуре података су скупови података који су међусобом повезани неким релацијама. Они служе да повежу програмерски **логички** поглед на податке, и њихов **физички** смештај у меморији рачунара.

Структуре података су скуп правила помоћу којих се појединачни подаци стављају у одређене односе са другим подацима. Оне обезбеђују средства за организовање података и решавање проблема. Многи проблеми не би уопште били решиви, ако не би сви делови података били стављени у одређене релације међу собом, односно ако би остали независни. Структуре података: табеле, листе, низови, повезане листе, стабла итд. помажу у решавању проблема обраде великог броја података.

Са тачке гледишта хардвера рачунарског система и његовог оперативног система, логичка организација меморијских структура није од великог значаја. Оно што њих занима јесте стварна организација података на диску или траци, и како су бајтови и речи смештени у главној меморији. Један део оперативног **система-систем за управљање подацима** је задужен да обезбеди везу између физичке и логичке организације података.

Подаци унутар једног рачунарског система могу бити различитог типа, што зависи од скупа вредности које тај податак може да има, а уједно тип податка одређује скуп операција које се над тим подацима могу извршити.

Све податке можемо сврстати у две основне групе:

– *елементарне или скаларне,*

– *сложене или нескаларне податке.*

7.1. СКАЛАРНИ ПОДАЦИ

Најпростија структура података садржи у себи изоловане, појединачне податке који се зову **скалари**. Скаларни подаци се могу класификовати као проблемски и контролни подаци. Проблемски (програмски) подаци су имена и бројеви које програмери користе у својим програмима, као и подаци за контролу тока програма. Контролни подаци су неопходни за правилно извођење програма и значајнији су за оперативни систем него за корисничке, апликационе програме.

7.1.1. ПРОГРАМСКИ ПОДАЦИ

Постоји неколико врста програмских, односно проблемских података, а сваки од њих може имати константну или променљиву вредност, тј. могу бити **константе** или **променљиве**.

Целобројни подаци (**integer**) су цели бројеви без децималне тачке или разломљеног дела, а могу бити приказани као бројеви са знаком (**signed**) или без знака (**unsigned**), у стандардној или проширеној тачности (**long**). На пример, 0, –1, 109800, или VEZ%, BROJ% у **BASIC**-у, или M, J, у **FORTRAN** -у, 0xFFef, 0Xab05 у **C**–језику, где **x, X** представљају ознаку за хексадекадни број. Целобројни подаци се могу: сабирати (+), одузимати (–), множити (*) и делити (/) и при томе је и резултат такође цео број. Тако је, рецимо: 15/2=7, -7/3=-2, (-39)/(-13)=3, а 3/5=0. У неким програмским језицима дефинисана је и операција **модуо** (**mod**) која као резултат даје остатак при дељењу два цела броја: 5 mod 2 = 1, 39 mod 13 = 0, 8 mod 5 = 3.

Бројеви у фиксном зарезу, тј. **са фиксном тачком** (**real**) имају децималну тачку која се увек мора налазити на истом, фиксном месту. Садрже константе облика -1., 0.008, .51, или променљиве као на пример PIC 99V999 у **COBOL**–у. Иако се зову **real** (реални), ови бројеви су у ствари коначан подскуп скупа рационалних бројева јер имају ограничену величину и тачност, пошто се уписују у меморијске локације коначне дужине.

Бројеви у покретном зарезу, тј. **са покретном тачком** садрже знак, децималну тачку, значајне цифре и експонент. Децимална тачка је покретна, пливајућа, и налази се испред прве, крајње леве цифре, а експонент се коригује да то буде коректан запис броја. Ако је број облика 5555.2, он се у меморију смешта као број .55552 x 10^4, број –0.00000001234 као –.1234 x 10^{-7}.

Неки програмски језици немају посебне декларације за бројеве у фиксном и покретном зарезу. Тако рецимо, **C**–језик има само податак типа **float** (и **double** за двоструко већу тачност). Поред уобичајених аритметичких операција над овим подацима се врше још неке обраде као што су степеновање, нормализација и заокруживање.

Напомена: При извшавању аритметичких операција може се десити да је резултат већи од највећег дозвољеног броја за дату врсту бројева. Таква ситуација се зове прекорачење и као таква она је непожељна, па се мора, ако је то могуће, спречити, а ако се ипак догоди онда се мора детектовати. Шта ће се даље догађати са таквим бројем и програмом зависи од стандарда за тај рачунарски систем. Најчешће се обуставља извршавање таквих програма јер нема смисла обрађивати нетачне податке. Код бројева у покретном зарезу може се још десити да је резултат мањи од најмањег могућег броја за дату тачност (стандардна или проширена). Ова ситуација се зове поткорачење и такође се мора детектовати. Код бројева у покретном

зарезу се такође може десити да је експонент резултата већи од највећег дозвољеног (прекорачење експонента) или мањи од најмањег а да при томе мантиса није једнака нули (истек експонента). Исто тако може доћи и до нарушавања нормалозације, па се и то мора кориговати.

Напомена: У аритметичким изразима се могу комбиновати и бројеви различитог типа и тада рачунарски систем врши аутоматску конверзију једних типова у друге.

Знаковни подаци, алфанумерички подаци или подаци типа **карактер** (**character**) су **ASCII** или **EBCDIC** кодови који представљају слова, цифре или симболе, при чему се сваки симбол меморише у засебан бајт у меморији. Тако се и једна **BCD** цифра смешта у један бајт, иако би била довољна само четири бита. Групе података оваквог типа зову се **ниске** или **стрингови** (**string**), но то су већ сложени типови података. Над знаковним подацима се не могу вршити аритметичке операције.

Логички подаци (**Boolean, Logical**), садрже вредности тачно (**true**) или нетачно (**false**), и могу послужити као добар пример разлике између логичке структуре податка и физичке структуре, тј. начина меморисања податка. Тачно и нетачно су апстрактне величине, а морају имати конкретан приказ у меморији. Вредност тачан (истинит) може бити приказана коришћењем само једног бита, или као група битова, или као нека ненулта вредност. Стварна меморијска структура је потпуно небитна за корисника све док је "**тачно**" тачно и "**нетачно**" нетачно. На константе, променљиве и исказе логичког типа примењују се основне логичке операције: негација (НЕ), конјункција (И), дисјункција (ИЛИ), а у неким рачунарским системима и ексклузивно ИЛИ. У логичким исказима (и изразима) поред логичких операција користе се и релациони оператори: {=, >, <, < >, >=, < =}. Неки програмски језици не препознају овај тип података. Код тих програмских језика улогу ових променљивих преузимају бинарне цифре 0 и 1. Није дозвољено у изразима комбиновати логичке величине са нумеричким.

7.1.2. ПОДАЦИ ЗА КОНТРОЛУ ТОКА ПРОГРАМА

Лабеле, односно обележја су типичан пример ових података. Лабела податак је стварна адреса обележја у меморији. Адреса броја линије FORTRAN или BASIC програма је податак типа лабела, као што се то види из следећег примера:

```
10  PRINT "10 је лабела"
```

Лабелом се идентификује место у програму, односно инструкција на коју се врши гранање, а најчешће безусловни скок. У **C**–језику се безусловни скок остварује наредбом **goto**:

```
            goto порука;
            . . .
порука: printf ("приказ лабеле");
```

Показивачи или **указатељи** (**pointer**), представљају адресу локације у меморији, и најчешће указују на адресу податка или структуре података. Користе се и при преносу података из програма у потпрограм и обратно. Уместо да се преноси читава група података, преноси се само показивач (указатељ) на адресу табеле која садржи адресе података.

7.2. ЈЕДНОСТАВНИ НЕСКАЛАРНИ ПОДАЦИ

Стринг подаци, (**низови карактера**, **ниске)** добијају се спајањем више карактера (знакова), и често се користе као имена, адресе и други текстуални подаци. Неки програмски језици третирају стрингове као елементарне податке, док их други језици посматрају као поља података типа карактер. Стрингови се не могу користити као операнди у аритметичким или логичким операцијама, али се зато над њима могу вршити другачије обраде. У групу основних операција које се могу применити на знаковне податке и низове знакова спадају:

- *копирање алфанумеричког низа (низа знакова),*
- *поређење низова знакова,*
- *налажење задатог знака у неком низу знакова,*
- *налажење првог знака (у неком низу знакова) који је различит од задатог знака,*
- *налажење задатог низа знакова у другом низу знакова,*
- *проналажење знака из задатог скупа знакова у неком низу знакова,*
- *проналажење знака у низу знакова који не припада датом скупу знакова,*
- *копирање низа знакова уз конверзију,*
- *копирање једног низа знакова у други низ почев од неке позиције итд.*

Низови података, **поља** (**array**), представљају скуп појединачних података истог типа. Сви елементи једног поља су или цели бројеви, или бројеви у фиксном зарезу, или карактери. У неким програмским језицима постоје и вишедимензионална поља. Једнодимензионална поља се понекад зову **вектори.** Ако имају две димензије зову се матрице. Матрице су сличне табелама јер имају врсте и колоне, али су овде елементи истог типа.

Поједини елементи могу се лоцирати у пољу уз помоћ индекса. Индекси се, за разлику од оних у математици, пишу унутар малих или средњих заграда

A(2) или **A[5],** што зависи од програмског језика. Индекси могу бити целобројне константе или скаларне варијабле. Употреба променљивих као индекса пружа велику флексибилност у коришћењу поља а програмски код чини компактнијим. Ово посебно важи за оне језике који допуштају употребу аритметичких израза као индекса.

Поља имају неке посебне особине. Прва је број димензија. Неки језици допуштају постојање само једне или две димензије, неки седам (FORTRAN 77), док неки други у том погледу немају никаквих ограничења. Свака димензија има горњу и доњу границу за вредност индекса.

Друга карактеристика се односи на начин складиштења елемената поља. Већина језика памти дводимензионална поља или **врста по врста** (**COBOL**), или **колона по колона** (**FORTRAN**), а простор је резервисан (заузет) за сваки елемент, без обзира да ли на том месту постоји податак који се користи у текућој обради или не. Неки популарни апликациони програми као што су некада били **Lotus 1-2-3, Symphony**, користе посебне технике додељивања простора само оним елементима који стварно садрже податке.

7.3. СТРУКТУРЕ ПОДАТАКА

Без обзира на број димензија код низова, сви елементи поља садрже податке истог типа. Пошто је ово значајно ограничиње, неки програмски језици користе сложене форме података, такозване структуре података, да изврше агрегацију података различитог типа.

Структура је колекција једне или више променљивих (варијабли), које могу бити и различитих типова, груписаних заједно под једним именом ради лакшег руковања. Структуре података омогућавају да група "сродних" варијабли буде третирана као једна јединица, уместо као одвојени објекти.

COBOL и **Pascal** подржавају структуре података типа записа (**record**), док програмски језик **C** подржава сложене податке типа структура (**struct**). Посматрајмо следећи запис у **C**–у:

```
struct датум {
                int дан;
                int месец;
                int година;
                char име_месец [4];
                }
```

Структура **датум** садржи и податак **име_месец** типа низ карактера, и три податка **дан**, **месец** и **година** типа цео број. Сва поља у структури имају своје име и тип податка који садрже. Сваком саставном делу структуре може се приступити као независном податку ако напишемо **датум.дан**, или

датум.име_месец. Ако референцирамо **датум**, онда приступамо свим деловима структуре истовремено.

Структуре је могуће користити као елементе других структура, или као елементе низа података, и тада добијамо поље структура. Ако искористимо претходно дефинисану структуру **датум** као елемент структуре **особа** (сваки запис се односи на једног од 1000 запослених у једном предузећу), онда добијамо поље структура **радник**:

```
struct особа {
                char име[20];
                char адреса[30];
                long мат_број;
                double плата;
                struct датум рођења;
                struct датум запослење;
                }
struct особа радник[1000];
```

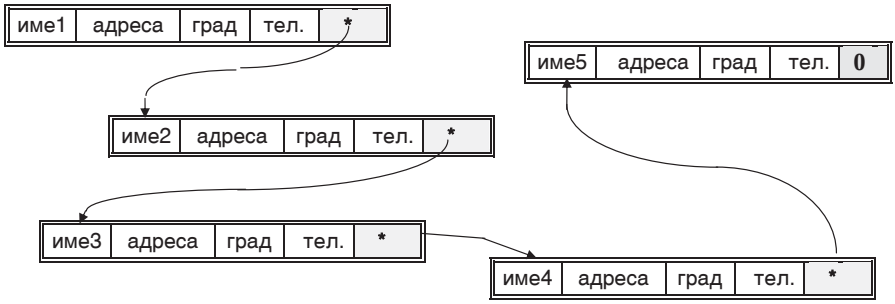
Сваки запис у структури **особа** садржи у себи два податка типа карактер које чине низови: **име** дужине 20 знакова (чије пуно име је **радник.име**) и **адреса** са 30 знакова (**радник.адреса**), затим целобројни податак двоструке дужине **мат_број** за матични број, број у покретном зарезу двоструке тачности са именом **плата** и два сложена податка типа датум: **рођење** и **запослење**. Структура **радник** састоји се од 1000 записа типа **особа**.

7.3.1. СПРЕГНУТЕ ЛИСТЕ

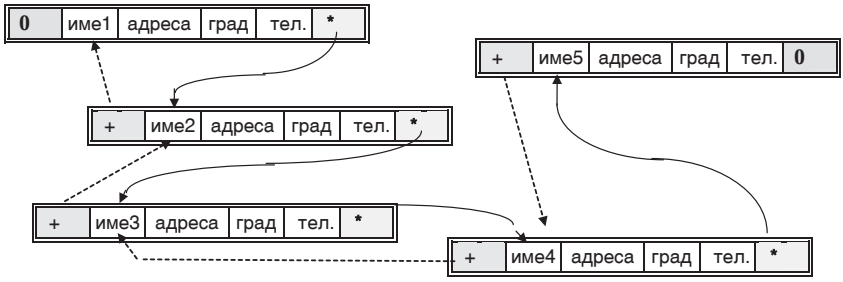
Листе су потпуно уређене структуре података, у којима изузев првог и последњег, сваки елемент има претходника и следбеника. Листе могу бити линеарне (слика 7.2. а)) и нелинеарне (типа стабла). Линеарне листе се могу реализовати као:

- *секвенцијалне (у суседним локацијама) и*
- *повезане помоћу показивача као спрегнуте листе.*

Спрегнуте листе су форма повезаних записа у којима је најмање један саставни део (подјединица), показивач (**pointer**). Тај показивач или показује на локацију следећег записа у листи, или има специјалну вредност **нула**. Нула показивач показује да више нема записа у листи, тј. да је тај запис последњи. Слика 7.2. приказује листу записа који садрже следеће подјединице, односно поља: **име**, **адресу**, **град** и информацију о броју телефона (**тел**), и, наравно, **показивач.** Уочимо да записи, сами за себе, нису у неком поретку, али пратећи показивач добијамо један поредак: **име1, име2, име3** итд.



а) Једноструко повезана линеарна листа



б) Двоструко повезана листа

Слика 7.2. Линеарне повезане листе

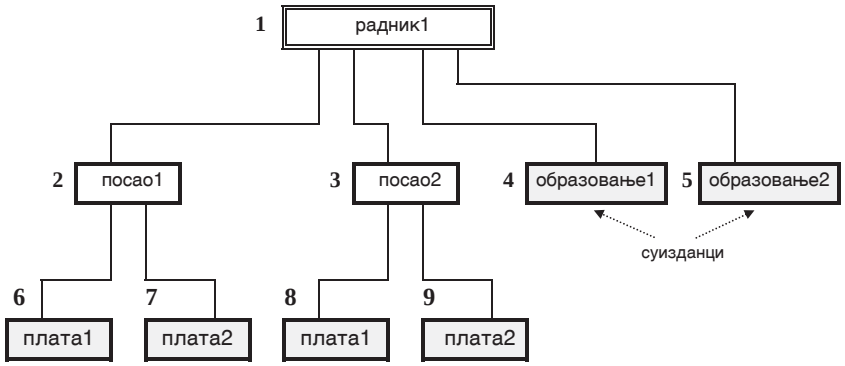
Листе са једним показивачем зову се једноструко повезане листе, слика 7.2 а). На слици 7.2 б) приказана је нешто сложеније спрегнута листа структура. Свака структура у овој листи повезана је са две друге. Пратећи показивач у десном смеру добијамо исти низ као и пре. Пратећи показивач улево пролазимо кроз листу у супротном смеру. Када сваки елемент листе показује на претходни и следећи елемент, кажемо да је то двоструко повезана листа. Спрегнуте линеарне листе могу бити и вишедимензионалне. Оне се састоје од више спрегнутих листа. Време претраживања линеарне листе је доста дуго и зависи од положаја елемента у листи што је понекад врло неповољно за коришћење.

7.4. СЛОЖЕНЕ СТРУКТУРЕ ПОДАТАКА

Скаларни подаци се могу груписати у структуре, а записи у структури се могу повезивати међу собом, формирајући повезане листе. Повезане листе се често уграђују у друге структуре података. Три уобичајене, и врло важне, структуре података овог типа су: **стабла** (**tree**), **магацини** (**stack**) и **редови** (**queue**). Све ове структуре су изузетно важне у системима за управљање базама података и у оперативним системима.

7.4.1. СТАБЛА

Стабла су разгранате, нелинеарне листе. На слици 7.3 приказана је општа структура типа стабла. Сваки правоугаоник у дијаграму зове се **чвор**, и представља потенцијалну тачку гранања. Пуне линије су релације (везе) између чворова и зову се **гране**. Чвор на врху назива се **корен** (дрво је окренуто наопако, одозго на доле). Завршни чворови, који испод себе немају следбеника, зову се крајеви стабла или **листови**.



Слика 7.3. Стабло које садржи информације о радницима у једном предузећу

Сваки чвор, изузев корена, спојен је са само једним чвором који је за један ниво виши од њега, и зове се претходник. Чвор 1 је претходник чворова 2, 3, 4 и 5, док је чвор 2 претходник чворова 6, и 7. Сви поменути чворови су спојени са чворовима нижег нивоа који се зову следбеници, а при томе су чворови 4, 5, 6, 7, 8 и 9 истовремено и листови стабла.

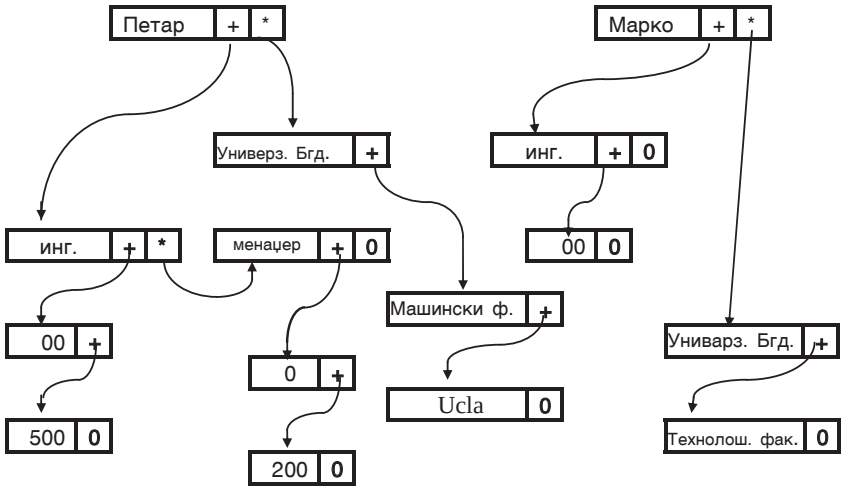
Чворови обично садрже структуре података, а често садрже записе. Подаци на једном нивоу не морају обавезно бити структуре истог типа. Уопштена структура стабла података назива се шема (у терминологији база података), и она садржи имена записа и везе које постоје међу њима.

Стабла се најчешће реализују помоћу повезаних листа. Ове листе могу комбиновати записе у датотекама или структуре података смештене на произвољан начин у меморији.

На слици 7.3 приказано је стабло које садржи информације о запосленима у једном предузећу. У корену су подаци о радницима, а они се могу односити на: име и презиме, адресу, матични број и друге непроменљиве податке. На другом нивоу су подаци о пословима и школској спреми (обрзовању и усавршавању). На трећем нивоу су подаци о платама, итд., што значи да број нивоа није ограничен.

Као што се види из овог примера, претпоставља се да један запослени може радити на више послова, а сваки посао може имати више платних спискова који су му придружени, тј. са којима је повезан. Међутим, сваки платни списак је повезан са само једним послом. Значи да у овој структури типа стабла један чвор може имати више следбеника, али само једног претходника.

На слици 7.4. приказана је повезана листа која реализује ову структуру типа стабла.



Слика 7.4. Повезана листа која релизује стабло структуре радника

Сваки запис о запосленом садржи два поља показивача. Један показује на први запис о послу за тог запосленог, а други на први запис који се односи на школску спрему.
Записи о пословима такође имају два показивача. Први показује на додатне записе о послу, и други који показује први запис плате за текући посао. Записи о плати имају само један показивач који показује на следећи запис о плати, који припада текућем запису о послу. Записи о стручном усавршавању имају, такође, само један показивач којим су повезани записи на том нивоу.

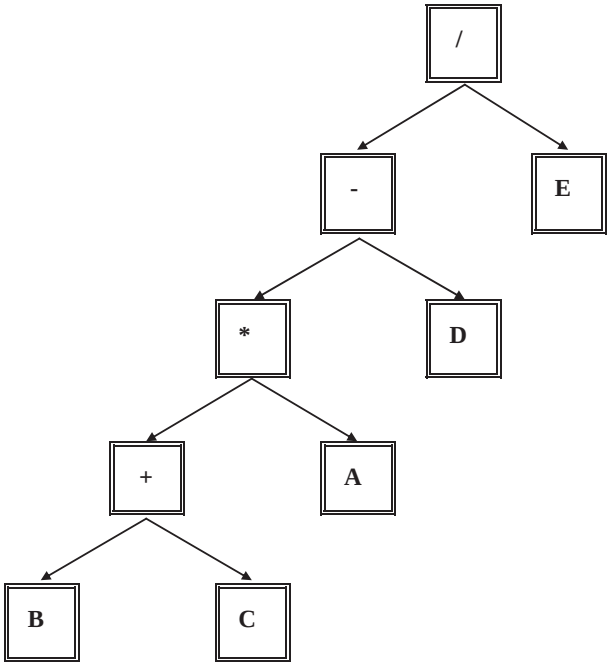
Бинарно стабло је тип стабла у којем сваки чвор има највише два подстабла. Стога бинарно стабло можемо дефинисати као коначан скуп чворова, који може бити празан или се састоји од корена и два дисјунктна бинарна стабла, названи лево и десно подстабло. Структуре података типа бинарно стабло имају одређених предности и често се користе.

Структуре типа стабла су широко распрострањене у оперативним системима, **системима за управљање подацима** (**Data Base Management Systems, DBMS**) и у апликацијама за обраду података. Пример приказан на сликама 7.3. и 7.4. показује како се стабла могу користити за представљање веза између различитих врста података. Веза између запосленог и посла је пословна веза, док је веза између запосленог и стручне спреме образовна релација.

Стабла се неизоставно користе у **преводиоцима** (**compiler**), када они конвертују кориснички програм у машински језик. Током процеса превођења реченице, линије изворног програма се разбијају на њихове појединачне саставне делове и проверавају да ли су написане у складу са правилима односно, синтаксом програмског језика. Елементарни саставни делови (речи) и везе међу њима (операције), такође се приказују структуром типа стабла и то је синтаксно стабло или синтаксно дрво.

На слици 7.5 приказано је једно синтаксно дрво којим се реализује израз:

(A *(B+C) - D) / E .



Слика 7.5. Синтаксно стабло за израз **(A *(B+C) - D) / E**

У рачунару се изрази израчунавају по следећим правилима. Најпре се израчунавају изрази у заградама; ако редослед операција није одређен

заградама прво се врши * и /, а затим + и –. Коректно извођење овог исказа (реченице) захтева следећи низ операција. Прво се саберу **C** и **B**, а затим се збир помножи са **A.** Онда се од претходно добијеног тезултата одузима **D** и на крају се добијена разлика подели са **E.** Овај количник представља коначни резултат.

При извођењу израза приказаног структурним стаблом на слици 7.5, креће се од корена. Пре обраде ма ког податка иде се увек крајње левом граном, што је даље могуће. Када се нађе крајње леви лист (**B**), одређује се вредност једне варијабиле, затим се нађе вредност прве суседне варијабиле (**C**) са којом **B** има заједничког претходника, тј. чвор који показује на аритметичку операцију која ће бити извршена. У сваком случају акција на коју показује чвор биће извршена после (**post**) обраде фактора. Ово је такозвани **post order** анализатор. Следећи ову шему, добијамо коначни поредак обраде: **B, C, +, A, *, D, -, E , /.** Овакав поредак лако можемо препознати на многим џепним и научним калкулаторима.

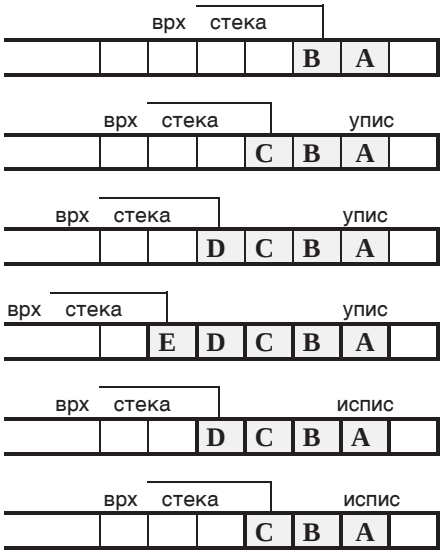
Постоји много операција које се могу дефинисати над стаблима. Једна од најважнијих је обилазак стабла, при чему се сваки чвор обилази једанпут и само једанпут. Могу се дефинисати три начина обиласка стабла у зависности од редоследа обиласка корена, левог и десног подстабла:

1. ***In order*** *редослед, обилази се: најпре лево подстабло, затим корен и на крају десно стабло,*
2. ***Pre order*** *редослед, обилази се редом: корен, лево подстабло, десно подстабло,*
3. ***Post order*** *редослед, при којем се обилази: лево подстабло, десно подстабло, корен.*

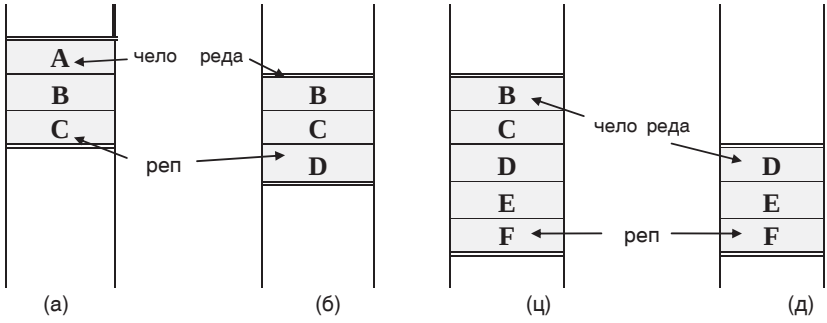
7.4.2. МАГАЦИНИ И РЕДОВИ

Магацини (**stack**) и редови (**queue**) су структуре података које су врло значајне за рад рачунара и организацију оперативних система. Као што је већ речено магацини или стекови се користе за држање података којима се мора приступити у **LIFO** поретку, (**последњи уписани први исписан**, **Last In First Out**). Подаци се увек уписују и исписују са истог краја стека. Стекови се најчешће користе за држање повратних адреса из потпрограма. Могу се такође користити за привремено држање података и за пренос података у потпрограме. На слици 7.6. дат је пример стека као **LIFO** структуре података. Структуре података са приступом **први уписан, први исписан** (**FIFO, First In First Out**), користе се за реализацију редова. На слици 7.7. (а) приказан је ред са три члана. Структура **A** је на челу реда, **B** долази следећи, а **C** је последњи додати члан. На слици (б) је приказано

стање реда након што је први члан реда, структура **A** напустила ред, и додата је структура **D**. На дијагарму (ц) је приказан ред након додавања још два члана **E** и **F** у структуру. У делу (д) је дато стање реда након узимања два члана **B** и **C** са чела реда.



Слика 7.6. Магацини су структуре типа последњи улази први излази



Слика 7.7. Редови су структуре података типа први улази први излази

Операције за рад са магацинима су следеће:

*1. читање без брисања врха стека (**read**),*

*2. упис у магацин са врха (**push**),*

*3. испис из магацина (читање са брисањем врха стека) (**pop**).*

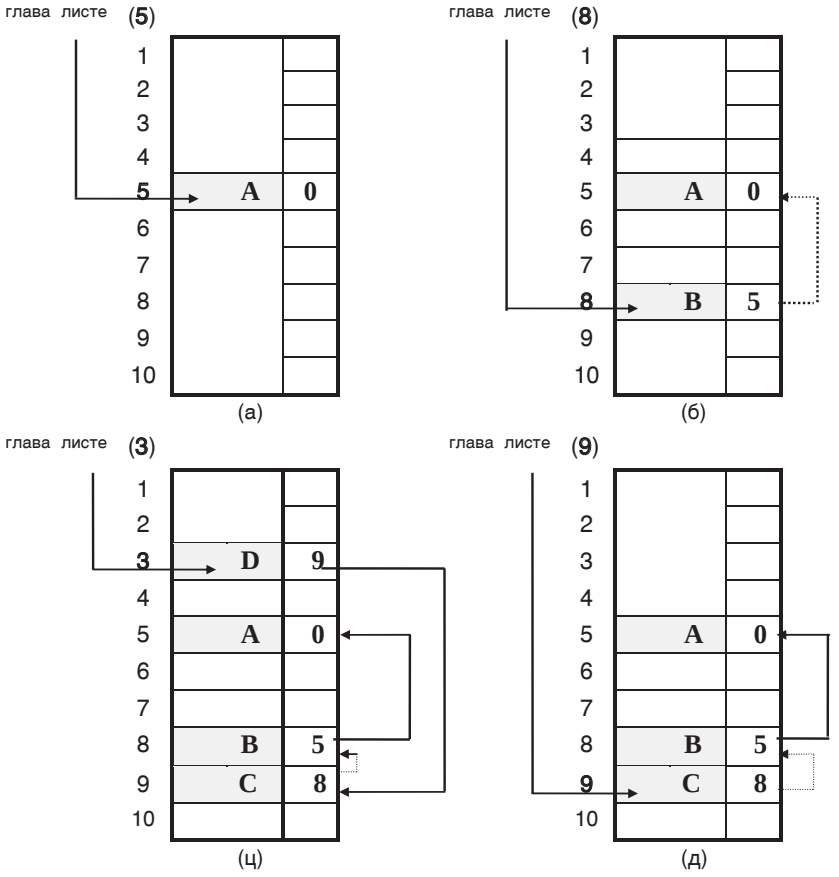
Операције са редовима су следеће:

1. уношење у ред на страни репа,
2. брисање из реда са стране чела,
3. читање првог елемента на страни чела.

Сваки елемент магацина и реда, сем показивача на следећи елемент, садржи и корисне податке. Тип података може бити произвољан.

7.4.2.1 Реализација магацина и редова

Магацини и редови се најчешће реализују помоћу повезаних листа. На слици 7.8 је дат приказ једне повезане листе којом се реализује магацин.



Слика 7.8. Реализација магацина помоћу повезане листе

Дијаграм (а) показује да магацин–стек, чија је **глава 5**, садржи запис **A**, који је смештен на позицију **5** у примарној или секундарној меморији. Специјална локација која се зове **глава листе** садржи показивач на врх стека, (слично као регистар **SP** – показивач стека). Дијаграм (б) показује стек после додавања (уписа) записа **B,** смештеног на локацији **8**. Додавање овог записа укључује следеће кораке:

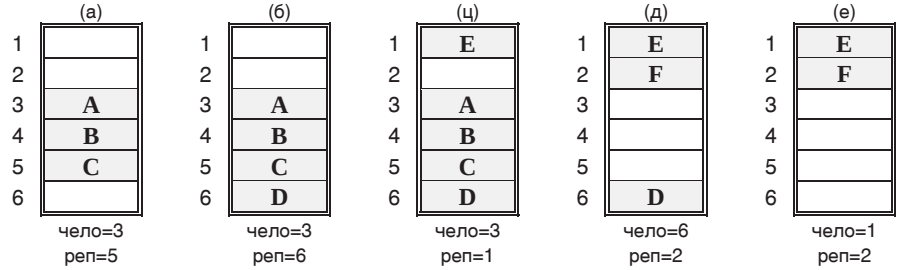
1. *налажење слободног места за смештање записа (нпр. позиција **8**),*
2. *копирање текуће главе листе у поље показивача новог записа. То је показивач чија је вредност **5**, и*
3. *постављање показивача у главу листе на нову локацију која одговара новом запису. То је показивач на позицију **8**.*

Уочимо да повезана листа допушта записима да се налазе у логичном поретку, а да при томе нису у физичком поретку. То се још јасније види са дијаграма (с), где су додата још два записа. Описана процедура се примењује и при упису записа **C** на позицију **9** и записа **D** на позицију **3**.

Узимање (испис) последњег записаног податка **D** (структура или запис) из стека проводи се у два корака.

1. *Приступа се запису на који указује глава листе (локација **3**).*
2. *У главу листе се копира показивач (вредност **9**) из записа коме се управо приступило (запис са локације **3**).*

Редови се могу имплементирати на сличан начина као стек, или пак користећи суседне меморијске локације. Овде, међутим, постоји ограничење колико се меморије може користити за листу. Не може се допустити раст листе без ограничења, па се зато често користе **циклични редови** (**circular queues**), као на слици 7.9**.**



Слика 7.9. Циклични ред смањује потребан меморијски простор за држање реда

Део (а) показује ред са шест елемената у који су уписане три структуре. А је најстарија, а **C** најмлађа. Структура **D** се може додати (уписати) на крај реда као што се види на дијаграму (б), Додавање структуре **E** изазива проблеме, јер нема више слободних места после позиције 6. Међутим,

принцип рада цикличног реда подразумева да за последњом позицијом логички следи прва позиција, дакле **E** се уписује на позицију 1, слика 7.9 (c). Део (d), показује ред после узимања (исписа) три најстарије улазне структуре и додавања структуре **F**. Исписивање структуре **D**, на слици 7.9 (d), опет би изазвало излазак из допуштене меморије и циклирање на позицију 1, као што је показано на слици (e).

Уочимо да употреба цикличних редова захтева држање два показивача, један за чело (или главу реда), и други за крај односно реп реда. Показивач за крај реда се користи када се додаје нови податак, а показивач на чело реда током исписивања. Алгоритам мора коректно да израчунава цикличне адресе, а мора обезбедити детекцију како пуног, тако и празног реда.

У случају празног реда показивачи за главу и реп се постављају на нулу (**0**). Повезане листе се често користе за прављење редова у оперативним системима, који садрже податке о програмима који су у стању извођења, и о програмима који су у стању чекања за коришћење U/I јединица. Спрегнуте листе имају одређене предности које допуштају програмским информацијама (деловима кода и подацима), у разним деловима меморије, да буду логички спојене, без физичког померања или реорганизације меморије.

Напомена: **Алгоритам** је тачно одређени и уређени скуп корака који воде ка решењу датог проблема.

7.5. КЛАСЕ МЕМОРИЈА

Сви подаци (инструкције, бројеви, стрингови, поља итд.) се састоје из битова, али памћење по правилу подразумева групе битова. Примарна меморија подржава податке величине бајта или речи, док секундарне меморије користе веће целине као што су сектори или записи.

Концепт класа меморија, повезан је са **алокацијом** (доделом) меморијског простора. Код неких рачунарских система додела меморијског простора обавља се једном и више се не може мењати. То је такозвана **статичка додела меморије** (**static allocation memory**). У овом систему се, на пример, датотеци на диску, приликом креирања, одмах додели максималан број сектора, који ће та датотека икада моћи да има.

Неки програмски језици и оперативни системи додељују примарну меморију програмима на основу потреба. Када програму треба простор за привремено смештање поља, он то захтева од оперативног система. Када програму то поље више није неопходно, или је обављена обрада, он враћа тај део меморије оперативном систему. Ово је такозвана **динамичка додела меморије** (**dynamic allocation memory**).

Датотекама на дисковима се нормално додељује само неколико сектора када се датотека креира, али јој се додељује додатни простор динамички, када им то затреба. Ово знатно смањује расипање меморијског простора у систему, који би се морао додељивати у максималној величини, чак и онда када се користи изузетно мали део.

7.5.1. ОРГАНИЗАЦИЈА МЕМОРИЈЕ

Подаци се у меморију не смештају појединачно, већ се претходно, на различите начине, организују. Један од приступа који се зове **узастопни смештај** (**consecutive storage**) слика 7.10., смешта податке у суседне локације, један за другим, без прекида.

локације	**садржај**
1B300	A
1B301	B
1B302	C
1B303	D
1B304	E
1B305	F
1B306	G
1B307	H
1B308	J
1B309	K

Слика 7.10. Смештање података у суседне локације

Овај приступ је типичан за програмски језик FORTRAN, а на слици 7.10 приказано је смештање групе од 10 података који чине једно поље са почетном локацијом $\mathbf{1B300}_{16}$. На овај начин се често смештају низови и записи који се записују на магнетне траке.

Повезано смештање (повезане листе) је други значајан начин смештања података. Код овог типа за смештање низа података није неопходан довољно велики континуирани меморијски простор, јер свака јединица која се меморише (елемент, запис) садржи и указатељ – показивач на следећу јединицу. На слици 7.11. је приказано смештање истог низа као повезане листе. Уочимо да смо прећутно претпоставили да свака меморијска позиција може да прихвати и податак и показивач.

Први елемент је на локацији 1B300, сам податак је структура **A**, а поље показивача указује да је следећи податак на локацији 1B304. На локацији 1B304 записан је податак **B**, а показивач казује да је следећи податак на локацији 1B305. Следећи податак је на 1B309, итд.

Последњи податак (чија је вредност **K**) налази се на локацији 1B308 и има показивач **00000**, који служи као **нула показивач,** а задатак му је да маркира крај листе.

локација	**садржај**	**показивач**
1B300	**A**	1B304
1B301	**J**	1B308
1B302	**F**	1B307
1B303		
1B304	**B**	1B305
1B305	**C**	1B309
1B306		
1B307	**G**	1B310
1B308	**K**	00000
1B309	**D**	1B30D
1B30A		
1B30B		
1B30C		
1B30D	**E**	1B302
1B30E		
1B30F		
1B310	**H**	1B301

Слика 7.11. Памћење података у облику повезане меморије

Трећи тип смештања података базиран је на коришћењу кључева. Кључ може бити утиснут у податак и касније, када је то потребно, издвојен из њега. Кључ може указивати на тачну локацију податка у меморији (апсолутна меморијска адреса или на пример, број цилиндра, главе и сектора на диску) или може означавати релативну позицију (на пример, 52. запис у датотеци).

Кључеви могу представљати директно адресу на којој је податак смештен или се могу трансформисати у адресу неком од техника (индексирање, математичке трансформације, итд.).

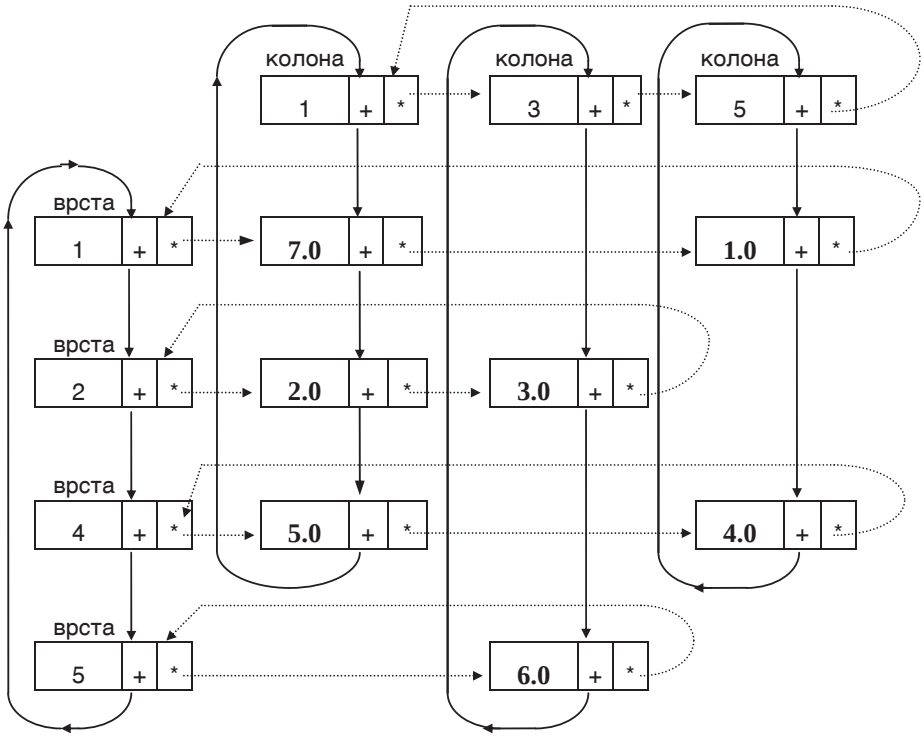
Четврти начин је смештање података у блоковима (странице, сектори, итд.). Сваки блок је лоциран помоћу кључа или показивача, док су унутар блока подаци смештени у суседне локације, један за другим.

Велики низови често садрже велике проценте неискоришћених, или нула елемената. Ако, рецимо, имамо низ са 100_{10} врста и 100_{10} колона, а само 50_{10} их је у текућој употреби, за меморисање читавог низа треба нам 10000_{10} локација. Ово би било енормно расипање меморијског простора, а проблем се решава концептом ретких (прорећених) низова. Концепт ретких низова додељује простор само елементима који садрже податке, а остале елементе занемарује.

Овакав приступ се може имплементирати такође уз помоћ повезаних листа, као што је то приказано на слици 7.12. Овде је за сваку врсту и сваку колону која има ненулте чланове, употребљена по једна засебна повезана листа. Свака листа по врсти и колони у почетном елементу, глави листе, садржи редни број врсте или колоне којој је придружена, и још два показивача. Један показивач повезује главе листа по врсти, а други повезује елементе те врсте. Исто тако су повезане и колоне. Сваки елемент низа је структура која садржи оригиналну вредност елемента (тј. податке) и два показивача.

једноставна матрица

7.0	**0.0**	**0.0**	**0.0**	**1.0**
2.0	**0.0**	**3.0**	**0.0**	**0.0**
0.0	**0.0**	**0.0**	**0.0**	**0.0**
5.0	**0.0**	**0.0**	**0.0**	**4.0**
0.0	**0.0**	**6.0**	**0.0**	**0.0**



Слика 7.12. Имплементација прорећених низова помоћу повезаних листи

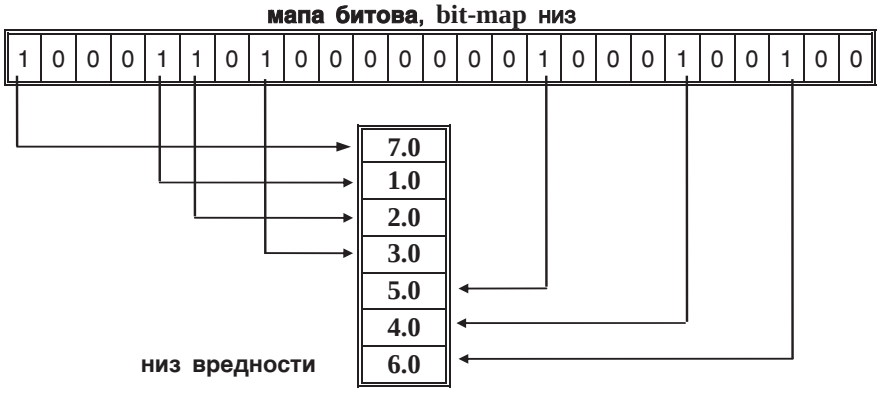
Посматрајмо главе повезаних листа по врсти. Леви показивач повезује главе листа за 1. врсту, са оном за 2. врсту, 4. врсту и 5. врсту. Глава листе за

било коју врсту се може лоцирати праћењем овог показивача. Десни показивач води до елемената. Тако, на пример, десни показивач у глави листе за другу врсту указује на елемент чија је вредност 2, а његов десни показивач на елемент чија је вредност 3. Овај елемент је последњи у овој листи, јер у 2. врсти више нема ненултих елемената, и његов десни показивач показује на главу листе за 2. врсту.

Сваки елемент низа садржи, поред вредности елемента, и два показивача. Леви показивач повезује тај елемент листе (један запис) са следећим елементом (записом) у тој истој колони, а десни показивач га повезује са следећим елементом те врсте.

Посматрајмо, на пример, 4. врсту. Пратећи десни показивач, почев од главе листе, долазимо до елемента који садрже вредност **5.0** и **4.0**. Којој колони припада елемент који садржи вредност **4.0**? Да бисмо добили одговор на то питање, морамо следити везу дуж колоне, тј. леви показивач све док не дођемо до главе листе. Када то урадимо сазнајемо да тај елемент припада 5. колони. Уочимо да је елемент који садржи вредност **4.0** у 4. врсти и 5. колони, и да је то последњи члан у листи пете колоне (и четврте врсте). Уочимо, такође, да показивач колона (и врста) уместо вредности нула, повезује задњи члан са главом листе. Ово су такозване цикличне листе (**circular lists**).

једноставан низ					мапа битова				
7.0	**0.0**	**0.0**	**0.0**	**1.0**	**1**	**0**	**0**	**0**	**1**
2.0	**0.0**	**3.0**	**0.0**	**0.0**	**1**	**0**	**1**	**0**	**0**
0.0	**0.0**	**0.0**	**0.0**	**0.0**	**0**	**0**	**0**	**0**	**0**
5.0	**0.0**	**0.0**	**0.0**	**4.0**	**1**	**0**	**0**	**0**	**1**
0.0	**0.0**	**6.0**	**0.0**	**0.0**	**0**	**0**	**1**	**0**	**0**



Слика 7.13. **Ретки низови** реализовани помоћу низа **bit-map** и низа вредности

Могућ је и другачији приступ проблему ретких низова, то је такозвана битна мапа (**bit map)**, приказана на слици 7.13. **Bit map** је структура података која садржи само један бит за сваки могући елемент низа. Ако је садржај бита нула онда он указује на празан елемент; када је садржај неког бита јединица, онда тај бит указује на елемент који садржи податак који је различит од нуле. Уз **bit-map** низ, иде и једнодимензионални низ (вектор) који садржи податке, тзв. **низ података.** Првом елементу у низу података одговара прва јединица у **bit-map** низу. Друга јединица у мапи битова одговара другом елементу у једнодимензионалном пољу, итд.. Проналажење жељеног члана низа података захтева испитивање **мапе битова** да би се одредила његова локације у низу, и да би му се затим приступило.

7.6. ПРИСТУП МЕМОРИЈИ

Технике које се користе при приступу меморији, тј. подацима не морају нужно зависити од метода које су коришћене при физичкој организацији података. Користе се две основне технике приступа, а то су: секвенцијални и директан, односно случајан приступ.

7.6.1. СЕКВЕНЦИЈАЛНИ ПРИСТУП

Секвенцијални приступ подацима (који су често смештени у суседне локације), подразумева читање једног елемента након неког претходног. Повезаним листама се приступа секвенцијално, пратећи показивач са једног члана на други члан. Неке технике памћења, засноване на примени кључа, користе се за имплементацију индексираних секвенцијалних датотека и оне допуштају секвенцијални приступ подацима у поретку кључа.

Секвенцијални приступ захтева да се са претраживањем крене од почетног члана, а да би се дошло до петог члана, мора се, најпре, приступити 1., 2., 3. и 4. члану. Отуда следи да је време оваквог приступа зависно од редног броја члана, и утолико је дуже што је члан даље од почетка. Ова метода приступа је погодна када се подаци обрађују оним редоследом којим су смештени, јер се тада увек након обраде једног члана прелази на обраду следећег члана, којем се одмах може приступити.

Секвенцијалне датотеке се могу обрађивати као неуређене или несортиране, и као уређене или сортиране. Када се формира датотека, редослед уписивања слогова је одређен хронолошким редоследом њиховог настанка и уношења. Тако се добија неуређена датотека. Неуређене секвенцијалне датотеке у већини случајева нису погодне за обраду, па је обично једна од првих обрада секвенцијалне датотеке након формирања њено сортирање.

Најчешће се за обраду секвенцијалних датотека користе магнетне траке. Међутим, секвенцијалне датотеке се могу реализовати и на свим меморијским медијумима код којих су могући разни други облици приступа (индиректни, полудиректни и директни).

Подаци на траци се не могу адресирати, па је могућ само индиректни приступ. Предност секвенцијалних датотека на магнетним тракама је у њихова једноставност и економичност, јер су траке најјефтинији магнетни медијум којим се постиже добра искоришћеност меморијског простора.

Када се резултати обраде добијају по истом редоследу као што је редослед података у датотеци (на пример, при обради плата), програм најчешће тражи баш наредни слог, тако да тражење не мора почињати од почетка. Овако се знатно смањује време приступа, па је за многе послове секвенцијална организација врло погодна и економична.

Најкраће време приступа би се постигло када се при обради не би прескакао ни један слог. Сортиране секвенцијалне датотеке на магнетним тракама често се користе у пакетно оријентисаној обради података, када се слогови обрађују увек на исти начин, када обрада није честа, и када је број промена велики тако да се обрађује претежан број слогова у датотеци.

7.6.2. ДИРЕКТАН ПРИСТУП

Методе директног или случајног приступа, омогућавају лоцирање појединачних чланова, без секвенцијалног претраживања од почетка меморије. Могуће је, на пример, директно приступити 19. елементу низа, чак и ако је низ у меморији смештен у сукцесивне локације.

Директан приступ подразумева да су подаци смештени у познате позиције на медијумима. У свакој позицији региструје се по један слог. Сваки слог има адресу која једнозначно одређује његову локацију. Физички редослед смештања слогова у локације није важан и слог коме се приступа бира се на случајан начин. Слог се проналази преко адресе локације, а за одређивање локације користи се **кључ.** Да би се приступило траженом слогу мора се успоставити једнозначна кореспонденција између кључа траженог слога и адресе локације на меморијском медијуму.

Индиректан приступ омогућује налажење слогова на медијумима и у локацијама које се не могу адресирати. Да би се приступило жељеном слогу, врши се претраживање датотеке.

Поље, тј. део структуре по којем се врши претраживање зове се **аргумент претраживања**, или **кључ.** Претраживање се може вршити према једном, примарном кључу, или по једном или више секундарних кључева.

Претраживање по примарном кључу се завршава када се пронађе слог чији је кључ једнак вредности аргумента претраживања. Као резултат успешног претраживања може се добити садржај само једног слога. На пример, у датотеци за обраду плата, примарни кључ је матични број радника и помоћу њега се може приступити слогу сваког конкретног радника. Ако се такав слог не нађе, претраживање је неуспешно.

Када се претраживање врши на основу вредности неког секундарног кључа као резултат се може добити садржај већег броја слогова. Ако се, на пример, претраживање врши према стручној спреми радника, приступиће се слоговима свих радника који имају тражену спрему.

Индиректан приступ је ефикаснији код уређених него код неуређених структура. Уређивање структура се врши на основу вредности неког кључа, а према релацији поретка (на пример, у опадајућем–**descendent** или у растућем–**ascendent** поретку). За разлику од неуређене структуре, код које се у случају неуспелог претраживања претражује цела датотека, код уређене структуре претраживање се врши само док вредност аргумента не пређе вредност кључа (према задатој релацији поретка). Ако се до тада не нађе тражени слог, претраживање је безуспешно и не треба га даље настављати. Индиректан приступ може се вршити помоћу показивача (код повезаних структура) или помоћу индекса, који су уређени на бази кључева.

Полудиректан приступ омогућава да се групи података може приступати директно, а конкретном слогу унутар групе претраживањем. На пример, директно се може приступити свакој стази на магнетном диску (али не и сектору или појединим слоговима). Поступци приступа се могу и комбиновати.

Ако је структура уређена и ако је могућ директан приступ слоговима, може се применити такозвано **бинарно претраживање.**

Структура се дели приближно на две половине и процес претраживања започиње од средине, тако што се аргумент претраживања упоређује са кључем средњег слога. Ако су вредности кључа и аргумента исте, слог је пронађен. Ако нису једнаке, одбацује се једна половина структуре, и то она у којој су све вредности кључа веће или све мање од вредности аргумента. Друга половина, код које су неки кључеви већи, а неки мањи од аргумента претраживања, поново се дели. Процес претраживања се прекида када се нађе задати слог, или када су обе добијене "половине" такве да аргумент претраживања није у границама могућих вредности кључева, и тада је претраживање неуспешно. Бинарно претраживање је логаритамско по ефикасности, тј. просечно време потребно да се пронађе жељени податак је: $\mathbf{t}=\log_2 \mathbf{n}$, где је **n** број елемената структуре (нпр. базе података). Ако се број слогова (записа) у једној бази података повећа са 10000 на 20000 просечно време приступа повећаће се само за око 7,5%. Овај поступак врло често примењујемо, а да тога нисмо свесни. На пример, када тражимо неку реч у речнику ми отворимо речник негде око средине. Ако се деси да је реч управо ту добро је. Ако није, онда гледамо да ли је у првој или другој

половини. Онда ту половину половимо, и тако постепено смањујемо подручје где се налази тражена реч на 1/4, па на 1/8, 1/16 речника . . ., све док не остане само једна страница.

7.6.3. ИНДЕКСИРАЊЕ

Случајан приступ није, нормално, на располагању код спрегнутих метода памћења, јер нема начина да се пронађе појединачни члан, изузев праћењем показивача. Међутим, копије поља кључа и показивача, могу бити смештене у посебне структуре, формирајући такозване индексе (**index**), који могу послужити за директан приступ. Индекси се још зову и инвертоване листе (**inverted list**). Једна табела која садржи низ кључева (**име**) и показивача, дата је на слици 7.14.

име	*цилиндар*	*глава*	*сектор*
Перо	120	5	15
Марко	098	0	10
Јанко	003	3	8
Петар	075	1	5
Милан	021	7	3
Соња	086	2	7
Слобо	086	6	12

Слика 7.14. Индекс који садржи поље кључа (име) омогућава директан приступ информацијама (**cylinder, head, sector**).

Уочимо да су записи смештени на врло различитим (несуседним) локацијама, па је секвенцијалан приступ немогућ или у најмању руку није сврсисходан. Приступ појединачним записима је, с друге стране, и лак и брз. Обзиром да је табела индекса мала у односу на стварну количину података у записима, држање индекса у редоследу смештања података је могуће и олакшава идентификацију појединачних кључева.

Друга предност индекса је могућност приступа записима података у редоследу кључева. Рутине се могу пројектовати тако да лоцирају први кључ у индексу, користећи га као показивач за приступање подацима, и затим поновити поступак за следећи кључ. Индекси се могу реализовати на магнетним дисковима и у оперативној меморији. На диску се слогу може директно приступити када се знају адресе цилиндра, главе и позиција слога на стази–трагу. У том случају индекси слогова садрже редни број: цилиндра (стазе), главе и сектора на стази.

Индексна организација датотека се користи када је време приступа критично, а датотеке нису велике. Коришћењем више индекса –

мултииндексирањем, може се у једном претраживању добити више информација без реорганизације или вишеструког чувања (редундантности) података у датотеци. На пример, при резервацији авионских карата, од значаја су разне информације који се односе на: број лета, одредиште, датум, име путника и слично.

7.6.4. МЕТОД ОСТАТКА ДЕЉЕЊА

Директан приступ подразумева да су подаци смештени у познате позиције на медијуму. У свакој позицији може да се региструје по један слог. Сваки слог има адресу која једнозначно одређује његову локацију у меморији и преко које се може директно пронаћи. Физички редослед смештања слогова у локације није од значаја. Након приступа једном слогу може се приступити било ком другом. Приступ слоговима је брз. При тражењу слога успоставља се веза између вредности кључа сваког слога са адресом локације у којој се налази. Та веза се остварује трансформацијом вредности кључа у адресу.

Коришћење апсолутних адреса је непрактично јер свака промена уређаја на ком се чува датотека, или самог меморијског простора на уређају, захтева промену идентификатора.

Директна трансформација изједначавањем кључа и адресе врло се ретко примењује, а чешће се адресе локација израчунавају на основу вредности кључа. За такве трансформације користи се више метода. Оне се разликују према примењеном алгоритму за израчунавање адресе. Најчешће се користе:

- *метода остатка дељења (уобичајена за генерисање случајних бројева),*
- *метода преклапања,*
- *метода централних цифара квадрата кључа,*
- *метода цифарске анализе и др.*

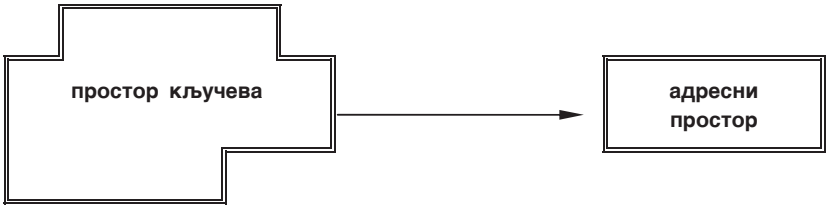
Коришћењем било које од ових метода желе се постићи два циља:

- *што равномернија трансформација кључева и адресе, и*
- *сажимање слогова у датотеци да би се заузело што мање меморијског простора.*

Метод остатка дељења је један од најраспрострањенијих метода трансформације кључа у адресу. Као што се види са слике 7.15., простор кључева (**key space**), односно скуп могућих кључева је знатно већи од потребног броја адреса (**address space**). Одавде следи да овај алгоритам (а то важи и за остале), не може да обезбеди јединствену адресу за сваки

кључ. Односно долази до појаве вишеструке заузетости једне исте локације. Ово нормално није дозвољено па алгоритми морају имати могућност за превазилажење тих проблема. Један алгоритам, који се често користи, решева овај проблем тако што сваком наредном податку који има исти остатак (исту потенцијалну адресу) додељује прву наредну слободну локацију у меморији.

Овај поступак се састоји у дељењу нумеричког кључа са првим бројем који је већи од величине адресног простора. Добијени остатак се памти и представља адресу податка. На пример, нека имамо 520 запослених у једном предузећу, а као кључ користимо матични број грађанина. То значи да нам треба само 520 адреса, па матичне бројеве треба делити са 521, а добијене остатке користити као адресе. Међутим, може се десити да се при дељењу два разна матична броја са 521 појави исти остатак. Тај случај се назива колизија, и мора се превазићи на неки начин. Примера ради, податак чији је кључ број 710168 при дељењу са 521 даје остатак 45, тј. њему се додељује локација у меморији са адресом 45. Међутим, и број 710689, такође, има остатак 45 при дељењу са 521. Како је ова локација већ заузета, онда се податку са кључем 710689 додељује прва наредна слободна адреса, тј. локација 46. Али, ако је локација 46 заузета, онда му се додељује следећа адреса (ако је слободна), тј. локација 47, и тако редом.



Слика 7.15. Однос између простора кључева и адресног простора

Овај се поступак може применити и код кључева који нису нумерички, тако што им се прво одреди нумерички еквивалент. Могуће је на пример, користити нумеричке вредности **ASCII** и **EBCDIC** кодова имена података као нумерички кључ.

7.6.5. МАТЕМАТИЧКА ФОРМУЛА

Математичке формуле се користе за израчунавање адресе жељене локације и директног приступа ускладиштеним подацима. Добар пример за ову методу налази се у техници која се обично користи за смештање вишедимензионалних низова у примарну меморију. Посматрајмо како се у

језику FORTRAN, смешта дводимензионални низ података које зовемо **X,** а садржи податке **data1**, **data2**, ... , **data15** који је декларисан као низ са 5 врста и 3 колоне, слика 7.16.

data1	**data2**	**data3**
data4	**data5**	**data6**
data7	**data8**	**data9**
data10	**data11**	**data12**
data13	**data14**	**data15**

Слика 7.16. Садржај једног дводимензионалног поља података

Програмски језик **FORTRAN** смешта податке секвенцијално, у поретку по колонама, тј. најпре се сместе чланови **I** колоне, затим **II** колоне, па онда **III** колоне, као на слици 7.17. Приступ било ком елементу, рецимо **X(3,3),** захтева коришћење неке математичке формуле за израчунавање тачне меморијске локације.

локација	садржај	
32000	**data1**	
32001	**data4**	
32002	**data7**	колона1
32003	**data10**	
32004	**data13**	
32005	**data2**	
32006	**data5**	
32007	**data8**	колона2
32008	**data11**	
32009	**data14**	
3200A	**data3**	
3200B	**data6**	
3200C	**data9**	колона3
3200D	**data12**	
3200E	**data15**	
3200F	**Y**	

Слика 7.17. Смештање поља "колона по колона"

Претпоставимо да је почетна адреса (32000_{16}) смештена на локацију са именом **ПОЧЕТАК**, тј. садржај локације **ПОЧЕТАК** је 32000_{16}. Локација

БР_ВРСТА садржи укупни број врста, а локација **КОЛОНА** садржи редни број колоне, а локација **ВРСТА** садржи редни број врсте у којој се налази жељени елемент. Локација податка **X(3,3),** израчунава се по формули:

ЛОКАЦИЈА = **ПОЧЕТАК** + (**КОЛОНА** - 1) * **БР_ВРСТА** + (**ВРСТА** - 1) односно:

ЛОКАЦИЈА = 32000 + (3 - 1) *5 + (3 - 1) = 3200C.

Израчуната адреса податка је 3200C, а то је, у ствари, локација податка **data9**, јер је елемент **X(3,3)** у трећој врсти и трећој колони.

Програмски језик **COBOL** смешта табеле и низове у поретку **врста по врста** (**row major**), слика 7.18. По овој методи сви подаци се смештају тако што се прво меморишу подаци из прве врсте, а за њима из друге врсте, и тако редом. Која је сада физичка адреса члана низа **X(3,3)?**

локација	*садржај*	
32000	**data1**	врста 1
32001	**data2**	
32002	**data3**	
32003	**data4**	врста 2
32004	**data5**	
32005	**data6**	
32006	**data7**	врста 3
32007	**data8**	
32008	**data9**	
32009	**data10**	врста 4
3200A	**data11**	
3200B	**data12**	
3200C	**data13**	врста 5
3200D	**data14**	
3200E	**data15**	
3200F	**Y**	

Слика 7.18. Смештање дводимензионалног поља врста по врста

Поступак израчунавања локације је сличан као у претходном случају. Локација **БР_КОЛОНА** садржи укупан број колона, елемент **X(3,3)** је у трећој врсти и трећој колони, па је онда локација траженог члана:

ЛОКАЦИЈА = **ПОЧЕТАК** + (**ВРСТА** - 1) * **БР_КОЛОНА** + (**КОЛОНА** - 1)

односно:

ЛОКАЦИЈА = 32000 + (3 - 1) *3 + (3 - 1) = 32008.

Израчуната адреса је 32008, а то је, као и у претходном случају, локација податка **data9**. На сличан начин се израчунавају адресе елемената низова са више димензија.

На крају да истакнемо да се проблем организовања података у једном рачунарском систему мора разматрати са два аспекта: логичког и физичког. Логичка организација података зависи, пре свега, од тога како корисници и њихове апликације "виде", тј. обрађују и употребљавају податке. Физичка организација података у великој мери зависи и од карактеристика самих меморијских уређаја (на пример, магнетне траке су по својој природи секвенцијалне), али и од начина обраде података. Да би се рачунарски систем ефикасно користио, физичка и логичка организација података се морају разматрати повезано. О организацији података и њиховом смештању на физичке медијуме и приступу, тј. проналажењу података брину оперативни систем, систем за управљање базама података (**Data Base Management System**, **DBMS**) и посебне групе људи које чине систем аналитичари и систем програмери.

Треба имати у виду да су подаци релативно непроменљиви у времену, али различити корисници их могу обрађивати на различите начине са циљем да добију различите информације. То значи да се једни исти подаци јављају у великом броју апликација. Поставља се питање: Да ли је за сваку апликацију потребно те исте податке меморисати засебно? Наравно да не, јер то доводи до непотребног расипања меморијског простора. Идеално би било када би се један податак меморисао само једном, то јест да се налази само на једном месту. Но, то најчешће није могуће остварити. У сваком случају треба тежити да се подаци минимално редундују (више пута меморишу, тј. складиште на више места). То је неопходно да се не би десило да подаци буду неконзистентни. Наиме, ако би се један исти податак налазио на више места, могло би да се деси да један корисник унесе измену (која је оправдана), а да остали корисници и апликације то не знају и самим тим обрађују нетачан податак.

Апликативни програм не треба да зависи од физичке организације података, односно од тога како су они стварно смештени. Уколико то не би могло да се оствари, свака промена у физичкој организацији одражавала би се на програм, и обрнуто. Подаци се морају физички меморисати тако да буду независни од програма који их користе и да истовремено омогуће програмима да их виде онако како је то њима најпогодније. Исто тако, кориснику се мора омогућити да податке означава апстрактним–симболичким именима, а не да им приступа преко њихове физичке адресе. Дакле, мора се обезбедити физичка и логичка независност података.

Једне исте податке користе разни корисници, па је самим тим безбедност и поузданост података угрожена. Рачунарски систем мора створити могућност да приступ заједничким подацима буде могућ само преко одређених,

дозвољених канала у којима су дефинисана и права приступа за сваког корисника. на овај начин се смањује опасност од случајног или злонамерног уништења података, или њихове измене.

7.7. ЗАКЉУЧАК

Рачунарски системи складиште и обрађују бинарне бројеве и разне кодове реализоване на основу бинарног бројног система. Ефикасна обрада захтева да се ови појединачни чланови организују у различите структуре података. Структуре података су збирке или скупови међусобом повезаних података и могу имати више различитих форми.

Једноставне структуре података допуштају кориснику да представи и смести целе бројеве, бројеве у покретном зарезу, стрингове итд. Низови су скупови ових једноставних елемената организованих ради лакшег приступа подацима и њихове обраде. Сваки део односно сваки елемент низа садржи податак истог типа.

Сложеније структуре података познате као записи (у Pascal–у) или структуре (у језику C) допуштају груписање различитих типова података у логички повезане целине у циљу лакше обраде и чувања.

Повезане листе су форма структура података у којој најмање једна компонента указује на локацију на којој је смештен неки други члан листе. Повезане листе допуштају својим члановима да логички буду повезани, чак и када су физички раздвојени, тј. када су смештени у удаљене локације у примарној или секундарној меморији.

Друге структуре података познате као стабла, магацини и редови не само да садрже податке већ приказују и везе (односе) које постоје између два разна члана. Ове релације могу указивати на поредак (**FIFO, LIFO**) или на односе типа претходник, следбеник.

Логичке структуре података помажу програмерима да по својим потребама организују податке не водећи рачуна како су они стварно смештени на меморијским уређајима. Подаци се коришћењем повезаних листи могу сместити на суседне локације (сукцесивно), или на несуседне локације. Могу се такође смештати на локације које су одређене неком врстом кључа, или комбинацијом разних метода.

Без обзира како су подаци смештени, њима се може приступати секвенцијално или случајно. Случајан приступ може бити остварен коришћењем формула за израчунавање адреса, индексима, или методом остатака при дељењу да би се кључ конвертовао у адресу.

Разумевање структура података је неопходно да бисмо разумели како системи за управљање базама података (**DBMS**), оперативни системи и програмски језици оперишу (са) подацима.

7.8. ПИТАЊА

1. У чему је разлика између скалара, низова и структура података?
2. Шта је заједничко за скаларне податке као што су: цели бројеви, бројеви у фиксном и покретном зарезу, знаковни подаци, логички подаци, показивачи?
3. Описати појмове који се односе на низове као што су: елемент (члан), број чланова (димензија) итд.
4. Која је основна разлика између структуре података и скаларних података?
5. У чему је разлика између једноструко и двоструко повезане листе са шест чланова?
6. Описати појмове везане за стабла: грана, лист, претходник, следбеник, корен.
7. У чему је разлика између магацина и реда?
8. Описати основне типове организација меморије: секвенцијалне, индексиране, повезане.
9. Да ли се код континуалних меморија за ретке низове резервише простор за све чланове низа?
10. Како проблем ретких низова решавају цикличне листе и мапе битова? Да ли и оне резервишу меморијски простор за све чланове низа?
11. Како се меморишу подаци у севенцијалним меморијама? Када су секвенцијалне меморије погодне за коришћење?
12. Како се још може приступити записаним подацима?
13. Објаснити улогу кључа, и како се кључ трансформише у адресу?
14. Како се помоћу индекса пронализи податак?
15. Да ли метод остатка дељења једнозначно пресликава кључ у адресу?
16. Како се у FORTRAN–у одређује адреса члана неког дводимензионалног низа?
17. Која је основна разлика у смештању вишедимензионалних низова у FORTRAN–у и COBOL–у ?

7.9. КЉУЧНЕ РЕЧИ

- адресни простор (**address space**)
- алгоритам (**algorithm**) 218
- аргумент (**argument**)
- бит мапа (**bit map**)
- бинарно стабло (**binary tree**)
- бинарно претраживање
- број у непокретном зарезу (**fixed point number**)
- број у покретном зарезу (**floating point number**)
- цео број (**integer**)
- цео број у проширеној тачности (**long integer**)
- циклична листа (**circular list**)
- циклични ред (**circular queue**)
- динамичка меморија (**dinamic storage**)
- FIFO (**First-In-First-Out**)

- глава листе (**list head**)
- грана (**branch**)
- индекс (**index**)
- информација (**information**)
- **inorder**
- колизија (**collision**)
- колона (**column**)
- колона по колона (**column major**)
- корен (**root**)
- кључ (**key**)
- лабела (**label data**)
- LIFO (**Last-In-First-Out**)
- логички податак (**logical data**)
- магацин, стек (**stack**)
- матрица, дводимензионални низ
- мултииндексирање
- ниска, стринг (**string**)
- низ (**array**)
- остатак дељења
- покретни зарез (**floating point**)
- **postorder**
- повезана, спрегнута листа (**linked list**)
- **preorder**
- претходник (**parent, predecessor**)
- приступ (**access**)
- поље, низ (**array**)
- проблемски подаци (**problem data**)
- простор кључева (**key space**)
- проширена тачност (**double**)
- ред (**queue**)
- ретки низови (**sparse array**)
- секвенцијални (**sequential**)
- синтакса (**sintax**)
- системски подаци (**system data**)
- скаларни подаци (**scalar data**)
- следбеник (**succesor, children**)
- случајан (**random**)
- стабло (**tree**)
- статичка меморија (**static memory**)
- структура података (**data structure**)
- врста (**row**)
- врста по врста (**row major**)
- запис (**record**)
- знање (**knowledge**)
- чвор (**node**)

8. ПРОГРАМСКИ ЈЕЗИЦИ и ЈЕЗИЧКИ ПРОЦЕСОРИ

8.1. О СИСТЕМСКОМ СОФТВЕРУ

Сваки интелектуални поступак који можемо алгоритамски изразити помоћу коначног броја елементарних операција може се преточити у програм и предати рачунару да га изведе. Тиме је постигнуто даље ослобађање човека и преношење рутинског интелектуалног рада на рачунар. За реализацију овог врло важног вида ослобађања човека од заморног рутинског посла, потребно је много сложених поступака. Један од кључних и, слободно се може рећи, најсложенијих делова рачунарског система, који омогућава преношење једног дела интелектуалног рада на рачунарски систем, јесте системски софтвер, а основни део системског софтвера чини оперативни систем. Сви програми, који се у рачунарском систему користе, могу се поделити на две основне врсте:

– *системске програме,*

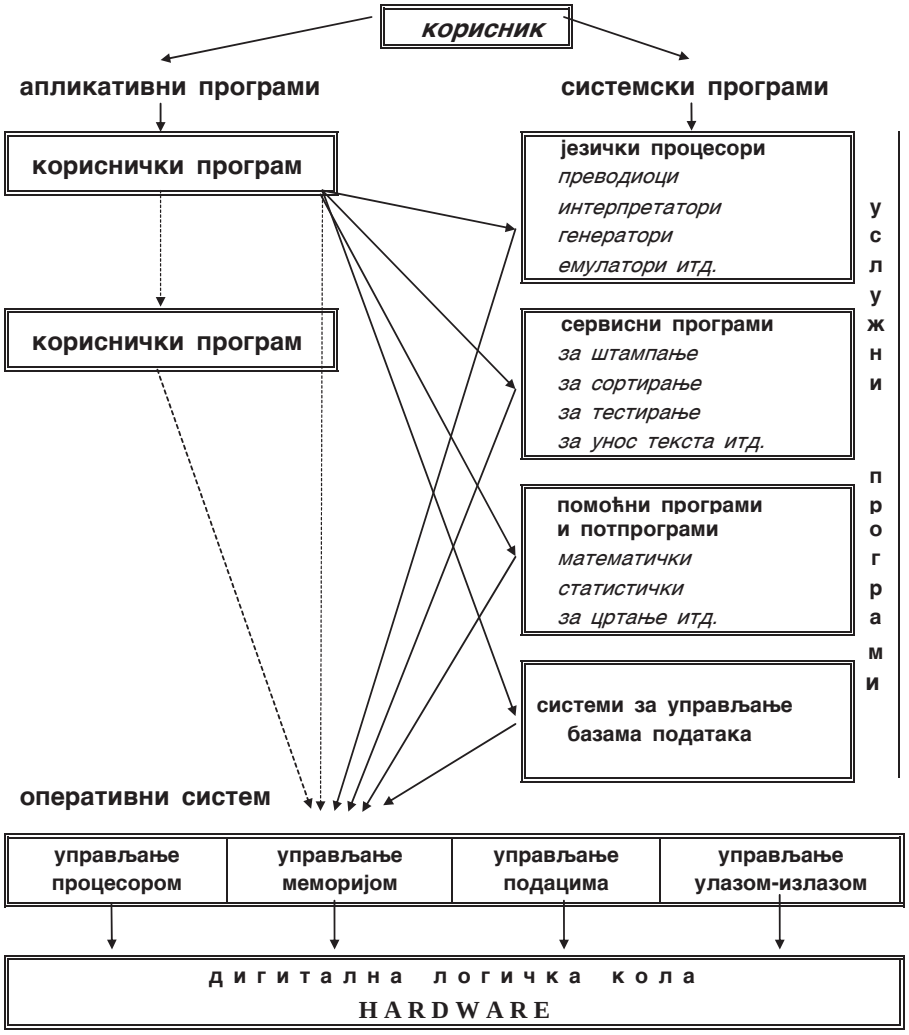
– *корисничке, проблемске* или *апликативне програме*.

Системски програми имају искључиво задатак контроле и управљања рачунарским системом, а проблемски програми имају функцију решавања корисникових проблема. Системски програми намењени су сваком потенцијалном кориснику рачунарског система за који су израђени, док су проблемски програми, који су израђени од стране појединог корисника и намењени да реше његов специфичан проблем (апликацију) и не могу се применити за решавање другог проблема, и за другог корисника често немају никакав значај. Системски програми најчешће не решавају конкретан проблем, и не дају кориснику директно употребљиве резултате. Оперативни систем је на одређени начин посредник између корисника, односно његовог програма, и хардвера рачунарског система.

Кориснички програми су намењени решавању неког конкретног проблема у примени рачунарског система (апликација). Проблемски програми дају конкретне резултате које корисник може директно да користи.

Системске програме (слика 8.1) можемо, такође, поделити на две групе:

– контролно–управљачки програми, тј. оперативни систем,
– услужни програми (**utility software**).



Слика 8.1. Подела и састав системских програма

Оперативни систем има улогу управљања и контроле рада самог рачунарског система. Он је, у ствари, посредник између дигиталних логичких кола и свих осталих врста програма. Оперативни систем је основа, а остали програми су надградња. Оперативни систем извршава управљање и кон–

тролу свих операција у рачунарском систему преко своје четири основне функције. То су функције управљања меморијом, управљања процесором, управљања подацима и управљања улазом–излазом, тј. јединицама за улаз–излаз.

Услужне програме можемо поделити, као што се види на слици 8.1., у четири врсте, и то на:

– *језичке процесоре,*
– *сервисне програме,*
– *помоћне програме и потпрограме,*
– *системе за управљање базама података.*

У језичке процесоре сврставамо све врсте програмских преводилаца, који служе за превођење програма написаног на једном програмском језику, у функционално еквивалентан програм, најчешће у машинском језику. У ову групу програма спадају и разни програмски генератори који на основу задатих параметара генеришу програм, а такође и интерпретери, који за разлику од преводилаца интерпретирају и изводе инструкцију по инструкцију програма.

У сервисне програме спадају, најчешће, програми за обраду текста (**editor**), програми за сортирање и мешање датотека, програми за руковање датотекама (креирање, копирање, ажурирање), програми за тестирање (**tester**) и други.

8.2. ПРОГРАМСКИ ЈЕЗИЦИ

Језик је средство за комуникацију. Језик мора бити тако дефинисан да је разумљив за све учеснике у комуникацији. Постоје природни језици којима се служе људи, и вештачки, измишљени за специјалне намене. Вештачки језици који се користе за писање програма и рад на рачунару зову се програмски језици.

Хардвер рачунара користи меморисане инструкције да би обрадио податке. Рачунарске инструкције се обично организују у програме. Скуп бинарних инструкција које рачунар може извршити директно назива се програм у машинском језику. Људи, с друге стране, радије пишу програме у неком другом језику, а да би се они могли извршити морају се превести у машински језик.

Постоји више различитих програмских језика. Програми се пре извршавања преводе у погодну форму, врло блиску машинском језику. Више програмских целина (модула) у машинском језику, може се повезати тако да чине једну јединствену целину.

Данас постоји неколико стотина програмских језика. Ако као критеријум узмемо степен зависности од хардвера рачунара, језике делимо на две основне групе:

– *машински зависне и*
– *машински независне.*

8.2.1. МАШИНСКИ ЗАВИСНИ ЈЕЗИЦИ

Машински зависни језици се деле на:

– *машинске језике и*
– *машински оријентисане језике.*

8.2.1.1 Машински језици

Програм је логички самоконтролисани скуп рачунарских инструкција. Програмери могу писати ове инструкције у много различитих језика, али хардвер разуме само један облик – машински језик.

Програм написан на машинском језику састоји се од низова бинарних бројева, који се могу директно интерпретирати уз помоћ декодера инструкција. Да би програмирао директно на машинском језику, програмер мора да зна (или да погледа) бинарне кодове за сваку инструкцију која је на репертоару тог рачунара.

Нека је помоћу процесора MC6800 потребно сабрати садржаје две меморијске локације чије су адресе 1A и 1B, и нека се резултат смешта на локацију са адресом 1C. Програм за сабирање написан на машинском језику смештен је у меморију почев од локације 100_{16} и има следећи облик:

локација	**садржај**	
0100:	10010110	пренеси у акумулатор A број са меморијске локације
0101:	00011010	1A
0102:	10011011	сабери садржај акумулатора A и меморијске локације
0103:	00011011	1B
0104:	10010111	пренеси садржај акумулатора A у локацију
0105:	00011100	1C

У овом примеру, у преносу података и сабирању, коришћено је директно адресирање и акумулатор А. Низови нула и јединица би сасвим другачије изгледали у случају коришћења неког другог начина адресирања.

Ако би у преносу података и сабирању користили акумулатор B, онда би исти програм у машинском језику имао следећи облик:

локација	**садржај**	
0100:	11010110	пренеси у акумулатор B број са меморијске локације
0101:	00011010	1A
0102:	11011011	сабери садржај акумулатора B и меморијске локације
0103:	00011011	1B
0104:	11010111	пренеси садржај акумулатора B у локацију
0105:	00011100	1C

Програмер (који пише на машинском језику) мора бити упознат и са архитектуром рачунара, и мора знати који регистри се користе у тим инструкцијама. Програмер мора, такође, прорадити све расположиве начине адресирања, и знати како да каже рачунару да користи одређени начин адресирања.

Програмирање у машинском језику захтева од програмера да одреди тачне меморијске локације, које су потребне за остваривање свих инструкција гранања, и локације свих података. Захтева се и непосредно познавање кодова стања, и кодова прекида.

Све ово чини програмирање у машинском језику врло дуготрајним, напорним и подложним грешкама. Како свака врста CPU има свој сопствени сет инструкција, програми у машинском језику, развијени за један рачунар, у општем случају су неупотребљиви за други рачунар. Због овога су развијени други приступи програмирању. Неки језици су сасвим мало померени у односу на машински (машински оријентисани језици), док други нису ни налик на бинарне инструкције које непосредно контролишу рад рачунара (виши програмски језици).

8.2.1.2 Машински оријентисани језици

Машински оријентисани језици су врло блиски машинском језику, и директно се ослањају на њега, а самим тим и на хардвер рачунара. Они се могу сврстати у два нивоа:

– *асемблерски језици или симболички машински језици и*
– *макроасемблери.*

8.2.1.2.1 *АСЕМБЛЕРСКИ ЈЕЗИЦИ*

Асемблерски језици (**assembly languages**) решавају неке од проблема који су везани за програмирање у машинском језику. Ови језици допуштају програмеру да уместо бинарних кодова операција користи симболичке кодове инструкција (**mnemonics**). Коришћењем мнемоничких инструкцијских кодова,

програмер на **IBM PC** може да пренесе податак из меморије у регистар CPU-а, помоћу **MOV** инструкције уместо употребом операције $\mathbf{A1}_{16}$.

Термин мнемоник значи "звучи као", и користе се зато што симболички кодови често звуче као инструкције. **DEC**, на пример, звучи као **decrement**. Иако **A** не звучи превише као **add**, то је свакако бољи избор него рецимо **Q**, или неки други симбол који ни на који начин не асоцира на сабирање (**add**). Асемблерски језици допуштају употребу симболичких имена за регистре, и дозвољавају програмеру да придружују имена меморијским локацијама. Дакле, програмер не мора да зна тачну нумеричку адресу локације податка у меморији.

Због коришћења симболичких имена за инструкције, регистре и меморијске локације, ови језици се још називају симболички машински језици. Програмери који програмирају у асемблеру, још морају да знају који су начини адресирања на располагању, и како се задаје жељени начин адресирања. За сваки асемблерски језик то је различито.

Како би изгледао програм на симболичком машинском језику за сабирање два броја из претходног примера, уз коришћење акумулатора A, и ако би локацијама дали симболичка имена num1 (1A), num2 (1B) и sum (1C):

```
LDA A   num1
ADD A   num2
STA  A   sum
```

У случају коришћења акумулатора B програм би био:

```
LDA B   num1
ADD B   num2
STA  B   sum
```

Како су асемблерски језици врло слични машинским језицима, они се најчешће сматрају језицима ниског нивоа (**low - level languages**). Једна линија асемблерског кода постаје једна машинска инструкција Програмер мора знати мнемонике за сваку инструкцију из репертоара рачунара, и разумети како да укаже на различите начине адресирања. Захтева се, такође, и познавање заставица стања, како се оне постављају и бришу, и шта оне означавају. Асемблерски језици су, као и машински језици, специфични за сваки CPU. Асемблерски програм написан за **VAX 11/780** не може се користити за процесоре **Intel 80386, MC 68020, IBM system /370** итд.

8.2.1.2.2 *МАКРОАСЕМБЛЕРСКИ ЈЕЗИЦИ*

Употреба виших језика за програмирање делотворно искоришћава радно време програмера, али се у меморији троши много простора за уписивање програма, као и време процесора за извођење таквих програма. Супротно

томе, програмирањем у асемблерским програмским језицима делотворније се искоришћава меморија и време процесора, али се захтева много радног времена програмера. Када би се заједно примениле асемблерске наредбе и наредбе вишег прогамског језика, добиле би се све предности асемблерских и виших језика, а уклонили њихови недостаци.

Такву могућност пружа употреба тзв. макроасемблера. Макроасемблер омогућава програмеру да помоћу асемблерских наредби сам дефинише **макроинструкције** за своју употребу (а које се састоје од низа асемблерских наредби), а такође може да дефинише и нове наредбе виших програмских језика. Разлика је у томе што су у вишим програмским језицима наредбе стандардне и унапред дефинисане за различите могуће употребе. Напротив, макроинструкције развија сам корисник за своју употребу, па се извршавају исто тако ефикасно као и асемблерски програми. Свакој макроинструкцији програмер додељује своју симболичку ознаку. Другим речима, програмер сам дефинише своје макроинструкције које су ефикасне као и асемблерске инструкције, а омогућавају лакшу и бржу израду врло читљивих програма, као и језици вишег нивоа.

8.2.2. МАШИНСКИ НЕЗАВИСНИ ЈЕЗИЦИ

Програмски језици које зовемо језици високог нивоа (**high - level languages**), смањују потребу програмера да разуме унутрашње детаље архитектуре рачунара. Свака реченица у програму на вишем програмском језику мора се превести у инструкције на машинском нивоу, пре него се почне са извршавањем. Док се свака инструкција, на нивоу асемблера, преводи у једну машинску инструкцију, дотле се реченица на језику вишег нивоа нормално преводи у много машинских инструкција. Постоје специјални програми који се зову компајлери **(compilers)** и интерпретери (**interpreters**), који врше превођење. Ово превођење се обавља у присуству одређеног програмског окружења које одређује оперативни систем рачунара.

Превођење обично није коначна фаза обраде, тј. добијени машински еквивалент се не може одмах извршавати. Потребне су још неке обраде над добијеним резултатом превођења; наиме, потребно је још извршити повезивање са неким другим програмима, а на крају треба извршити пуњење, тј. смештање програма у меморију.

Језици високог нивоа такође се могу разврстати на следеће хијерархијске нивое:

– *ниво процедуралних језика*,
– *ниво проблемски оријентисаних језика* (STRESS, COGO, SIMULA),
– *ниво непроцедуралних језика* (QBE, SQL, FOCUS, CUPID),
– *ниво декларативних језика* (DYANA, PROLOG).

Ми ћемо у даљем тексту разматрати само процедурално оријентисане програмске језике.

8.2.2.1 Процедурално оријентисани језици

Процедурално оријентисани језици (**procedure oriented languages**) исказују експлицитно секвенцијалну природу решавања проблема помоћу рачунара, а да при томе не захтевају од програмера познавање организације рачунара и његовог машинског језика. Ови језици допуштају програмеру да специфицира процедуре, или алгоритме, које треба користити за решавање проблема.

Многи процедурално оријентисани језици су развијени да подрже специфичне потребе: **FORmula TRANslation language** (**FORTRAN**) за научнике и инжењере, **Beginers All-purpose Symbolic Instruction Code** (**BASIC**) за почетнике у програмирању, **COmonon Busines Oriented Language** (**COBOL**) за пословне потребе, **LISt Processing** (**LISP**) за обраду листа, **SNOBOL** за обраду стрингова итд. Језици попут **Ada** и **PL/1** су пројектовани општије. Не гледајући за коју примену су развијени, процедурално оријентисани језици често имају сличне функције. Појединачне реченице које имплементирају поједине функције у сваком језику су другачије, али се у свима налазе сличне категорије примена. То су функције за манипулисање подацима, улазно–излазне операције, контролне структуре, декларације, потпрограми итд.

У развоју процедуралних језика два су момента од пресудног значаја:

- *формални опис језика* ***ALGOL-60*** *(**ALGOrithmic Language**) помоћу **BNF** нотације којом се уводе нови концепти у програмске језике, који су имали велики утицај на програмске језике који су се касније појавили,*
- *појава **PASCAL**–а који је развијен на бази **ALGOL**–а, а значајан је нарочито због увођења нових структура података и структурираног програмирања.*

Језици вишег нивоа допуштају програмеру да меморијске локације означава симболичким именима. Та имена се често зову **промењиве** или **варијабле** (**variables**), јер представљају меморијске локације чији се садржаји могу мењати.

Непроменљиве вредности се зову **константе,** и у општем случају то су бројеви (10, 150.25, итд) или карактери (**"б**", "**ово је константа**", итд.). Неки језици допуштају да се и константама придруже имена; тако се броју 3.14159 може придружити име **PI**. Број и врста знака, и начин њиховог комбиновања за прављење имена варијабли и константи зависе од врсте језика.

Програм написан на процедурално оријентисаном језику за један рачунар, уз неке мање измене или без измена, може се пренети и на било који други рачунар који подржава тај језик. Тачније, који има одговарајући преводилац са тог језика на машински језик.

8.2.3. СИНТАКСА ЈЕЗИКА

Правила која дефинишу како се граде елементарне (речи) и сложене конструкције (реченице) језика чине граматику језика. Синтакса језика изучава да ли су језичке конструкције граматички коректне и даје могућност формалног откривања грешака. Синтакса не улази у значење тих конструкција. Значење конструкција изучава семантика језика.

Синтакса језика је често врло сложена, а врло је важно да програмер схвати како се конструишу правилне реченице. Има неколико приступа који програмеру пружају ове информације.

Један од начина приказивања синтаксе програмског језика је употреба дијаграма речи, односно **нотација са заградама**. Сваки програмски језик има одређени скуп речи чији је смисао и начин употребе унапред дефинисан, а које се називају резервисане речи или службене речи.

Посматрајмо правила која се користе за интерпретирање COBOL програма, чије реченице обично имају следећи облик:

DEVIDE{ Идентификатор1 / Литерал1 }INTO идентификатор2 [ROUNDED][ON SIZE ERROR реченица]

Велика слова указују на резервисане речи, а мала слова су корисничке речи. Велике заграде указују на могућност избора, један од више. У овом случају програмер може да бира између **идентификатора1** и **литерала1** (у COBOL–у се варијабле зову идентификатори). Подвучене резервисане речи су обавезне, док се оне неподвучене могу, а не морају користити (опције). Искази који се налазе у средњим заградама такође су опционални, али ако их употребимо онда се они морају писати по другим правилима. На пример, ON SIZE ERROR израз није обавезан, али ако се користи онда он мора бити написан као SIZE ERROR заповедна–реченица или као ON SIZE ERROR **реченица** (заповедна).

Други начин приказивања правила за писање коректних реченица јесте **BNF нотација** (**Backus–Nauer Form**). Ово је врло распрострањен метод за дефинисање синтаксе. Бекус је ову нотацију први пут употребио за опис нотације програмског језика ALGOL–60.

Овај приступ користи изведене (дељиве), и коначне (недељиве) симболе. Недељиви симболи су основне, коначне јединице језика и укључују у свој састав уобичајене симболе као што су **0, 1, 2, A, B,** +, =, итд. Дељиви еле–

менти, ниске, састављени су од недељивих и других дељивих симбола. Сви изведени, дељиви елементи одређени су именом унутар угаоних заграда, нпр. <цифра> која је дефинисана на следећи начин:

< цифра > :: = 0|1|2|3|4|5|6|7|8|9

где ::= има значење "је дефинисано као", а "|" има значење "**ИЛИ**". Претходна реченица дефинише изведени симбол < цифра > као 0 или 1 или 2 или 3 или 4 или 5 или 6 или 7 или 8 или 9.

Прости изведени симболи се називају подниске, и могу се користити за дефинисање сложенијих конструкција језика (ниске), као на пример у ALGOL-у:

< цео број без знака > ::= < цифра > | < цео број без знака > < цифра >|.

Ова реченица каже да је < *цео број без знака* > било која < *цифра* >, или било који < *цео број без знака* > иза којег следи < *цифра* >. Ово је рекурзивна дефиниција, јер се сам појам, који се дефинише, налази као алтернатива унутар дефиниције.

Претходна дефиниција означава да цео број знака може бити 0, 1, 2, 3, 4, 5, 6, 7, 8 или 9, али такође и 11, 12, итд, јер је он састављен од целог броја без знака (само једна цифра) иза кога следи нека друга цифра. По истом правилу су и бројеви 123, 5432 и 654321 такође цели бројеви без знака.

ALGOL иде даље у дефинисању изведених симбола:

< цео број > ::= **< цео број без знака** > | + **< цео број без знака** > | – **< цео број без знака** >

< децимални део > ::= **.** **< цео број без знака** >

< децимални број > ::= **< цео број без знака** > | **< децимални део** > | **< цео број без знака**> **< децимални део** >

Сагласно овој синтакси, цели бројеви су облика 543, –543 или +543.

Децимални део садржи у себи комбинације попут .05 или .001, али не укључује –.005 или +.25, јер се испред **.** не може налазити ни један симбол.

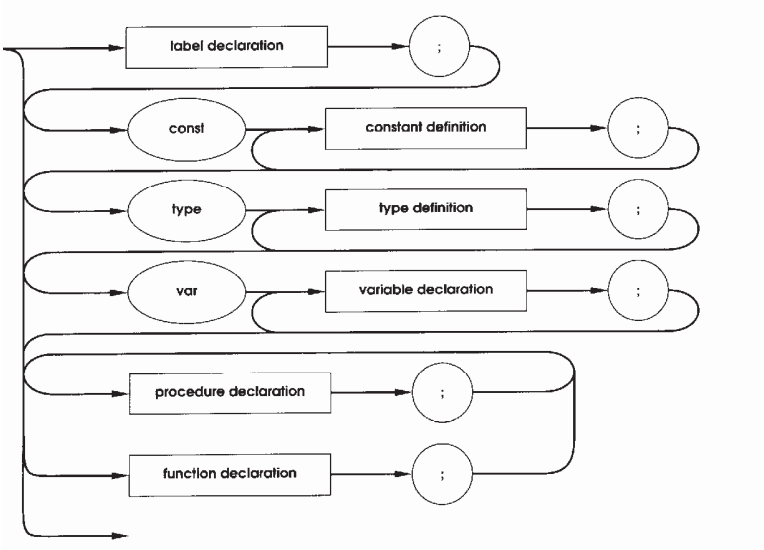
Децимални бројеви су облика 123, .05 или 12.5, али не и 5. јер иза децималне тачке, по дефиницији децималног броја, мора следити бар једна цифра.

Трећи је графички начин дефинисања синтаксе који користи кружиће, правоугаонике и усмерене линије и зове се **синтаксни дијаграм**. Кружићи личе на мехуриће, па се овај метод зове и bubble diagram. Кругови означавају завршне, недељиве симболе, правоугаоници ниске и подниске, тј. изведене, дељиве симболе, а усмерене линије указују на потребан поредак низања

делова. На слици 8.2 дата је једна оваква структура за **Pascal** – програме. Она показује да Pascal програм почиње речју **PROGRAM** (кључна реч и то је коначни симбол), иза које следе: један идентификатор, параметри програма, тачка–зарез, део за декларисање, тело, тачка итд. Свака изведена конструкција мора бити дефинисана негде у синтакси.

Слика 8.2. Дијаграм главних структура Pascal програма

На слици 8.3 описан је део Pascal–програма за декларисање и показује нам велике могућности које овакви дијаграми имају. Можемо пратити усмерену линију право на доле и било где скренути удесно, или стићи до дна, када део за декларисање не постоји. Разним елементима дела програма за декларисање приступа се праћењем стрелице удесно,



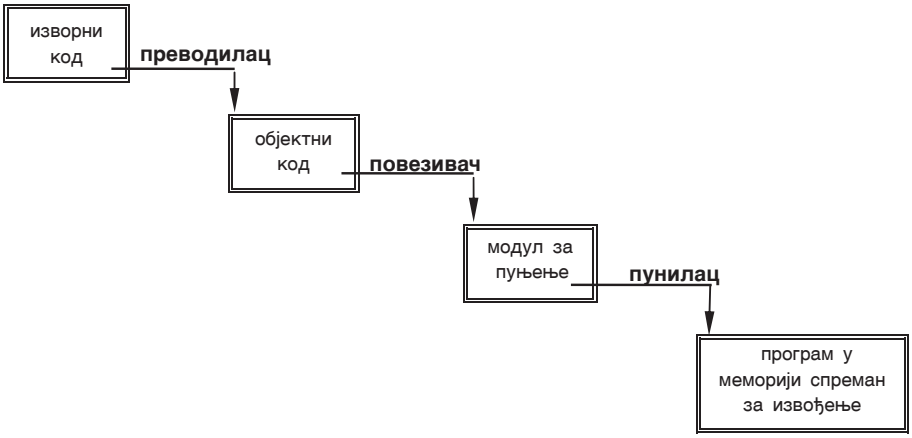
Слика 8.3. Синтакса декларационог дела **Pascal** програма

На пример, дијаграм каже да се променљиве декларишу коришћењем коначног симбола **VAR** иза којег следи стварна дефиниција и након тога знак **;** (тачка–зарез). Усмерена линија може садржати и петљу, што допушта дефинисање већег броја променљивих. Овај дијаграм јасно показује и редослед декларисања и дефинисања: најпре лабеле, онда подаци (константе, типови и тек онда варијабле), а на крају дела програма за декларисање, дефинишу се процедуре и функције.

8.3. ЈЕЗИЧКИ ПРОЦЕСОРИ

Језички процесори су системски програми пројектовани да поједноставе процес програмирања. Преводиоци су језички процесори који прихватају програме писане на неком од програмских језика (асемблеру или неком вишем језику), и генеришу функционално еквивалентне програме на неком другом језику (најчешће на машинском). На слици 8.4 приказан је поступак превођења неког асемблерског програма до машинског програма који се може извршити у рачунару.

Асемблерски, бејзик (BASIC), или неки други програм, који улази у језички процесор зове се **изворни код** (**source code**). Преводилац чита тај код и генерише бинарну верзију програма. Неки језички процесори креирају бинарни код који је одмах спреман за извршавање. То су такозвани апсолутни или нерелокативни програми.



Слика 8.4. Процес превођења, повезивања и пуњења

Други пак генеришу специјални тип кода на машинском нивоу који се зове **предметни** или **објектни код** (**object code**), или **машински симболички код**. Објектни код се не може извршити, већ се мора даље модификовати пре извођења. Ту додатну обраду обављају системски програми који се зову **повезивачи** (**linkers**) и **пуниоци** (**loaders**).

Зависно од нивоа улазног језика преводиоци се деле на:

- ***асемблере**, који преводе програме са симболичког–машинског језика на машински језик,*
- ***макроасемблере**, који преводе програме са макроасемблера на машински језик,*

- ***компилаторе*** *или* ***компајлере****, који преводе програме са процедурално–оријентисаних програмских језика на машински језик,*
- ***генераторе****, који преводе програме са проблемски–оријентисаних програмских језика на машински језик,*
- ***интерпретере*** *или* ***интерпретаторе****.*

8.3.1. АСЕМБЛЕРИ

Асемблер је језички процесор (преводилац), чији је улазни језик симболичко машински језик (**assembly language code**), а излазни језик је близак машинском језику или машински језик.

У процесу превођења симболичко–машинских програма најсложенији посао је израчунавање израза јер је у асемблеру допуштено коришћење инструкцијских мнемоника и симболичких адреса података, што знатно олакшава писање програма. Асемблер не може да израчуна симболички израз док вредности свих симбола у њему нису дефинисане. Правила писања процедура у симболичко–машинском језику не захтевају да дефиниција симбола претходи њиховом коришћењу. Ове чињенице указују на могућност да асемблер не може израчунати неке изразе док не обради све наредбе у асемблерској процедури. Практична последица се своди на то, да асемблер мора двапут да обради асемблерску процедуру, пре него што обави потпуно превођење, тј. мора да буде организован у два пролаза.

У првом пролазу се чита изворни програм, дефинишу се вредности свих симбола који се у њему појављују, и формира се интерна форма асемблерског програма, како би се избегло понављање неких активности у обради програма. Други пролаз чита интерну форму програма, асемблира машинске наредбе, израчунава изразе чија вредност одређује садржај адресног поља наредби, гради предметни програм, и даје извештај о превођењу.

Погледајмо како се програм на **PC-XT** (са процесором **Intel 8088**) за сабирање три броја са симболичким именима **num1, num2, num3** а резултат смешта на локацију **sum** (слика 8.5.), конвертује у машински језик (слика 8.6.). Током првог проласка свака линија је прочитана и извршено је лексичко претраживање, (**lexical scan**), које идентификује инструкцијске и регистарске мнемонике, специјалне симболе и симболичке адресе. Када је компонента једном идентификована, асемблер може да одреди да ли та линија садржи инструкцију која се мора конвертовати у машински код, или псеудоинструкцију, која представља директиву самом асемблеру.

Асемблер претпоставља да програм почиње од унапред одређене локације (нпр. $\mathbf{0000}_{16}$ или $\mathbf{0100}_{16}$), и ту адресу држи у бројачу локација (**location counter**), да би означио део меморије који користе инструкције и подаци. Када се свака линија прочита и испита, овај бројач показује величину

простора који ће инструкције користити, али стваран машински код неће бити креиран пре другог проласка.

Током првог проласка, асемблер такође креира и попуњава табеле које садрже све симболичке адресе (које се налазе у програму) и њихове ло–кације. Неки асемблери смештају у табеле и неке друге информације.

```
1          mov   ax, num1
2          add   ax, num2
3          add   ax, num3
4          mov   sum, ax
5          hlt
6  num1    dw    205
7  num2    dw    961
8  num3    dw    347
9  sum     dw    ?
10         end
```

Слика 8.5. Асемблерска верзија програма

symbol	**type**	**location**	
num1	w	?	(а)
symbol	**type**	**location**	
num1	w	?	
num2	w	?	(б)
symbol	**type**	**location**	
num1	w	?	
num2	w	?	(ц)
num3	w	?	
sum	w	?	
symbol	**type**	**location**	
num1	w	010F	
num2	w	?	(д)
num3	w	?	
sum	w	?	
symbol	**type**	**location**	
num1	w	010F	
num2	w	0111	(е)
num3	w	0113	
sum	w	0115	

Слика 8.6. Развој табеле симбола током процеса асемблирања

Први пролаз кроз програм на слици 8.5 врши следеће обраде:

1. Постави бројач локација на почетну вредност, на пример: **0100.**
2. Чита прву линију и врши лексичко претраживање, и идентификује три јединствена симбола: **mov, ax**, и **num 1**.
3. Покушава да идентификује сваки од њих, користећи табеле операционих кодова. Овај процес идентификује **mov** као мнемоник инструкције, а **ax** као мнемоник регистра.
4. Како симбол **num1** није нађен у табели инструкцијских и регистарских мнемоника, проверава се да ли је он већ у табели симбола, и пошто

није додаје се у ту табелу. Маркира се **num 1** као адреса речи, јер је **ax** 16–битни регистар, слика 8.6 (а).

5. Инструкција захтева три бајта (информација се добија из табеле операционих кодова); дакле бројач локације се повећава на **0103**.

6. Чита се и претражује следећа линија, идентификују се симболи **add, ax** и **num 2**.

7. Користећи табелу операционих кодова идентификује **add** као инструкцију и **ax** као регистар.

8. Како **num2** није мнемоник ни инструкције ни регистра, проверава се табела симбола, **num2** није у табели, па се додаје у табелу и маркира као адреса речи, слика 8.6 (б).

9. Повећава се садржај бројача локација за **4** (наредба **add** има 4 бајта).

10. Понављају се кораци 6–9 за линије 3, 4, 5 у програму, и бројач локације се сваки пут повећава сагласно дужини инструкција. У табелу симбола се уписују симболи и њено стање након ових корака дато је на слици 8.6 (ц).

11 Чита се и претражује линија 6, утврђује да је то поље лабела (**label field**), које указује на локацију симболичке адресе.

12. Испитује табелу симбола и у поље **num 1** уписује садржај који указује на локацију која одговара текућој вредности бројача локација, **010F,** слика 8.6 (д).

13. Поље **dw** је псеудоинструкција која казује асемблеру да резервише меморијску реч на текућој локацији. Операнд **205** је податак који ће бити смештен на текућој локацији. Бројач локација се повећава за 2 јер једна реч има 2 бајта.

14. Понављају се кораци 11–13 за линије 7, 8 и 9. Симбол "**?**" у линији 9, указује да нема податка који ће бити смештен у простор резервисан **dw** псеудоинструкцијом. Слика 8.6 (е) показује садржај табеле симбола после вишеструког понављања ова 3 корака.

15. Инструкција "**end**" у линији 10 је још једна псеудо–инструкција која указује на крај изворног програма.

Псеудоинструкција "**end**" такође сигнализира крај првог проласка кроз изворни код. У овој тачки асемблер обавља разне провере, а пре свега проверава да ли све симбличке адресе имају додељене локације, и прелази на други пролаз. Форма добијена након првог пролаза назива се **интерни програм**.

У другом пролазу се поново испитују све линије али не изворног већ интерног програма, а асемблер замењује симболичке адресе одговарајућим

вредностима из табеле симбола, генерише машинске кодове инструкција и улазних података, и те бинарне кодове уписује у **објектну датотеку** (**object file**), тј. **предметни програм**.

Резултат превођења је предметни програм и извештај о превођењу (извршни извештај). Процес асемблирања се често завршава генерисањем изворног извештаја као на слици 8.7., који садржи програм на машинском језику добијен на бази асемблерских инструкција и табелу симбола. Табела симбола садржи сваки симбол, његову програмску локацију, меморијску локацију и тип.

```
1     0100    A1      010F                          mov     ax, num1
2     0103    03      06      0111                  add     ax, num2
3     0107    03      06      0113                  add     ax, num3
4     010B    A3      0115                          mov     sum, ax
5     010E    F4                                    hlt
6     010F    007B                    num1          dw      123
7     0111    01C8                    num2          dw      456
8     0113    0141                    num3          dw      321
9     0115    ????                    sum           dw      ?
10                                                  end

        symbols:
name                            type            location        line

NUM1                            Word            010F            6
NUM2                            Word            0111            7
NUM3                            Word            0113            8
SUM                             Word            0115            9
10 Source Line(s) , No Assembly Error (s) .
```

Слика 8.7. Једноставан асемблерски извештај

8.3.2. КОМПИЛАТОРИ

Компилатори или компајлери (**compilers**) су програми који преводе програме написане на процедурално–оријентисаним језицима, у програм на машинском језику. Значи, изворни језик је неки виши програмски језик, а циљни језик је машински језик. Компилатор је врло сложен програм који се састоји од више десетина хиљада инструкција, па развој једног преводиоца траје и по неколико година.

Саставни делови сваког језика јесу: азбука, лексика, синтакса и семантика. Лексика језика проучава саставне делове конструкција језика, и начине образовања исправних конструкција језика. Значење синтаксно исправних

реченица чини семантику језика. Конкретно, семантика реченице програм–ског језика је дефинисана акција која ће се извршити на рачунару.

Азбука, лексика и синтакса у потпуности одређују скуп исправних кон–струкција језика и односе унутар и између конструкција. Семантика одређује везу између конструкција различитих језика. Превођење је пресликавање једног записа у други, при чему семантика мора остати непромењена. Превођење с једног језика на други, у општем случају, састоји се у промени азбуке, лексике и синтаксе, а семантику треба сачувати. При томе пунилац мења само лексику, асемблер–азбуку и лексику, а компилатор–азбуку, лексику и синтаксу програмског језика. Превођење се извршава у више корака, тако да се у сваком кораку извршавају строго одређени задаци. На крају сваке обраде издаје се и одговарајући извештај.

Први корак превођења је лексичка анализа. Задатак лексичке анализе је довођење улазног програма на стандардни (каноничан) облик и превођење на унутрашњи језик. У унутрашњем језику све речи су истог формата, што олакшава даљи рад преводиоца. Упоредо с превођењем на унутрашњи језик извршава се лексичка контрола улазног језика. Задатак лексичке контроле је провера да ли симболи припадају азбуци. На крају се генерише извештај који обавештава и о евентуалним лексичким грешкама.

Други корак превођења је синтаксна анализа. Задатак синтаксне анализе је распознавање врсте наредби и њихова синтаксна контрола. О откривеним синтаксним грешкама шаље се одговарајући извештај.

Трећи корак превођења је семантичка анализа, тј. генерисање кода. Задатак семантичке анализе је генерисање еквивалентне наредбе (или наредби) излазног језика. Експлицитно речено, у трећој етапи извршава се превођење на циљни језик. На крају се врши још и оптимизација, која има за циљ скраћивање времена извршавања и смањивање потребног мемо–ријског простора.

8.3.3. ИНТЕРПРЕТАТОРИ

Обично су процес превођења и процес извршавања програма временски раздвојени. Прво се цео програм преведе, а потом изврши. Преводиоци који тако раде називају се преводиоци компилаторског типа. Међутим, постоје преводиоци код којих се процес превођења и процес извршавања програма одвијају истовремено. Такви преводиоци се називају интерпретатори.

Интерпретатори су један други тип језичких преводилаца. Уместо да праве предметни (**object**) код, ови преводиоци надгледају и извршавају програм по принципу **линија по линија**. Типични примери програмских језика који подржавају овакав режим извођења програма су BASIC, APL и ADA.

Интерпретатор се састоји из блока за анализу који служи за распознавање оператора изворног језика, скупа потпрограма од којих сваки одговара по једном оператору, и управљачког блока који одређује редослед извршавања оператора.

На основу инструкција управљачког блока, блок за анализу разматра операторе изворног програма, препознаје их и испитује услове њиховог извршавања. Информације о могућности извршавања оператора шаљу се управљачком блоку, а овај на основу њих позива одговарајући потпрограм који извршава операције предвиђене за тај оператор.

Интерпретатори омогућавају рад кориснику рачунара у режиму дијалога са терминала, а такође се користе за извршавање програма написаних за други рачунар.

Недостатак интерпретатора је неефикасно коришћење машинског времена. На пример, при извршавању циклуса један исти оператор се интерпретира при сваком пролазу кроз циклус. Компилатор обави само једанпут компилацију, а потом се извршава радни програм на машинском језику.

8.3.4. ПОВЕЗИВАЧИ и ПУНИОЦИ

Процес превођења се не може сматрати завршном фазом у развоју једног програма јер многи компилатори и асемблери конвертују изворни код у предметни код. Предметни код је верзија програма у машинском језику (то је такозвана машинско–симболичка форма), али се не може извршавати из неколико разлога. **Прво**, програм може позивати потпрограме, који нису конвертовани у машински језик у исто време када и текући програм, па самим тим његове адресе нису познате. **Друго**, програм не мора бити смештен у меморију почев од почетне адресе коју је претпоставио преводилац.

Повезивач (**linker**) конвертује предметни код у модул за пуњење који се може сместити у меморију и извршити.

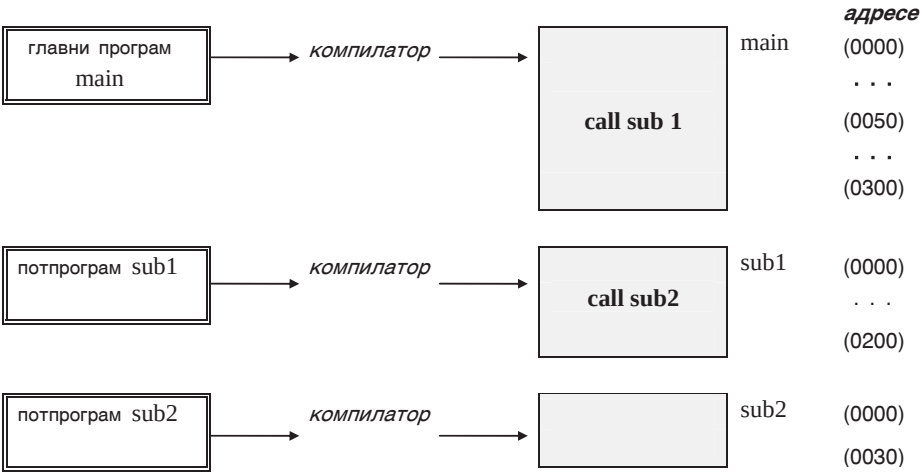
Неки системи користе такозвани пунилац да премосте корак модула за пуњење, и смештају предметни код директно у меморију ради извршавања.

Локације које ће програм заузимати у меморији обично нису познате у време превођења. Компилатори и асемблери подразумевају да сваки програм почиње од локације нула (или неке друге фиксне унапред одређене локације). То значи да ће и главни програм и његова два потпрограма (слика 8.8) бити преведени у три сегмента предметног кода, тако да почињу од исте локације.

Још један проблем се може наслутити из овог примера, а тиче се локација потпрограма. У току превођења главног програма, локација потпрограма

sub1 (и било којег другог) није позната. Превођење изворног кода у предметни код праћено је следећим проблемима:

1. *Сви модули предметног кода имају исту стартну позицију.*
2. *Адресе потпрограма нису познате.*
3. *Многе меморијске адресе ће бити погрешне, ако се предметни модул помери у меморији (релокација).*
4. *Морају се остварити неки путеви за потпрограме да укажу на њихову стартну локацију или улазну тачку.*



Слика 8.8. Изворни код је транслиран у предметни код

Треба напоменути да постоји једноставно али непрактично решење које отклања потребу за пуниоцем. Програмски систем се може написати као једна целина, на пример, на симболичко–машинском језику. Ако се у програму зада адреса смештања преведеног програма у оперативну меморију, асемблер може преведени програм да смести директно, почевши од задате адресе, и да одмах започне његово извршавање. Очигледно је да овакво решење елиминише потребу за пуниоцем.

Међутим, недостаци оваквог приступа су тако велики да он не представља прихватљиво решење. Цео програмски систем се мора написати на једном језику и преводити сваки пут кад треба да се изврши. Велики и сложени програмски системи се веома тешко могу правити као једна целина, како због поузданости рада, тако и због ефикасности у програмирању. Осим тога, може се десити да се велики програми не могу преводити преводиоцима на коришћеном рачунарском систему.

8.3.4.1 Време повезивања и придруживања (додељивања меморије)

Време придруживања (**binding time**) је време када се одређују коначне локације програма и података у главној меморији, односно, када им се додељује меморијски простор. Постоје бројне могућности избора, а направљени избор битно утиче где ће се и како извршавати програм. Придруживање (додељивање, а неки аутори га зову повезивање) може се извршити у четири различита тренутка:

- *за време писања програма,*
- *за време превођења програма,*
- *за време пуњења програма у меморију,*
- *за време извођења програма.*

Зависно од начина придруживања добијамо различите начине адресирања:

- **апсолутно адресирање** (програмирање у машинском језику), где сам програмер специфицира стварне физичке адресе, па нема никаквих трансформација адреса.
- **симболичко адресирање** (програмирање у симболичком језику), где је функција одређивања физичких адреса фиксирана у програмском преводиоцу, и он генерише стварне физичке адресе уместо симболичких имена које користи програмер.
- **релокативно адресирање**, где програм за пуњење (**loader**) одређује стварне адресе, а програмски преводилац даје само релативне адресе у односу на почетак програма.
- **динамичко адресирање**, где се физичке адресе одређују за време извођења, обично употребом посебних регистара.

Прва могућност је крајње једноставна, а она се односи на придруживање меморијских локација програму и подацима у току писања програма. Сва одговорност за коришћење меморије и могуће конфликте, у овом случају, пада на програмера.

Мало бољи приступ је придруживање у време превођења (компилирања). Ово омогућава изворном програму и симболичким подацима да буду модификовани. Такође је могуће укључити потпрограме и низ рутина у ове програме. Међутим, када је једном преведен, програм може да ради само са једне фиксне почетне локације у меморији. Ово је прихватљиво за микрорачунаре који раде у једнопрограмском окружењу.

Када се потпрограми преводе засебно, придруживање мора бити одложено док се рутине не преведу и повежу. Ово омогућава креирање библиотека рутина које могу бити коришћене од било ког броја програма и у било којим комбинацијама. Када је придруживање једном урађено, програм може да ради са било које стартне позиције. и ово је погодно за једнокорисничке системе (**personal computers, PC**). На овај начин је омогућено смештање програма на било који слободан меморијски простор, но проблеми и даље постоје. Када је програм једном напуњен не може се више померати.

Врло модерни оперативни системи користе **preemptive** процедуре доделе процесора (са претпражњењем, тј. превентивним пражњењем процесорског реда). То значи да чак и програм који се управо извршава (у стању извођења, **run**) може изгубити контролу над централним процесором и привремено бити избачен из главне меморије. Када се тај програм поново враћа у меморију, можда више неће бити на истим оним локацијама на којима је претходно био. Све претходно омеђене локације биће нетачне. Решење овог проблема је употреба адресирања уз помоћ базних регистара. Придруживање се тада врши након пуњења базних регистара новим базним адресама.

Крајње решење јесте да се придруживање обавља у време извршавања инструкције и то је динамичко придруживање. Ово захтева коришћење система са такозваном виртуелном меморијом.

8.4. ЕДИТОРИ

Едитори текстова су најчешће коришћени делови системског софтвера. За већину корисника у интерактивним рачунарским системима они представљају главну спрегу са рачунаром. Данас је у употреби на стотине различитих едитора текста: многи рачунарски центри имају своје локалне едиторе, нови рачунари се појављују са сопственим едиторима, а у персоналним микрорачунарима најчешће се користе екрански едитор (**edit**), **WordPad**, **Notepad** и **Norton editor**.

Да би фаза превођења могла отпочети, неопходно је претходно припремити програм у форми коју прихвата преводилац. Припрема програма обавља се у посебној фази развоја програма. Та фаза се назива едитовање, а део системског софтвера који реализује ту фазу едитор текста. Резултат едитовања, с тачке гледишта преводиоца, је изворни програм. Међутим, едитори имају ширу примену од припреме и модификације изворних програма. Те примене су: писање разних писама и меморандума, извештаја, разних табела и спецификација, структурираних текстова и књига (као текст–процесори).

Оно што је заједничко за све едиторе текста, је скуп едиторских функција које су стављене на располагање корисницима. Могућности едитора обухватају преко 200 функција које се њиме могу извршавати, а ми ћемо навести само неке од најважнијих и начешће коришћених функција:

– *уметање објеката у текст,*

– *уклањање објеката из текста,*

– *замена објеката у тексту новим објектима,*

– *премештање објеката с једне на другу позицију у тексту,*

– *копирање објеката на разне позиције у тексту,*

– *раздвајање објеката у тексту на више делова,*
– *спајање више објеката у тексту у један објекат,*
– *приказивање појединих делова текста,*
– *адресирање одређеног места у тексту,*
– *адресирање делова текста по садржају,*
– *формирање новог документа,*
– *уклањање постојећег документа,*
– *припрема документа за уређивање.*

Под објектима текста подразумевамо знакове, речи, линије, реченице, параграфе и секције, који, као целина, подлежу одређеној операцији *уређивања*.

Процена квалитета едитора може се засновати на четири димензије коришћења које су од фундаменталне и практичне важности, а то су:

– *време потребно да се обаве основни задаци уређивања од стране стручњака*. Временска димензија процене квалитета едитора односи се на време које је потребно стручњацима да обаве рутинско едитовање (модификацију) текста.

– *цене коштања грешке* при уређивању од стране стручњака. Ефекти грешака приликом едитовања се мере временом грешке. Време грешке је време које корисник утроши чинећи грешке, откривајући их, исправљајући их и успостављајући поново нормалан ток едитовања. Време без грешака је време едитовања умањено за време грешака.

– *учење основних функција уређивања* од стране почетника (који нису имали никакво претходно искуство у раду са рачунаром). Димезија учења се састоји у учењу почетника раду са едитором и мерењу резултата учења. Појединачна мерења резултата учења се одређују временом трајања учења, подељеног укупним бројем едиторских функција које ученик може да обави у потпуности и без гледања у белешку.

– *функционалност едитора* се мери процентом функција које тај едитор може да обави у односу на укупан број свих познатих едиторских функција. Одлучивање о томе да ли се нека функција може или не може обавити није бинарног типа. Скоро свака функција се може обавити сваким едитором уз већи или мањи напор, па се другачије вреднује она функција која се реализује већим напором.

8.5. ЗАКЉУЧАК

Да би хардвер рачунара обавио користан рад неопходне су му инструкције. Сам хардвер разуме само инструкције машинског језика, док програми могу бити написани на неком другом језику, па се морају преводити на машински језик.

Језик који је најближи машинском зове се асемблер. Он користи симболичка имена за кодове операција, регистре и меморијске локације. Како је асемблер врло сличан машинском језику, програмер мора добро познавати архитектуру рачунара.

Језици који нису везани за хардвер рачунара зову се језици вишег нивоа. Они користе преводиоце познате као компилаторе и интерпретере, а који испитују линије изворног кода програма и растављају их на препознатљиве компоненте, које издвајају из њих.

Када је линија једном растављена, преводилац може да провери да ли су поштована правила, односно синтакса програмског језика.

Модул предметног кода (**object**) садржи машинске инструкције и специјалне табеле које показују на инструкције које морају бити модификоване када се програм смести у меморију или је релоциран, тј. померен на нову локацију. Смештање у меморију обављају директно пуниоци, али се они често комбинују са повезивачима.

Разумевање програмских језика и њиховог превођења је неопходно да би се схва–тило зашто ће неки програми радити на неким системима, а на другим неће. То ће, такође, помоћи да се схвати зашто се неки програми извршавају брже од других, и троше мање меморије.

Схватање померања програма у меморији (релокација) и придруживања меморијских локација програму (и подацима), помаже да се схвати извођење више програма у једном CPU (мултипрограмирање), као и једнопрограмско окружење које налазимо у персоналним рачунарима (**Personal Computer systems, PC).**

8.6. ПИТАЊА

1. Зашто је тешко програмирати на машинском језику?
2. Које су предности асемблерског језика у односу на машински?
3. У чему се битно разликују виши од нижих програмских језика?
4. Шта чини синтаксу језика?
5. Које методе се користе за дефинисање правилних програмских реченица?
6. Кратко описати неколико проблема везаних за програмирање на машинском језику, и како су они превазиђени асемблерским језиком.
7. Објаснити зашто је потребно да се програми преведу пре извођења?
8. Зашто асемблер (преводилац) мора два пута да обради сваку наредбу?
9. Како се добија предметни програм?
10. Које су основне фазе компилирања?

11. У чему је разлика између компилатора и интерпретера?
12. Како повезивање програма и пуњења преведеног програма у меморију утиче на израчунавање адреса меморијских локација?
13. Када је све могуће извршити придруживање стварних меморијских локација програму и како то утиче на адресе меморијских локација?
14. Шта је улога едитора текста?
15. Које су основне функције едитора?
16. Који су основни критеријуми за одређивање квалитета едитора?

8.7. КЉУЧНЕ РЕЧИ

- асемблер језик (**assembly language**)
- асемблер преводилац (**assembler**)
- асемблерски програм
- бројач локација (**location counter**)
- директива (**pseudoinstruction**)
- интерпретатор,интерпретер (**interpreter**)
- изворни код (**source code**)
- језици ниског нивоа (**low-level languages**)
- језици високог нивоа (**high-level languages**)
- компилатор (**compiler**)
- коначан, завршни, недељиви (**terminal**)
- лабела,обележје (**label**)
- лексичка анализа, претраживање (**lexical scan**)
- машински језик (**machine language**)
- машински зависни језици (**machine oriented languages**)
- мнемоник (**mnemonic**)
- модул за пуњење (**load module**)
- непомични, нерелокативни, програм
- повезивач (**linker**)
- предметни, објектни код **(object code**)
- придруживање (**binding**)
- процедурално оријентисани језици (**procedure oriented languages**)
- процедуре (**procedures**)
- псеудоинструкција (**pseudoinstruction**)
- релокација (**relocation**)
- релокативан (**relocated**)
- резервисана реч (**reserved word**)
- синтакса (**sintax**)
- синтаксна анализа (**sintax analysis**)
- табела симбола (**symbol table**)
- улазна тачка (**entry point**)
- виши програмски језици (**high-level languages**)
- време придруживања (**binding time**)

9. ОПЕРАТИВНИ СИСТЕМИ

УПРАВЉАЊЕ ПРОЦЕСОРОМ
УПРАВЉАЊЕ МЕМОРИЈОМ
УПРАВЉАЊЕ ПОДАЦИМА
УПРАВЉАЊЕ УЛАЗОМ–ИЗЛАЗОМ

9.1. ДЕФИНИЦИЈЕ И МОДЕЛ

Рачунарски систем је сложени скуп техничких уређаја (хардвера), програма који том хардверу обезбеђују инструкције које му казују шта да ради, и података који се обрађују. Људи који користе рачунаре, било као оператори, програмери или пројектанти, у раду се придржавају разних формалних и неформалних процедура. Многе процедуре могу се преточити у програме, тако да их обављају рачунари, а не људи.

Неке од поступака смо већ поменули: језички преводиоци омогућавају људима писање програма на језику који разуме човек, а потом се тај програм преводи на машински језик који машина разуме. Повезивачи и пуниоци омогућавају спрезање више машинских програма у једну програмску целину. Тај програм се може више пута употребити, са разним комбинацијама потпрограма, а да се не мора поново писати, и увек се смешта на разне локације у меморији. Али, многа питања остају још увек без одговора: **Како** више програма може бити истовремено у меморији? **Ко**

одлучује који ће програм бити у меморији? **Ко** ће стварно користити CPU? **Како** ћемо обезбедити да сваки програм добије ресурсе? **Како**....? Одговор на сва ова питања садржи у себи оперативни систем.

У првој генерацији рачунара, опслуживање рачунарског система било је потпуно препуштено оператеру, који је морао да припреми све што је потребно да се задатак обраде може обавити. Човек је имао пуну контролу над рачунарским системом. Оператер је био у могућности да све потребне радње обави, јер је систем био спор и изводио се само један програм, односно један задатак. Може се рећи, да је **искоришћење** рачунарског система, односно његових најважнијих ресурса **централног процесора и централне меморије, било слабо**. Претежни део времена трошио се на послове оператера и улаз–излаз, а много мањи део времена на рад централног процесора, па је рад система био неефикасан. Ипак, због мале брзине система у целини, као и самог централног процесора, однос тих времена је био у прихватљивим границама.

У другој генерацији рачунарских система долази до повећања брзине рада централног процесора, повећања централне и екстерних меморија, нових и бржих улазно–излазних јединица, а програми се пишу у симболичком машинском језику. Оператер више није у стању да ефикасно опслужује рачунарски систем, јер су његове реакције сувише споре. Једини излаз је био у пребацивању низа контролних функција, израдом посебних контролних програма, са оператера на сам рачунарски систем. Дакле функције опслуживања и управљања системом подељене су између оператера и контролних програма. Ти програми се укључују у одређеним ситуацијама, као што су, на пример, припрема програма за извођење, контрола улазних и излазних уређаја и разни послови око учитавања програма и његове припреме за извођење. Дакле, у рачунарима друге генерације можемо разликовати две основне врсте програма, и то:

– контролне програме,

– проблемске програме.

У трећој генерацији рачунарских система, контролно управљачке функције се још више пребацују на сам рачунар, због још већег пораста брзине, величине меморија, броја улазних и излазних јединица, али и због појаве мултипрограмирања. Дакле, човек дефинитивно губи могућност контроле и управљања интерном ситуацијом у рачунарском систему и све што је могуће пребацује се на рачунарски систем, односно на поједине системске програме. Скуп свих тих програма можемо назвати једним именом– **оперативни систем**. Програмер се ослобађа низа сложених рутинских послова и даје му се могућност већег ангажовања на креативном делу посла. Но, поред контролно–управљачких програма у рачунарима треће генерације развијен је и читав низ услужних програма, чији је задатак да

даље олакшају и поједноставе употребу рачунарских система. Због тога, према намени, можемо извршити поделу и читавог софтвера, на две врсте:

– *системски софтвер*, и
– *кориснички, проблемски* или *апликативни софтвер*.

Оперативни систем је део системског софтвера, и један од најважнијих и најсложенијих делова рачунарског система. Оперативни систем је скуп системских програма који служе за контролу рада и управљање читавим рачунарским системом и проблемским програмима. Он је ум самог рачунарског система.

Оперативни систем је сложени низ програма који снагу рачунара стављају у службу корисника. Он је одговоран за надгледање и контролу коришћења рачунарског система. Оперативни систем кориснику обезбеђује спрегу са рачунарским системом. Он обавештава о стању рачунара и о насталим грешкама. Оперативни систем дели примарну и секундарну меморију између више корисника, и свима омогућава употребу **CPU**, тј. обраду њихових инструкција и података у процесору.

Оперативни систем је веза (**interface**) између самог хардвера рачунарског система и корисника којима омогућује да лакше креирају, тестирају, изводе и одржавају програме, а истовремено и контролише међусобно дељење ресурса рачунарског система у циљу ефикасног рада.

Оперативни систем је део системског софтвера који је најближи хардверу рачунара. Он представља основну помоћ у раду и коришћењу хардвера рачунара. Функција оперативног система је да олакша решавање проблема који интересује корисника, пружајући му, притом, разноврсне програмске услуге и да при томе обезбеди ефикасно и продуктивно коришћење рачунара.

Оперативни систем је врло сложен део системског софтвера, а састоји се од више релативно независних целина. Треба имати на уму да сваки произвођач рачунара има своје оперативне системе, па је тешко дати општу структуру оперативног система. На слици 9.1. приказан је хијерархијски модел оперативног система.

Поједини делови представљају нивое оперативног система. Хијерархијски модел има, овде, следећи смисао: на посматраном нивоу оперативног система могу се захтевати услуге само од његових нижих нивоа, а никако од виших. Најнижи слој је познат као језгро оперативног система (**nucleus, kernel**).

Не постоје чврсто дефинисана правила која регулишу расподелу функција оперативног система по нивоима. Оперативни ситеми су огромни, већина од њих не може стати у главну меморију. Зато се у меморији увек налазе само најважнији делови оперативног система, такозвани **резидентни део**, који

активира и завршава корисничке програме, врши доделу меморије и датотека, обавља операције улаза – излаза.

Слика 9.1. Хијерархијски модел оперативног система

Резидентни део оперативног система мора обавезно подржавати механизам прекида, јер је он основа вишепрограмског рада и комуницирања рачунара са спољним светом. Део оперативног система који мора увек бити присутан у оперативној меморији обично се назива језгро или нуклеус. Функције које се користе од стране свих нивоа морају се сместити у језгро оперативног система. Остали делови се убацују у оперативну меморију када су потребни и из ње избацују када више нису потребни.

9.1.1. ЈЕЗГРО

Језгро оперативног система обезбеђује реализацију следећих функција:

- *управљање прекидним системом рачунара и обрада прекида,*
- *планирање и евиденција извршавања процеса* (програм који је унет у рачунар, и налази се у фази извођења назива се процес),
- *манипулација над програмима и комуникација између програма.*

Да би језгро оперативног система остварило своју основну функцију, морају постојати хардверски реализовани делови (дигиталне логичке мреже), на које ће се језгро надограђивати. Дигитална логичка кола која чине основу за остваривање функција језгра оперативног система подржавају реализацију:

- *механизма прекида,*
- *привилегованог скупа инструкција,*
- *заштитног механизма адресирања меморије,*
- ***real-time** сатног механизма.*

Треба стално имати у виду да се оперативни систем и хардвер рачунара непрекидно преплићу, тј. језгро оперативног система је делимично реализовано хардверски а делимично софтверски. Хардверски део обезбеђује потребну брзину, а софтверски део флексибилност.

Привилеговани скуп инструкција (или привилеговане инструкције) чине све инструкције које проблемски програм не може директно да изведе. Те инструкције може изводити само оперативни систем а служе, на пример, за маскирање и демаскирање прекида, извођење улаза–излаза, заустављање централног процесора, приступ у адресне и заштићене регистре, итд.

Оперативни систем изводи привилеговане инструкције када је систем у стању које називамо **контролно** или **привилеговано стање** (**control mode, supervisory mode, privileged mode**), док се неприлеговане инструкције изводе у стању које називамо **проблемско стање** (**problem mode, user mode**). Проблемски програм посредно изводи привилеговане исктрукције на тај начин што посебном инструкцијом позове оперативни систем (**supervisor call, extracode**), а оперативни систем онда изведе привилеговану инструкцију. Овакав начин рада је усвојен зато да програми међусобно не би утицали један на другог, али и због унификације приступа **U/I** јединицама. Систем се непосредно пребацује из проблемског у контролно стање и обрнуто, према томе како се догађају и решавају прекиди.

Механизам заштите меморије служи за заштиту од погрешног адресирања и утицаја једног процеса на други на недопуштен начин. Такав механизам аутоматски осигурава интегритет сваког програма и његових података. Он зависи од самог меморијског хардвера као и од система за доделу меморије.

Real-time сатни механизам је бројач времена који служи за контролу и евиденцију коришћења ресурса рачунарског система за сваки појединии процес.

Само језгро оперативног система можемо, такође, поделити на три основна дела:

- *први ниво обраде прекида (**FLIH, First Level Interrupt Handler**),*
- *диспечер (**dispatcher**),*
- *рутине за комуницирање међу процесима.*

Први ниво обраде прекида садржи анализатор прекида и сервисне рутине за обраду појединих врста прекида. Овај део језгра се зове први ниво обраде прекида, јер је након извођења овог дела језгра прекид делимично

опслужен, али није коначно решен, јер треба одлучити да ли се прекинути програм наставља или не. У то одлучивање се укључује посебни део оперативног система који о томе одлучује, који је такође део језгра и зове се диспечер, а који служи за пребацивање процесора међу програмима, узимајући увек онај програм који има највећи приоритет.

9.1.2. САСТАВ ОПЕРАТИВНОГ СИСТЕМА

Оперативни систем сачињавају четири групе програма које обезбеђују управљање основним ресурсима рачунарског система: управљање процесором, управљање меморијом, управљање улазом и излазом и управљање подацима.

Управљање процесором

Процесор је један од најважнијих ресурса рачунарских система, мада у појединим конкретним ситуацијама неки други ресурси могу бити критичнији. Управљање процесором се може поделити на два нивоа:

– *ниво непосредне доделе процесора неком процесу* (програму), тј. предаје контроле над процесором неком процесу (програму) да би се извршавале његове инструкције, и

– *ниво одлучивања који од могућих програма има највећи приоритет да би постао процес*, и да би у неком следећем тренутку времена добио контролу над процесором.

Управљање меморијом

Управљање меморијом обавља управљање оперативном, главном меморијом рачунара, којој централни процесор приступа директно, ради узимања инструкција или података. На овом нивоу обављају се следеће функције:

– *реализација одређене стратегије додељивања меморије* (редослед додељивања меморије пословима, статичко или динамичко додељивање, принципи додељивања, итд.),

– *додељивање меморије* (алгоритми избора почетне локације сегмента који се додељује),

– *спровођење одређене стратегије ослобађања меморије* (укрупњавање мањих ослобођених делова меморије у веће, ослобађање делова меморије, редослед обраде захтева за доделу меморије).

Управљање улазно - излазним уређајима

На нивоу управљања уређајима реализују се следеће функције:

– *обезбеђивање независности уређаја* (програми треба да буду независни од типа уређаја који се користе за улазно–излазне операције),

– *обезбеђивање ефикасног рада уређаја* (пошто улазно–излазне операције представљају често уско грло рачунарског система, пожељно је да се оне обаве ефикасно колико год је то могуће),

– *обезбеђивање јединственог концепта анализе рада свих уређаја рачунарског система,*

– *реализација одређене стратегије додељивања уређаја* (на пример, редослед додељивања уређаја пословима, начин додељивања уређаја: наменски, са деобом, као виртуелни, итд).

– *додељивање уређаја* (физичко додељивање уређаја, контролних јединица и канала пословима),

– *реализација одређене стратегије ослобађања уређаја* (на пример, уређај се ослобађа тек кад се посао којем је додељен заврши).

Управљање подацима

Ниво управљања подацима треба да обезбеди софтверска средства за организовање и приступање подацима на начин који одговара кориснику рачунарског система. Концепција управљања зависи пре свега од врсте података и режима њиховог коришћења. На овом нивоу реализују се следеће функције:

– *формирање и брисање основних структура података* (датотека),

– читање датотека и уписивање у датотеке,

– *управљање секундарним меморијским простором* (на пример, меморијским простором на дисковима),

– *обезбеђивање услова за симболичко обраћање датотекама* (према њиховом имену),

– *заштита података од намерног и ненамерног уништења* (услед отказа система),

– *заштита података од неовлашћеног приступа и коришћења,*

– *деоба датотека* (података) између више послова (корисника).

9.2. ПРОЦЕСИ И ПРОМЕНЕ СТАЊА

Рачунарски систем који има само једну централну процесорску јединицу може да извршава само једну инструкцију у једном тренутку времена. Оперативни систем мора овим корисницима да обезбеди приступ овом

најважнијем ресурсу, и не сме допустити никоме да је монополизује. Да би извршио овај задатак, оперативни систем мора имати одређене информације о сваком кориснику, и сваком програму у рачунару, у сваком тренутку времена.

Већина оперативних система управља процесима (**process**) или задацима (**tasks**). Процес је програм у извођењу, а изводи га програм. То значи да ће три корисника, који сви компилирају своје програме, бити представљени са три разна процеса. То такође значи да ће један програм, који је сам себе разделио на два дела ради истовременог (симултаног) извођења, бити представљен са два разна процеса. Сам оперативни систем је састављен од низа процеса. Процес је динамичког карактера; он чини низ активности које се проводе унутар рачунарског система, за разлику од програма који је статичког карактера и састоји се од низа инструкција.

Сви процеси који уђу у рачунарски систем пролазе кроз низ стања током свог боравка у рачунару. Превођење процеса из једног стања у друго (**state transition**) обавља такође оперативни систем. Процес се састоји из низа корака који следе један за другим. Између два корака процес може бити прекинут, а његово извршавање може се наставити у неком другом тренутку времена, или чак на другом процесору.

Када се два или више процеса у току неког интервала времена налазе у рачунару, и у неком су стању између свог почетка и краја извођења, кажемо да су то паралелни процеси. Паралелни процеси се стално такмиче ради коришћења процесора или других ресурса рачунарског система, иако сами, међу собом нису зависни.

Процеси се у току свог извођења могу наћи у следећим односима:

– *међусобно искључење* (**mutual exclusion**),

– *синхронизација процеса* (**synchronization**),

– *застој* (**deadlock, deadly embrace**).

Међусобно искључење процеса је такав однос међу процесима који не допушта истовремено извођење два процеса у појединим њиховим деловима У оперативном систему постоји механизам који временски усклађује извођење процеса, дакле механизам који ће при сваком коришћењу недељивог ресурса проводити међусобно искључивање процеса. Уколико процеси истовремено улазе у своје критичне секције (део процеса у коме се захтева коришћење недељивог ресурса), треба провести међусобно искључење, дакле пропустити само један процес.

Синхронизација процеса састоји се у обезбеђивању одређеног редоследа у извођењу појединих делова два разна процеса. Синхронизација није једнозначан однос као међусобно искључење. То је сложенији однос међу

процесима, а врло често се јавља у облику који се назива однос **произвођач-потрошач** (**producer-consumer relationship**). Понекад се овај однос назива и однос предајник–пријемник (**sender-receiver relationship**).

Као пример, треба споменути процес који припрема излазну линију за штампање на штампачу. Тај процес преузима са спољне меморије (диска) излазне резултате које је неки процес тамо уписао, и припрема их за штампање, стављајући их у одређену међумеморију. Кад се тако формирана линија стави у међумеморију, други процес који изводи штампање, узима линију и преноси је даље на штампач, тј изводи штампање линије. Процес који ставља готове линије у међумеморију за штампање, можемо поистоветити са процесом произвођачем, а процес који изводи штампање са процесом потрошачем. Као што се види процес оваквог типа сусреће се доста често у рачунарском систему.

Измена информација између два процеса одвија се према одређеним правилима која регулишу синхронизацију између ова два процеса. **Прво** правило се може једноставно изразити овако: процеси не могу истовремено приступати међумеморији. То значи да када **први** процес ставља неку величину у међумеморију, не сме допустити **другом** процесу да узима величину из међумеморије. Важи и обрнуто, па када **други** процес узима величину из међумеморије, **први** процес јој не може приступити.

Друго правило каже да **први** процес произвођач не може пунити у пуну међумеморију. Процес може ставити у међумеморију само онолико података колико у међумеморију може да стане. Стављање у међумеморију више величина него што је капацитет међумеморије значи да се нека величина која није искоришћена (потрошена), односно узета из међумеморије уништава, јер би се преко ње уписала нова величина.

Треће правило каже да процес потрошач не сме празнити празну међумеморију. Оно спречава процес потрошач да узима величине из међумеморије које је већ једном прочитао.

Синхронизација се обавља интервенцијом оперативног система. Процеси се у рачунару одвијају потпуно асинхроно, јер:

- *у сваком тренутку током извођења било ког процеса може наступити прекид,*
- *процеси могу за своје коришћење тражити ресурсе система који су у том тренутку заузети,*
- *потпуно је непредвидиво колико ће дуго неки процес задржати процесор, и колико ће нових процеса ући у систем и када ће добити процесор,*
- *додела процесора је повезана и са доделом меморије и других ресурса, па се у општем случају не може са сигурношћу знати каква ће ситуација настати у систему у току извођења ових процеса.*

Трећа врста односа процеса јесте **застој**, када се процеси заустављају. Оваква ситуација може лако настати у рачунарском систему (ако смо неопрезни). Нека се у рачунару изводе два процеса **П1** и **П2**, и нека оба у току свог извођења траже коришћење ресурса **Р1** и **Р2** (нпр. штампач и дисплеј). Нека процес **П1** преузме контролу над штампачем, а **П2** контролу над дисплејом. Ако у току штампања, процесу П1 затреба и дисплеј за приказ неких података, **П1** ће морати да сачека да дисплеј буде слободан. Ако истовремено процес **П2**, не ослобађајући дисплеј, пожели нешто да штампа, ни он неће моћи да продужи са својим извођењем. Значи, ова два процеса су се нашла у стању застоја.

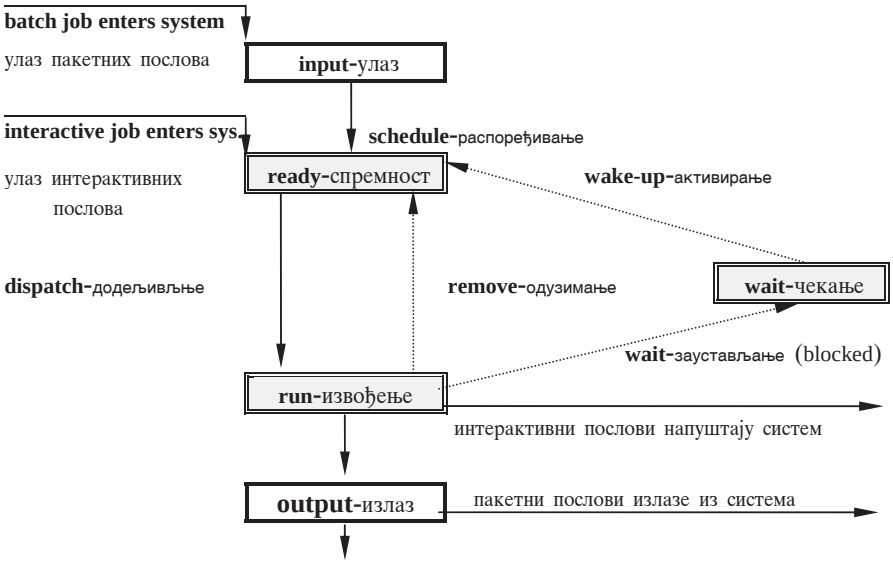
Сваки процес у току свог извођења може да се нађе у једном од три основна стања, која се међусобом могу смењивати, и то су:

1. *стање спремности за извођење, готовост* (**READY state**),
2. *стање извођења, извршавања* (**RUN state**),
3. *стање чекања, блокирања* (**WAIT state, blocked**),

Неки аутори разликују и нека помоћна стања, кроз које процес пролази само једном у току извођења, као што су :

4. *стање започињања, улаза* (**START, input state**),
5. *стање заустављања, излаза* (**STOP, output state**)

На слици 9.2 шематски су приказане промене стања процеса у току његовог боравка у рачунарском систему.



Слика 9.2. Промена стања процеса

Многи велики рачунари имају два главна начина за улазак процеса у систем. Најчешћа улазна тачка данас је један интерактивни терминал, који је придружен рачунару. Други приступ је могућ преко групног, пакетног система (**batch**). Процес који је прихваћен од стране групног система прелази у стање започињања процеса. Ово значи да ће процес бити препознат од стране оперативног система, али му неће бити додељени ресурси, нити ће му бити допуштено да се такмичи (конкурише) за CPU.

Да би конкурисао за CPU, процес мора да из стања улаза пређе у стање спремности. Овај корак се зове распоређивање (**scheduling**), а обавља га део оперативног система који се зове: **управљач задацима**. Оперативни систем даје процесу ресурсе који су му неопходни за рад. У њих спадају: оперативна меморија, простор у табелама оперативног система, расположиве датотеке. Процес у стању спремности (обично) има све што му треба за извођење, осим CPU. Интерактивни задаци се прихватају директно у стање спремности. Ако је систем препуњен, и не може да подржи нове процесе, оперативни систем ће једноставно спречити прикључење нових корисника.

У једном интервалу времена бира се само један процес и он користи CPU. Изабрани процес се пребацује из стања спремности у стање извођења посредством компоненте оперативног система која се зове диспечер (**dispatcher**). Рачунарски системи са једном CPU могу имати само један процес у стању извођења (сви остали процеси су у неком другом стању). Тај процес има контролу над процесором и то је једини процес чије се инструкције извршавају у том интервалу времена. Процес остаје у стању извођења и користи CPU све док не заврши свој посао или док не буде блокиран (у стању чекања).

Интерактивни процеси напуштају систем по завршетку њиховог извођења, док групни процеси прелазе у стање заустављања, тј. излаза из система. Процесу у стању излаза више није потребна CPU, али још увек чека да се заврше његове операције излаза (штампање или памћење на спољне меморије). Процес напушта рачунар тек након што су завршене све његове излазне активности.

Процес у фази извођења може захтевати коришћење ресурса који нису одмах расположиви, као што је приступ датотеци на диску или траци, штампање и слично. Док чека на ресурсе, процес није у могућности да користи CPU. Ова ситуација се зове блокирање или чекање, и тада кажемо да је процес у стању чекања.

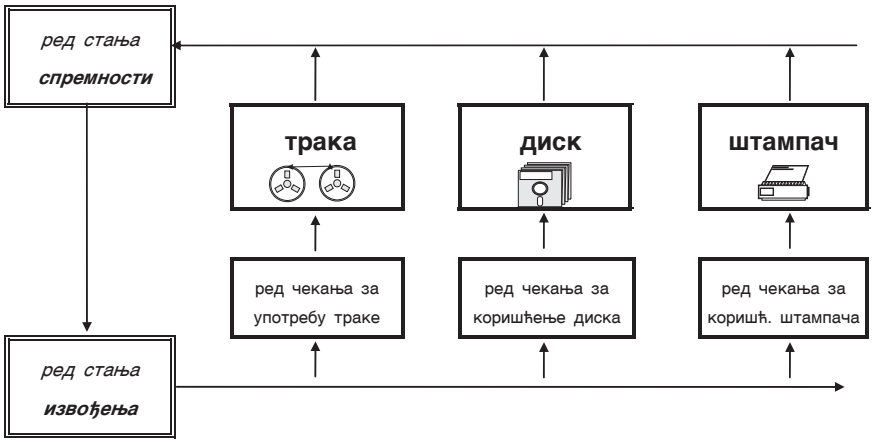
Како ће оперативни систем обрађивати блокиране процесе зависи од алгоритма управљања који користи диспечер. Неки оперативни системи користе алгоритам управљања без претпражњења, превентивног пражњења процесорског реда **(non preemptive scheduling algorithm**), који допуштају процесима, који се изводе, да задрже контролу над CPU–ом, иако су

блокирани. Ови системи чекају док се ресурси не ослободе а онда се процеси деблокирају (ослобађају) и прелазе у стање извођења.

Време које је потребно да се обаве улазно–излазне операције, и да се процес деблокира, може бити врло дуго, јер су ови уређаји врло спори у односу на CPU. Процесор је докон док је процес блокиран. Ово стање се зове запослено чекање (**busy wait**).

Други оперативни системи, који су пројектовани тако да држе процесор што је могуће дуже у заузетом стању, одузимају процесор од блокираног процеса и препуштају CPU неком другом задатку који је у стању спремности. Овакав приступ доводи до бољег коришћења рачунарског система, јер ресурс који је потребан блокираном процесу можда неће бити одмах расположив (али у то време процесор обрађује инструкције других процеса).

Контрола приступа ресурсима, тј. додела ресурса, обавља се уз помоћ редова за чекање. Оперативни систем за сваки активни уређај (у употреби) формира ред чекања у који оперативни систем уписује процесе који траже употребу тог уређаја. Када уређај заврши опслуживање текућег процеса он се обраћа оперативном систему. Оперативни систем узима следећи процес из реда за чекање, и из блокираног стања га преводи у стање спремности, а уређај се ставља на располагање том процесу, слика 9.3.



Слика 9.3. Процеси који чекају на **U/I** операције, чекају у реду за приступ уређају. Када је **U/I** операција извршена процес се враћа у стање спремности.

Одузимање централног процесора од процеса, назива се претпражњење (**preemption**). Када се процес деблокира, он прелази у стање спремности и

враћа се назад у ред који се назива **процесорски ред**, и поново може да се такмичи за CPU.

Блокирање процеса се такође догађа и када процес тражи услуге од оперативног система, јер је и сам оперативни систем такође скуп процеса и активан процес (из састава оперативног система) мора преузети контролу над CPU-ом пре него испоштује захтев за пружање услуга. Када је услуживање завршено, процес који је тражио услугу је деблокиран и враћа се у стање спремности.

Неки процеси би могли монополизирати коришћење CPU, јер никада не прелазе у блокирано стање, и не траже услуге ни од оперативног система ни од других уређаја у систему. То су **процесорски ограничени послови** (**processor bound jobs**). Ово је случај код програма који имају обимна израчунавања над подацима који су већ у меморији.

Многи оперативни системи спречавају овакве ситуације коришћењем специјалног тајмера, чији је задатак да генерише сигнал прекида. Када процес пређе у стање извођења, тајмер се стартује. Ако процес није завршио обраду, или се није блокирао током неког јединичног интервала, кванта времена, тајмер генерише сигнал прекида, процес (задатак, таск) губи контролу над CPU и враћа се у процесорски ред. Диспечер изабира неки други процес, и допушта да се извршавају његове инструкције.

Добар оперативни систем ће покушати да направи баланс између послова који имају много захтева за улазно–излазним операцијама (**I/O bound jobs**), и задатака који стално захтевају употребу CPU (**processor bound jobs**).

Да би успоставио баланс, оперативни систем мора понекад неке процесе избацити из система, односно суспендовати их, а неке друге процесе унети у систем.

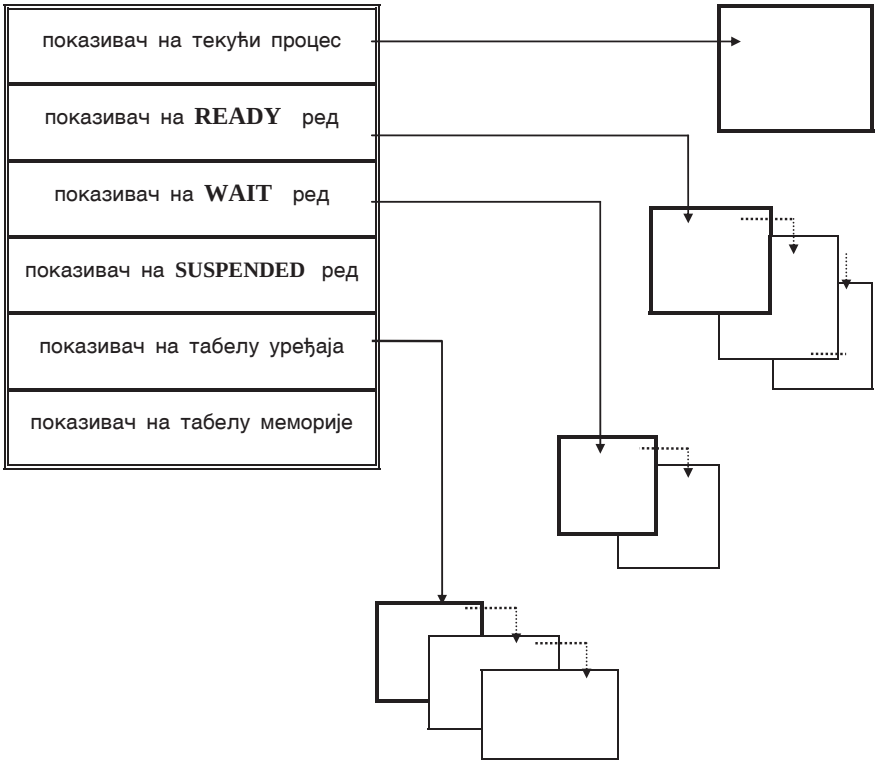
Из свега до сада реченог можемо закључити да се стање појединих процеса у рачунарском систему, као и њихови међусобни положаји и односи у току њиховог боравка у систему непрекидно мењају.

9.2.1. ОПЕРАЦИЈЕ НАД ПРОЦЕСИМА

Оперативни систем може над процесима да извршава бројне функције. Већ смо поменули: креирање, распоређивање, додељивање, претпражњење, блокирање итд.

Да би све ово извршио, део оперативног система мора бити резидентан у меморији. У овај резидентни део, језгро, укључени су диспечер и разне табеле потребне за управљање процесором и меморијом.

На слици 9.4. приказана је једна табела која се зове **централна табела** или системски контролни блок (**SCB, System Control Block**).



Слика 9.4. Контролни блок система

SCB садржи информације о свим процесима који се тренутно налазе у систему. Већина тих информација је у форми показивача на разне редове у табеле. За свако могуће стање постоји по један ред. Пошто су ови редови реализовани као повезане листе, оперативни систем може брзо да среди садржај било ког реда без дуплирања података (смештених на другом месту).

Диспечер током свог извођења мора утврдити који је процес најпогоднији да му се додели процесор. То је, међутим, обично решено тако што диспечер испита **процесорски ред**, односно узме из тог реда први процес. Диспечер долази до појединих процеса, тј. до процесорског реда, преко централне табеле која је меморисана увек на истој фиксној адреси. У централној табели диспечер налази адресу процесорског реда, и приступа му. Први процес у том реду добиће право извођења.

Ред се формира тако да је први процес у реду онај који има навећи приоритет. Приоритет процесу додељује део оперативног система који називамо **управљач процесора**, или главни управљач процесора (**high level schedular, master schedular**).

Последњи процес у процесорском реду назива се **null-process**, и служи само за ознаку завршетка реда. Он има најмањи приоритет, а може представљати и неку функцију коју је пожељно, с времена на време, извести (као што је на пример неки тест програм).

9.2.1.1. Креирање процеса

Сваки процес у рачунару мора имати јединствен идентификатор. Овај идентификатор може бити име, број итд. **Unix** оперативни систем за сваки нови процес генерише јединствен број, *идентификатор процеса* (**PID, Process Identifier**). Када је процес формиран, његово име или **PID** смешта се у **SCB** заједно са показивачем на табелу звану контролни блок процеса (**PCB, Process Control Block**).

9.2.1.1.1. КОНТРОЛНИ БЛОК ПРОЦЕСА

Контролни блок процеса (слика 9.5.) садржи информације различитог типа. Ове информације генерише сам оперативни систем. Стање процеса указује да ли је процес спреман, блокиран, суспендован или се управо изводи. Оперативни систем користи ове информације и чланове одговарајућег реда да би одредио који су задаци на располагању за доделу CPU (спремни), који чекају на неке спољне догађаје (блокирани), а који су кандидати за реактивирање (суспендовани).

Како је адреса следеће инструкције увек у бројачу наредби (PC), и како рачунари са једном CPU имају само један регистар PC, њега користи текући програм који се управо извршава, то у PCB табели сваког процеса мора бити запамћена адреса следеће наредбе за тај процес, тј. садржај његовог програмског бројача (**PC**) у тренутку када је процес изгубио контролу над процесором.

Иста је ситуација и са осталим регистрима у CPU, јер они садрже информације потребне за исправан наставак извођења процеса. Вишепрограмски системи морају имати начине да осигурају да процес неће случајно користити меморију додељену другом процесу.

Обично се памте апсолутна горња и доња граница додељене меморије, или базна адреса и број бајтова којима се може приступити.

У самом CPU постоји хардвер који обезбеђује заштитни механизам адресирања меморије на бази ових података из PCB. И не само ова, већ и многе друге функције оперативног система подржане су одговарајућим хардвером ради веће брзине обраде.

стање процеса
име процеса или PID
бројач наредби
регистри
. . .
. . .
приоритет
границе меморије
списак отворених датотека
информације о стању U/I

Слика 9.5. Контролни блок процеса

Неки PCB садрже и показиваче на листе датотека које процес користи, као и информације о стању улазно–излазних уређаја чије услуге су му потребне. Испитивањем овог поља оперативни систем лако може утврдити зашто је процес блокиран.

Многи системи при креирању процеса одређују и његов приоритет. Овај податак користи, касније, оперативни систем када одлучује који задатак да активира. Приоритет процеса зависи од много чинилаца, а постоји више различитих алгоритама за његово одређивање. Код неких оперативних система приоритет процеса се одређује при формирању процеса и током извођења се не мења, док се код других мења.

9.3. ДОДЕЉИВАЊЕ ПРОЦЕСОРА

Након што је процес формиран, он чека на приступ централном процесору. Диспечер врши избор процеса из процесорског реда који ће добити контролу над процесором. Но, како диспечер мора бити врло кратак, да не би трошио сувише процесорског времена, то овај поступак мора бити крајње поједностављен.

Већина диспечера то ради тако што узме следећи процес из процесорског реда. Како је и сам диспечер процес, то он мора вратити контролу процесора пре него почне извођење изабраног (следећег) процеса.

У основи сваког вишепрограмског рада је механизам прекида. Обрада прекида се делом обавља хардверски а делом софтверски. Почетна фаза обраде прекида увек подразумева спасавање садржаја програмског бројача (PC) и регистра стања (SR) или PSW, ради повратка у прекинути процес. Чување садржаја осталих регистара CPU код већине рачунара није одмах неопходно, јер процесор не користи исте регистре у корисничком режиму као у системском моду.

Контрола над процесором се онда предаје диспечеру.

Сам диспечер мора обавити следећих неколико једноставних корака:

- *прочитати из процесорског реда следећи процес, и утврдити треба ли наставити текући процес или неки други, па ако треба наставити текући процес, онда вратити контролу извођења на адресу коју је сачувао (спасио) прекидни механизам,*
- *ако треба наставити неки други, нови процес, потребно је прво сачувати околину текућег процеса у његовом контролном блоку,*
- *затим треба напунити околину новог процеса,*
- *коначно, треба пребацити контролу извођења у нови процес, на оно место где је извођење претходно било прекинуто.*

Као што је евидентно из овог приказа, након сваке обраде прекида (извођења сервисне рутине) потребно је укључити диспечер да би се наставио неки од процеса.

Диспечер се позива и када процес тражи од оперативног система. улазно–излазне услуге. Текући процес се блокира и контрола се пребацује на оперативни систем.

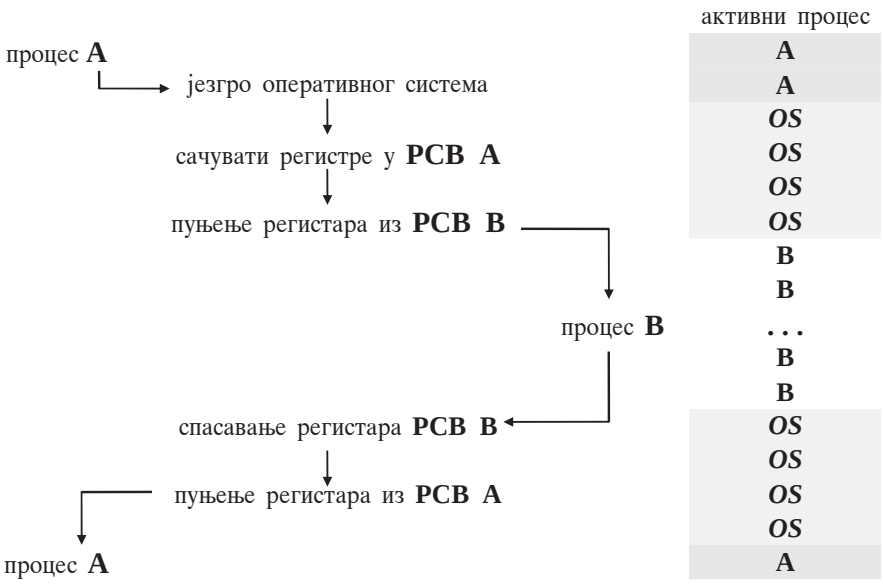
Слика 9.6 показује шта се догађа у рачунарском систему при промени процеса који добија контролу над централним процесором.

Процес **A** се блокира, језгро оперативног система спасава садржаје регистара из процеса **A**, и пуни регистре у **CPU** подацима везаним за процес **B**. Пошто процесор увек узима следећу инструкцију са адресе на коју указује бројач наредби, то ће следећа бити прибављена инструкција из процеса **B**. Слично се догађа након блокирања процеса **B**.

Овај скуп операција се понавља сваки пут када дође до блокирања процеса. То се догађа и код корисничких и код системских процеса. Овај процес се понавља и при позивању диспечера и када се активира процес који је диспечер изабрао.

Када је процес блокиран у PCB се додатно мења садржај поља **стање процесора**, као и неке друге информације везане за стање улаза–излаза, датотека и слично.

Суспензија процеса тражи додатне модификације PCB. Због поновне доделе меморије морају се решити проблеми повезивања, тј. израчунавања адреса локација.



Слика 9.6. Пребацивање са једног процеса на други

Уништавање процеса је последња модификација и подразумева уклањање свих информација о процесу из **SCB**, и свих других табела, и уништава се његов **PCB**. Меморија и датотеке које је користио процес, враћају се систему и ослобађају се сви коришћени излазни и улазни уређаји, једино остају трагови процеса у делу оперативног система који води рачуна о евиденцији коришћења рачунарског система.

9.4. МЕХАНИЗАМ ПРЕКИДА

Рачунарски систем са једним CPU може да извршава само један процес у једном тренутку времена. Оперативни систем дели ресурсе тако да омогући корисничким процесима што је могуће боље коришћење CPU. Ситуација унутар рачунарског система често тражи да се извођење корисниковог програма прекине и да се укључи неки контролни програм из састава

оперативног система. Таква ситуација настаје онда кад се деси неки догађај у рачунарском систему или изван њега, који изазове такву ситуацију у систему која мора одмах бити разрешена. Догађај који изазива прекид је временски непредвидив, и он генерише сигнал који није синхронизован ни са извођењем текућег програма, ни са другим активностима унутар рачунарског система. Прекид можемо дефинисати као несинхронизован, непредвиђени догађај који се појављује у рачунарском систему у виду посебног сигнала који хардвер рачунарског система може да региструје, а који тражи неодложно решавање од стране оперативног система.

Покушајмо да пронађемо сличну ситуацију у свакодневном животу, и начин на који је обично решавамо. Појава сигнала прекида може се упоредити са ситуацијом која настаје када читамо књигу, и у неком тренутку зазвони телефон, што представља догађај на који морамо одмах реаговати. У том тренутку неопходно је да прекинемо читање књиге, обавимо телефонски разговор и након тога наставимо даље да читамо тачно онде где смо прекинули читање. Телефонски позив очигледно може доћи у било ком тренутку читања и тражи моментални одговор.

Из овог примера је очигледно да ситуацију која настаје у систему карактеришу следећи кораци: прекид текућег програма, пребацивање контроле система у неки програм оперативног система, и поновно враћање контроле извођења у корисников програм.

У опису преноса улазно–излазних података имали смо прилике да се упознамо са једном од могућих ситуација када треба прекинути извођење корисниковог програма. Кориснички процес захтева помоћ за обављање **U/I** преноса, и шаље специјални позив оперативном систему познат као захтев за прекид (**interrupt**). У **IBM** системима он се зове SVC (**Supevisor Call**). **Unisys 1100** рачунари користе извршни захтев **ER** (**Executive Request**). **Intel 80x86** фамилија микропроцесора користи **INT** инструкцију итд. Врста помоћи која се тражи од оперативног система зависиће од типа рачунара, али код већине је то обављање улазно–излазних операција, управљање меморијом и неке специјалне функције. Но прекиде могу изазвати и фактори изван извршног процеса. То су разне хардверске грешке, сигнали са улазно–излазних уређаја, тајмер и сигнали изван рачунарског система.

Да би се прекиди могли обрадити, тј. да би проблемски програм био стварно прекинут у извођењу и контрола извођења пребачена на први корак, а то је анализа прекида, а затим на други корак, а то је обрада прекида, морају се у рачунарски систем увести нови хардверски и софтверски делови који ће то омогућити.

Могућа су различита решења, али најчешће само пребацивање контроле извођења обавља хардверски механизам, док анализу прекида изводи део оперативног система који се зове анализатор прекида. Само решавање

ситуације која је настала појавом прекида врши део оперативног система који се зове програм за обраду прекида. То, међутим, није увек случај. Анализа прекида може бити пребачена у хардверски део система, јер то обезбеђује знатно већу брзину обраде прекида, али и мању флексибилност и знатно скупљи процесор.

9.4.1. ВРСТЕ ПРЕКИДА

Сваки рачунарски систем има свој скуп, и систематизацију прекида, али се у већини система могу поделити у пет врста и то:

– *излазно–улазни прекиди* (**Input/Output interrupt**),
– *програмски прекиди* (**program interrupt**),
– *прекиди због позива "супервизора"* (**supervisor call interrupt**),
– *спољни прекиди* (**external interrupt**),
– *машински прекид* (**machine check interrupt**).

Излазно-улазни прекиди потичу од разних улазно–излазних уређаја, канала и контролера. Под њиховом контролом најчешће се полу–аутономно обављају U/I активности. Ови склопови шаљу своје захтеве за прекид асинхроно, суспендује се текући процес, канали посредством оперативног система добијају потребне информације и настављају да аутономно контролишу U/I операције, а оперативни систем враћа контролу прекинутом процесу. Најчешће је то сигнал да је завршена улазно–излазна операција. Међутим, ако нека улазно–излазна операција није извршена, улазно–излазна јединица такође сигнализира прекид ове врсте. Улазно–излазни прекиди се користе и када треба указати на грешке у подацима које су детектовали канали или уређаји.

Програмски прекиди настају током извођења програма. Током извођења програма, проверава се да ли су нарушени одређени услови: дељење нулом, препуњење, подкорачење, итд., а ако јесу систем генерише прекид. До прекида програма може доћи ако се утврди да операциони код не спада у важеће кодове које централни процесор декодира као инструкцију, дакле на месту у меморији, са које је садржај пренесен, није била уписана инструкција. Програмски прекид је прекид који је проузроковао сам програм који се изводи.

У оквиру оперативног система постоји скуп програма чији је задатак обрада прекида и контрола свих осталих програма. Тај скуп програма се назива супервизор. Програми који не улазе у састав супервизора (тзв. променљиви програми) могу да се обраћају супервизору помоћу специјалне команде и тада настаје супервизорски прекид. Овај прекид може да буде изазван и техничким системом рачунара.

Модерни рачунари раде или у корисничком или у системском режиму. Због безбедности и контроле, већина система забрањује да процес у корисничком режиму извршава улазно–излазне инструкције директно на хардверу. У ствари, процес мора да генерише једну **SVC** инструкцију односно прекид, и да захтева од оперативног система да изврши **U/I** операцију.

Системи за заштиту меморије се такође реализују у системском режиму. Кориснички процес мора остати унутар граница које одређују његови гранични регистри. Сваки покушај да се изађе из овог опсега изазваће генерисање прекида и оперативни систем ће бити позван да разреши конфликт, јер оперативни систем ради у системском режиму и доступне су му све адресе, сви регистри и све табеле.

Спољни прекиди имају свој извор изван текућег рачунарског система. У ову групу спадају и прекиди које изазива оператер са своје конзоле или који долазе из других **CPU**. То може бити, на пример, због захтева оператера да преда неку команду оперативном систему, или због прекида са комункицијске линије. У ову врсту прекида спадају и прекиди које генеришу уређаји за мерење интервала времена. **RESTART** је пример спољнег прекида, а активира се притиском на дугме које у разним системима има разна имена (**reset, Control–Alternate-Delete** на **IBM-PC**). **Restart** изазива ресетовање рачунара на почетне услове, па ће самим тим бити изгубљено све што је било у главној меморији – програми и подаци.

Машински прекиди обавештавају о проблемима насталим у систему, односно, када се у систему детектује хардверска грешка. Систем контроле меморије може, на пример, открити и обавестити нас, о нарушеној парности. Систем за контролу напајања може открити престанак напајања и укључити резервно напајање.

9.4.2. ОБРАДА ПРЕКИДА

Сви прекиди се у основи третирају на исти начин. Текући процес губи контролу над **CPU** и активира се специјални програм који се зове **анализатор прекида** (**interrupt handler**). Када дође до прекида морамо сачувати све податке који су потребни за евентуални, каснији, наставак прекинутог програма. Ови подаци се налазе у регистрима CPU (регистар стања **SR** и бројач наредби **PC**, или у једном регистру као што је **PSW**) Кад је прекид обрађен, морамо поново обновити садржај који је био у овим регистрима непосредно пре прекида. Пошто бројач наредби садржи адресу следеће инструкције коју треба извести, централни процесор ће тачно узети следећу инструкцију као што би то учинио да прекида није ни било.

Ситуацију опет можемо упоредити са примером из нашег свакодневног живота, када смо прекинути у читању књиге. Када зазвони телефон ми упамтимо докле смо у читању дошли. То може бити, на пример, страница књиге, пасус, ред или реч у реду. То није ништа друго него адреса која нам означава где ћемо, након завршетка телефонског разговора, који овде представља прекид, наставити читање. Такође, морамо знати, или боље, рећи упамтити, неке друге информације о садржају текста ако желимо несметано одмах да наставимо читање. То је на неки начин опис ситуације пре прекида коју је неопходно сачувати да би се касније могло наставити читање непосредно иза места где се прекид догодио.

Шта ће се догађати у рачунарском систему након обраде прекида, можемо закључити посматрајући наш прекид (телефонски разговор).

Често се догађа да се не враћамо на читање књиге, него због неког разлога (садржаја телефонског разговора, истеклог времена или слично) започињемо или настављамо неки други посао, који, опет, или довршавамо до краја или прекидамо, и тако редом све док се након неког времена поновно не вратимо на читање. Опет позивамо у сећање страницу, пасус, ред и реч где смо читање прекинули, присећамо се ситуације, дакле регенеришемо локацију прекида и стање пре прекида, и настављамо опет читање до следећег прекида или до краја књиге.

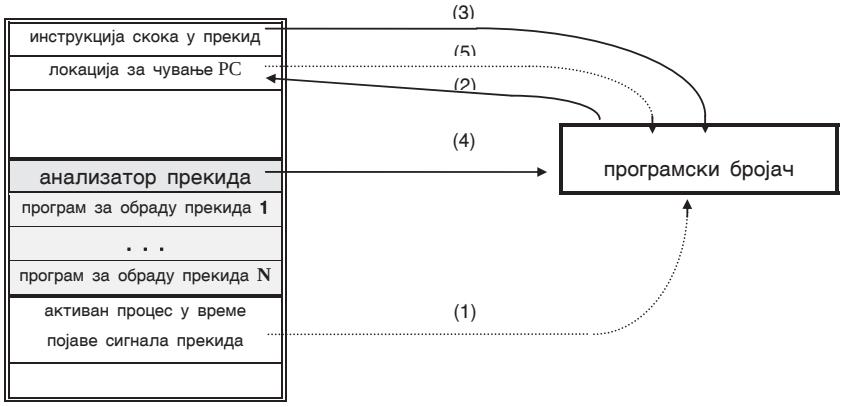
Можемо закључити, дакле, да није нужно да се након обраде прекида вратимо на обраду прекинутог програма. У сваком случају, морамо бити у стању да обновимо ситуацију која је била у систему непосредно пре прекида, а то је адреса наставка и стање пре прекида. То значи да смо те податке морали да упамтимо, те да смо у стању да их позовемо и сместимо на потребно место.

Преношење контроле извођења након прекида на неки други програм, затим трећи, итд., да би се тек након извесног броја таквих извођења вратили на први програм, није ништа друго него мултипрограмирање. Мулти–програмирање се, дакле, базира, такође, на обради прекида.

Неки рачунари имају једну локацију у меморији која подржава све врсте прекида. Ова локација у ствари садржи инструкцију скока на адресу на којој се налази анализатор прекида.

На слици 9.7 приказана је ситуација која настаје у тренутку генерисања прекида.

Програмски бројач (**PC**) указује на следећу инструкцију текућег процеса (1). Садржај бројача **PC** се смешта на унапред одређену локацију (2). Бројач наредби (**PC**) се пуни новом адресом, адресом специјалне инструкције прекида (3). Током следећег циклуса прибављања у **CPU** се преноси специјална инструкција скока у прекид.



Слика 9.7. Шематски приказ тока извршавања инструкција и измена садржаја бројача наредби током обраде прекида

Извршавање ове инструкције има за последицу пуњење бројача наредби тако да указује на почетак анализатора прекида (4). Након тога отпочиње извођење анализатора прекида, који мора сачувати садржај сваког регистра CPU који ће користити, одредити извор прекида и обрадити га на одговарајући начин.

Излазак из анализатора прекида изводи се инструкцијом повратка из прекида која аутоматски пуни бројач наредби (5) вредношћу која је сачувана када се десио прекид (следећа инструкција прекинутог процеса).

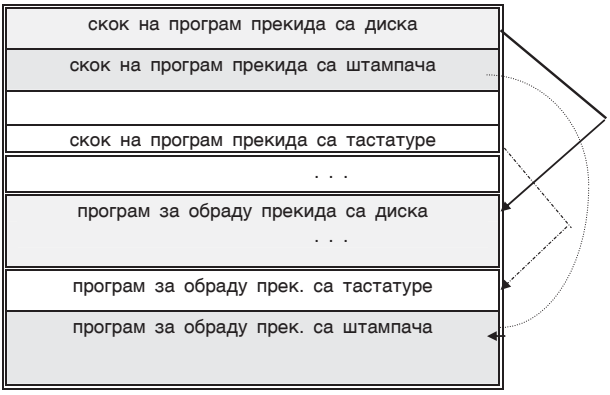
Други рачунари имају такозвани векторски прекид (**vectored interrupt**), приказан на слици 9.8. Овај приступ користи разне адресе за сваку врсту прекида. Као и у претходном примеру, свака адреса садржи инструкцију која изазива скок у анализатор прекида. Овај приступ захтева од уређаја који је генерисао захтев за прекид да пошаље адресу одговарајућег анализатора прекида. Садржај програмског бројача се мора сачувати у **стеку**, јер сада не може постојати унапред одређена локација предодређена за ту функцију.

Но, да не би дошло до забуне, и овде постоји анализатор прекида, само је он реализован хардверски, чиме се постиже већа брзина обраде прекида, што је понекад од огромног значаја.

Дакле, да резимирамо: постоје два начина обраде прекида:

– *први, код којег се сви прекиди исто започињу, а после се одређује тип прекида и њему одговарајућа конкретна рутина за обраду,*

– *други, код којег се већ при настанку прекида зна о којем је прекиду реч, и код којег се одмах иде на рутину за тај тип прекида. Овај тип прекида се често назива векторски прекид.*



Слика 9.8. Векторски прекид има посебну наредбу скока за сваку врсту прекида

Но, која год да је врста прекида у питању, најпре се мора спасити садржај наведених регистара (**PC, SR**, тј. **PSW**), а касније и садржаји свих других регистара које ће рутина за обраду прекида користити. Садржаји ових регистара се смештају у неке унапред одређене локације у меморији или у стек. У регистре **PC**, **SR**, или **PSW** се уписује нови садржај, тј. адреса рутине за обраду прекида (или адреса анализатора прекида), а један од задатака ове рутине је да сачува садржаје и осталих регистара која ће она користити, и које ће на крају свог извођења морати поново да упише у регистре **CPU**. По завршетку обраде прекида враћа се сачувани садржај регистара, а прекинути програм наставља са извођењем тачно на оном месту где је прекинут, односно извршава се следећа инструкција прекинутог програма.

Дакле, постоје хардверски склопови који препознају какве је врсте прекид. У општем случају, међутим, под механизмом прекида подразумева се само измена активног садржаја бројача наредби и регистра стања (односно **PSW**–а), тако да се њихов текући садржај пребацује негде у меморију, а уписују се нови садржаји ових регистара који постају активни, чиме се активира програм за обраду прекида. Сама рутина за обраду прекида мора се састојати од два дела. Први део је анализатор прекида који утврђује које је врсте прекид, а затим се према врсти прекида активира други део који служи за саму обраду прекида. При томе анализа прекида може бити решена софтверски, или као код система **IBM/370** где је анализа прекида решена хардверски.

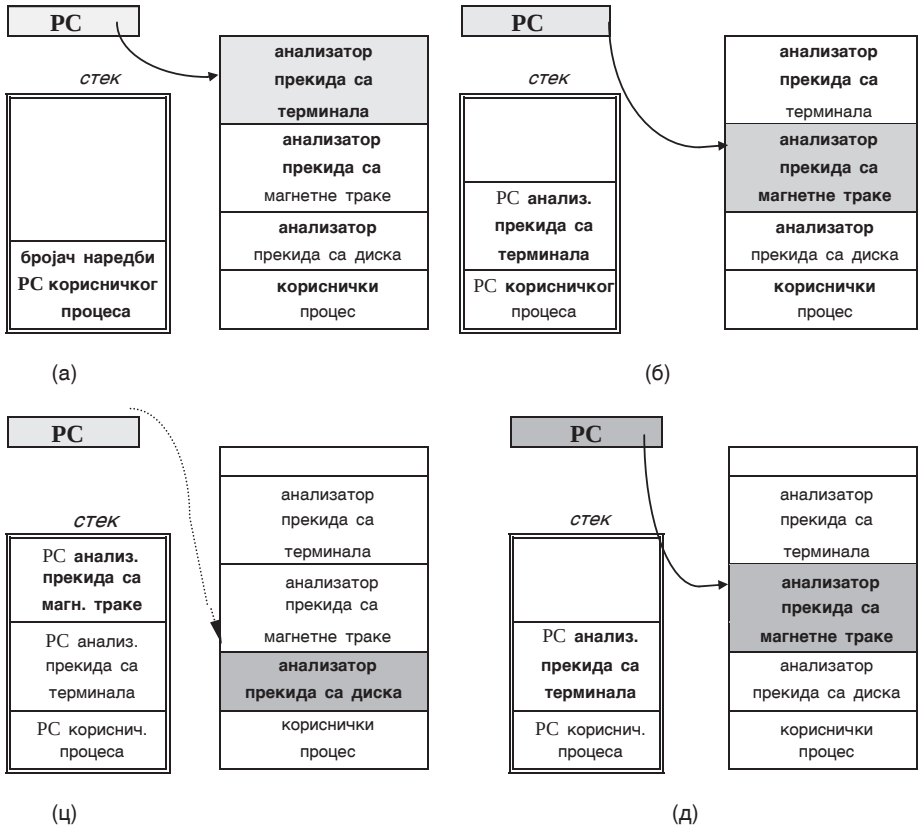
Да ли у току обраде једног прекида може да се јави други прекид? Многи системи не допуштају обраду нових прекида док траје обрада једног прекида. Но да ли то увек треба чинити? НЕ, јер погледајмо опет једну реалну ситуацију. Док смо читали књигу зазвонио је телефон. Прекидамо читање књиге и одговарамо на позив. Али у току разговора неко звони на вратима. Највероватније је да ћемо за тренутак прекинути разговор и видети ко је пред вратима. Дакле прекинућемо програм којим се обрађује први прекид и извршити неки други програм. Шта ће се десити када дође други прекид у току обраде првог прекида по поступку са слике 9.7.?

Ако је систем аутоматски сачувао садржај бројача наредби, и у њега напунио нову адресу, адресу специјалне инструкције скока у прекид, анализатор прекида почиње са извођењем. Ако би и сам анализатор био прекинут, његов бројач наредби би био пренет у ону исту специјалну локацију и садржај бројача наредби основног прекинутог процеса био би изгубљен. Да би се оваква ситуација онемогућила, анализатор прекида мора бити у могућности да спречи настанак неких или свих прекида. Овај поступак се зове маскирање (**masking**). Маскирање се, такође, може користити за задавање нивоа приоритета појединим врстама прекида, што има великог смисла.

Посматрајмо прекиде који долазе, један за другим, са терминала, траке и диска. Како је пренос са диска врло брз, диск мора хитно бити опслужен или ће подаци бити изгубљени. Мању брзину има трака, док је терминал изузетно спор, па може да сачека. На слици 9.9. (а) приказано је стање меморије, бројача наредби и стека, након што је прекид са терминала препознат. Пошто је терминал спор, анализатор прекида може бити прекинут захтевом са траке слика 9.9. (б). Уочимо да сада стек садржи адресе следећих инструкција корисничког програма, и прекинутог анализатора прекида. Обрада траке може бити прекинута захтевом са диска, слика 9.9. (ц). Када се заврши обрада диска, посао наставља прекид са траке. Бројач наредби се пуни адресом анализатора прекида траке са стека, слика 9.9. (д). Када се до краја изврши програм за обраду прекида са траке, прелази се на довршавање обраде прекида са терминала и на крају се враћамо у кориснички програм.

Анализатор прекида са терминала би требало да постави маску којом спречава да га прекине неки нови захтев за прекид са терминала, али не и са других јединица. Анализатор прекида са траке би требало да постави маску против прекида и са траке и са терминала, док диск рутина треба да се заштити (помоћу маске) од прекида са све три јединице. Неки прекиди су тако важни да не смеју бити маскирани. Систем мора бити способан, на пример, да детектује губитак напајања и мора спасити важне информације о стању. Пошто је расположиво време од тренутка губљења напајања до престанка рада рачунара врло мало, то рутина за обраду овог прекида не сме бити блокирана од стране других рутина.

Ситуације оваквог типа су подржане другом врстом прекида који се зове немаскирани прекид који, као што му име каже, не може бити блокиран.



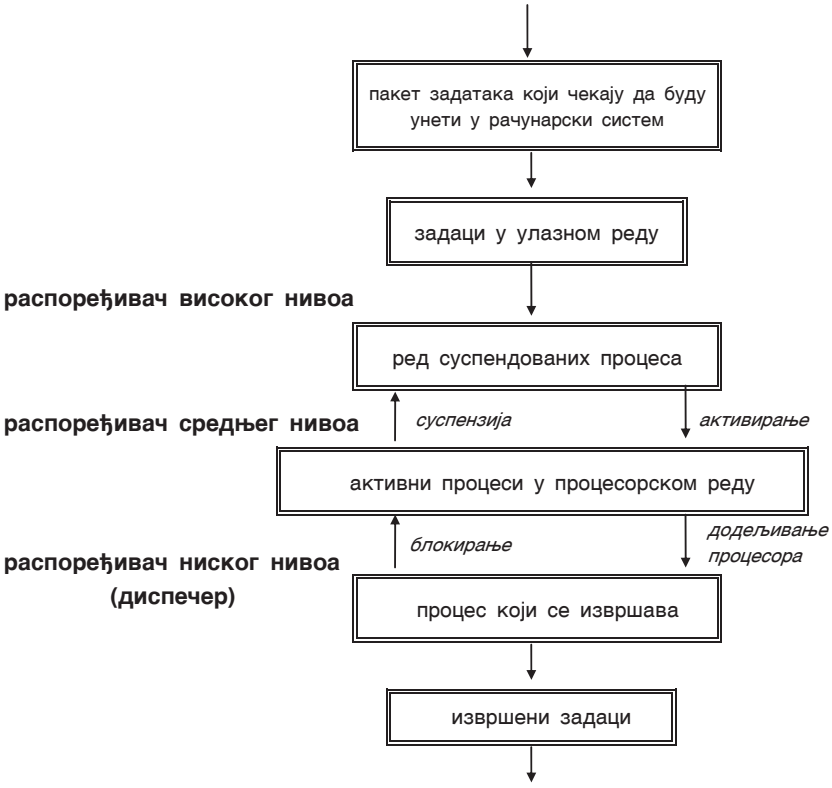
Слика 9.9. Анализатор прекида може бити прекинут уколико је адреса повратка у програм запамћена

9.5. РАСПОРЕЂИВАЊЕ РЕСУРСА - ПЛАНИРАЊЕ

Распоређивање, односно планирање процеса може се поделити на три нивоа, као на слици 9.10. Највиши ниво је повезан за отпочињање пакетних процеса. Задаци прихваћени на овом нивоу налазе се у стању које је слично суспендованим задацима. Они су препознати од оперативног система, али им се не допушта да се такмиче за ресурсе.

Пакетни систем обично захтева од корисника да предвиди ресурсе потребне за његов процес. Распоређивач високог нивоа може да користи ове

информације, да изабере добру мешавину процесорски ограничених и улазно–излазних ограничених задатака.



Слика 9.10. Главни нивои распоређивања

Распоређивач ниског нивоа (диспечер) непосредно одређује који ће задатак из процесорског реда (реда спремности) добити процесор.

9.6. УПРАВЉАЊЕ МЕМОРИЈОМ

Први рачунарски системи су били програмирани директно у машинском језику. Ови рачунари допуштали су присуство само једног програма у меморији и он је располагао, за све време свог извођења, процесором, читавом меморијом и свим периферијским јединицама. Програмери су морали савршено да познају читав хардвер у саставу рачунарског система и радили су без окружења које чини оперативни систем. Но врло брзо су програмери реализовали скуп програма који подржавају улаз и излаз, и тако је настао

главни део раних оперативних система који се зову монитори или **Input /Output control system (IOCS).** Али оперативни системи су се убрзано развијали и усложњавали, а једна од основних функција оперативног система, на којој је базиран вишепрограмски рад, јесте и управљање меморијом.

У рачунарском систему разликујемо два нивоа меморије, или две врсте меморије. То су централна меморија (**main memory, operating memory, core memory, central memory, main storage**) и екстерне меморије (**external memories, auxiliary storage, secondary storage, mass storage**). Екстерних меморија има разних врста – у виду разних јединица (нпр. магнетни дискови, траке, оптички дискови, итд.). У сваком рачунарском систему постоји само једна централна меморија. По технологији израде и централне меморије могу бити врло различите (феромагнетна, полупроводничка, ласерска итд.). Под управљањем меморијом (**memory management**) подразумевамо разматрање проблема који се у рачунарском систему појављују у вези коришћења централне меморије. Разматрања везана за проблеме коришћења екстерних меморија сврстана су у управљање подацима и системе датотека (**data management, file systems**).

Са становишта корисника, било би пожељно да систем има само један ниво меморије (**one-level-storage**). То би значило да корисник не мора да води рачуна где се његов програм и подаци налазе, када и како да их позове, меморише, ажурира итд.

Овај концепт је постављен већ у једном од рачунара прве генерације **ATLAS**, који је пројектован на **Manchester University**. Међутим, чак ни у данашњим рачунарским системима то још увек није потпуно остварив. Зато корисник при коришћењу рачунарског система мора да води рачуна о та два нивоа организовања меморије, па чак и о разликама између појединих врста екстерних меморија.

Ипак, у неким системима, и скоро свим програмским језицима, постоје решења – такозвани концепт виртуелне меморије, која такорећи у потпуности омогућавају кориснику коришћење меморије у једном нивоу (BASIC, APL).

Управљање меморијом се непосредно надограђује на модул за управљање процесором и представља следећи слој у хијерархијском моделу оперативног система који се ослања на језгро. Систем за управљање меморијом мора садржавати и обављати следеће четири основне функције:

1. *Потребно је непрестано водити рачуна о слободним и заузетим деловима централне меморије. Да би се генерисао нови процес треба му доделити, поред осталих ресурса, и оперативну меморију. То значи да у сваком тренутку оперативни систем мора бити способан да одговори на питање може ли нови програм бити смештен у меморију, и тако омогућити генерисање новог процеса.*

2. *Потребна је одређена стратегија додељивања меморије (алокација), на основу које се одлучује:*
   - **коме**, тј. којем програму, односно процесу, доделити меморију,
   - **колико** меморије треба доделити,
   - **када** треба доделити меморију, и
   - **где**, тј. који део меморије доделити процесу.
3. *Техника доделе меморије и израчунавање адреса.*
4. *Ослобађање меморије (делокација), тј. избацивање процеса из меморије.*

Управљање меморијом је један од најважнијих делова оперативног система, јер се може изводити само онај програм који се налази у централној меморији, и могу се обрађивати само они подаци који се налазе у главној меморији. Требало би управљање меморијом посматрати повезано са управљањем процесором, јер нема смисла додељивати меморију процесу који не може добити и процесор.

Управљање меморијом мора обезбедити ефикасност коришћења меморије, заштиту меморије од недозвољеног приступа, логичку и физичку организацију меморије, али и једноставност употребе, јер и овај, као и други програми оперативног система који се често користе, не сме трошити много процесорског времена.

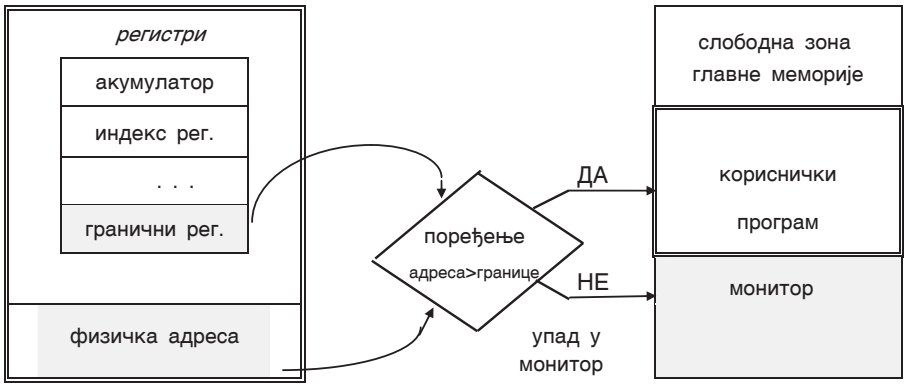
Да би се рачунарски систем што ефиксније користио треба сместити што више процеса у меморију ради што ефикаснијег мултипрограмирања, и оставити што мање малих и неискоришћених делова меморије. који се не могу искористити јер су мањи и од најмањег програма који чека да би био трансформисан у процес. У мултипрограмском режиму рада треба водити рачуна о заштити меморијског простора и података, тј. о заштити оперативног система од појединих процеса, као и о заштити сваког процеса од било каквог приступа у његово подручје од стране другог процеса. Такве се заштите спроводе проверавањем адреса и ограничавањем приступа, као што је, на пример, само читање записа у поједином делу меморије (**read access**), или се допушта само уписивање (**write**), или само позивање на извођење (**execute**), итд.

9.6.1. ЈЕДНОКОРИСНИЧКИ МОНИТОРСКИ СИСТЕМИ

У тренутку укључења система, активира се један мали пунилац из састава оперативног система, који се зове **bootstrap loader**, и чији је задатак да пренесе резидентни део оперативног система са неке перманентне меморије (диск, трака) у оперативну меморију. Овај поступак се зове пуњење или подизање система (**booting**). Тај основни део оперативног система код једнокорисничких рачунара представља монитор, и **bootstrap** пунилац га

обично смешта на један крај меморије. Корисников програм се затим пуни иза монитора, или на други крај меморије.

Када корисничком процесу треба улазно–излазна услуга, он шаље потпрограмски позив IOCS рутини, која обави услугу и враћа контролу програму. Оваква или слична ситуација постоји и данас у већини 8–битних и 16–битних мини и микрорачунара и односи се и на све врсте DOS-а. Да би се ова два дела меморије раздвојила у процесору се налази **гранични регистар** (**bound fence**), чија је улога да заштити део меморије у којем се налази оперативни систем од упада од стране корисничког програма. У овај регистар се записује почетна адреса корисничког програма. Поступак је једноставан, свака израчуната ефективна адреса у корисничком процесу се тестира како би се видело да ли је већа од граничне или не, као што је приказано на слици 9.11.



Слика 9.11. Заштита меморије помоћу граничног регистра

Ова се алокација иначе употребљава код система са групном обрадом (**batch system**), код којих се један процес изводи од свог почетка до свог завршетка, без дељења централног процесора с неким другим корисничким процесом.

На слици 9.11. може се видети да је један део меморије заузет оперативним системом док је у преостали део меморије смештен један једини процес који се изводи, а преостали део меморије је празан, односно неискоришћен.

Неискоришћени део меморије у овом случају може бити већи од искоришћеног дела. Ако имамо малу централну меморију, може се догодити да многи програми у њу не могу да стану. Због тога меморија мора бити довољно велика да бисмо могли користити довољно велике програме, али ће зато увек постојати неки део меморије који ће бити потпуно

неискоришћен. Неискоришћеност централне меморије главни је недостатак ове алокације меморије. Тај недостатак повлачи за собом неискоришћеност централног процесора, јер ће већину времена процесор чекати, а и улазно–излазне јединице ће бити слабо и неједнако искоришћене. Дакле, неискоришћеност меморије има за последицу и неискоришћеност читавог рачунарског система.

Но, и ови системи могу имати могућност слични мултипрограмском раду, при чему блокирани процеси одлазе на резервну, скривену секундарну меморију (**backing store**), на којој се најчешће налази и процесорски ред **(ready queue**). У овом случају треба бити врло опрезан. Наиме, због велике разлике у брзинама рада ових меморија, може се десити да рачунар одвише времена потроши на сувише честа пребацивања са једног процеса на други.

У оваквим системима се ипак могу обрађивати и програми чија је величина већа од оперативне меморије. То се постиже дељењем програма и његових података (који су на диску) на мање делове, који се потом пресликавају на одређена поља у меморији. Ово је такозвана техника преклапања (**overlay**). Али, и овде постоји опасност од значајног успоравања рада рачунарског система због сталног преноса информација између спољних меморија и оперативне меморије.

9.6.2. ДОДЕЉИВАЊЕ МЕМОРИЈЕ У ПАРТИЦИЈАМА

Додељивање меморије у деловима заснива се на дељењу меморије у делове (које називамо партицијама, **partition**). У сваки појединн део меморије смешта се по један процес тако да у меморији добијамо више процеса који могу делити време централног процесора, дакле, омогућава се мултипрограмирање.

Заштита меморије се обавља на хардверском нивоу помоћу два гранична регистра. Један садржи почетну адресу партиције а други крајњу адресу партиције. Приликом адресирања меморије дигитална кола проверавају да ли се тражена адреса налази између почетне и крајње адресе.

9.6.2.1. Додела меморије у фиксним деловима

Код доделе меморије у фиксним партицијама, односно код статичке поделе меморије, величина партиција је унапред одређена, и не може се мењати, па самим тим има статички карактер (иако се у принципу може мењати). Код овакве алокације по правилу се јављају мали неискоришћени делови меморије, фрагменти (јер додељена меморија увек мора бити већа од величине процеса), слика 9.12.

Стање заузетости меморије се прати уз помоћ табеле са следећим подацима: број партиције, величина партиције, почетна адреса (локација) партиције и стање партиције које може бити заузето или слободно. На слици 9.13 је приказан пример такве табеле и одговарајуће заузеће меморије.

Бр.	вел. KB	поч. лок.	стање
1	64	32000	Z
2	16	96000	Z
3	64	112000	S
...			
N	128		

Слика 9.13 Табела меморије

Слика 9.12. Дељење меморије на партиције фиксне дужине

9.6.2.2. Динамичка додела меморије у деловима

Оперативни систем стално надгледа меморију и води евиденцију о слободним и заузетим партицијама. Код система са динамичком доделом меморије, величина партиција, као и њихова почетна локација нису фиксне већ их оперативни систем лако може мењати. Величина партиција се одређују по величини и према броју долазећих процеса. На тај начин свака партиција постаје велика колико и процес који треба у њу сместити. Сви остали делови меморије чине неалоциране партиције. Оваква подела меморије елиминише многе мане статичке алокације, а пре свега појаву великог броја малих фрагмената.

Основне предности алокације меморије у партицијама у односу на појединачну континуирану алокацију меморије су следеће:

- *омогућава мултипрограмирање и тиме ефикасније коришћење целокупног рачунарског система.*
- *није потребан никакав специјални хардвер за спровођење било које верзије ове алокације.*
- *алгоритам за алокацију и деалокацију меморије је врло једноставан.*

Алокација меморије у партицијама има и низ недостатака од којих су најзначајнији:

- *неискоришћеност меморије може понекад бити врло велика, што зависи од редоследа и величине процеса и алгоритама за алокацију који се користе,*
- *по правилу долази до појаве стварања малих неискоришћених простора у меморији–фрагментација,*
- *програм за управљање меморијом заузима део меморије и генерише знатно већи интерни рад него што је то код мониторских система,*
- *у меморији се налазе и делови процеса који се тренутно не користе, или се више неће ни користити.*

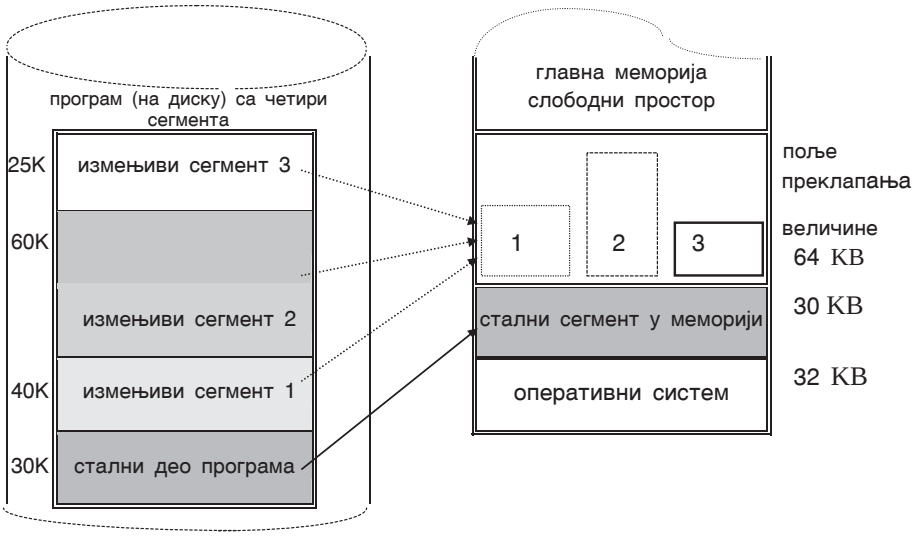
Проблем фрагментације се може решавати тако што се сви појединачни фрагменти сакупе у једну велику целину. Тај поступак треба обавити сваки пут када се у систем унесе нови процес, јер се тада увек ствара и нови фрагмент. Да би се то остварило понекад је потребно померити (релоцирати) већ постојеће процесе у меморији са једне почетне адресе на другу. Физичко помицање процеса, да би се у меморији ослободио простор у виду једне јединствене партиције, називамо збијање – **компакција**, јер се ради и о збијању постојећих процеса у једну целину. Основни недостатак је ипак у томе што се не могу обрађивати програми већи од величине оперативне меморије, тј. употреба рачунарског система се ограничава на мање процесе. Овај проблем се решава применом технике преклапања.

9.6.3. ТЕХНИКА ПРЕКЛАПАЊА

Технике укрупњавања слободних делова меморије (компакција) нису, међутим успеле да проблем фрагментације потпуно отклоне, а проблем обраде програма већих од оперативне меморије остао је нерешен. Зато су људи кренули у решавање ових проблема на супротан начин; уместо да укрупњавају партиције процеси се разбијају на делове, које зовемо сегментима, а затим се ти сегменти пуне у слободне делове меморије. Та се техника назива **техника преклапања** (**overlay technique**), слика 9.14. Техника преклапања се употребљава да би се врло велики програми могли изводити у рачунарима са малом оперативном меморијом.

Техника преклапања омогућава смештање програма у мањи меморијски простор, него што је сам његов адресни простор. Дакле, срећемо се са могућношћу да је адресни простор програма већи од меморијског простора. Обрађени део програма и података се избацује из оперативне меморије а у њу се убацује, на исто место, други део програма (података). Један део програма и података може бити стално у меморији. Програм се дели на

сегменте који представљају логичке, функционалне целине. Овај део посла некада су обављали сами програмери.



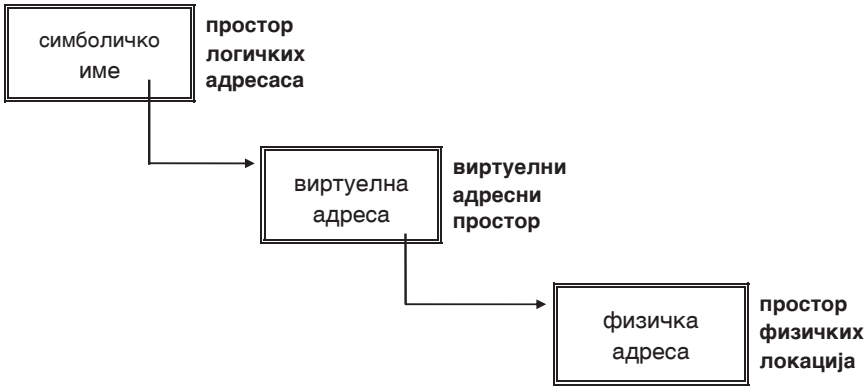
Слика 9.14. Дељење програма на делове и коришћење заједничке меморије (преклапање измењивих сегмената програма)

9.6.4. СИСТЕМ ВИРТУЕЛНИХ МЕМОРИЈА

Бољи оперативни системи допуштају смештање активних делова програма (и података) на било који слободан простор у главној меморији, док инструкције и подаци који нису више у текућој употреби ослобађају меморију, и враћају се на диск. Оперативни системи са концептом виртуелне меморије омогућавају програму да буде неограничено велики, већи од физичког капацитета меморије рачунара. Овај концепт подржава логички изглед програма као да је смештен у непрекидан простор (континуално). Физичке локације програма могу бити врло различите од логичких, које су предвиђене у програму. Превођење корисниковог виђења програма у физичку стварност обавља оперативни систем са виртуелном меморијом (такав је IBM–ов оперативни систем MVS, или Digital–ов VMS), као што је приказано на слици 9.15.

Програмер у току писања програма користи симболичка имена променљивих, тј. користи простор логичких адреса. Сем у машинском језику, стварне физичке локације нису од значаја за програмера.

У току фазе превођења, имена променљивих се транслирају у адресе које чине виртуелни адресни простор. Виртуелне адресе нису физичке адресе, већ се физичке адресе одређују у току пуњења програма, и оне чине физички простор или простор локација.



Слика 9.15. Везе између простора логичких адреса, виртуелног адресног простора и простора физичких локација

9.6.4.1. Израчунавање виртуелне адресе

Систем виртуелне меморије смешта инструкције и податке у блокове меморије. Поступак познат као **страничење (paging)** користи блокове једнаке дужине, док систем познат као **сегментација** (**segmentation**) користи блокове променљиве дужине. **Виртуелна адреса инструкције**, **V** (или података), је комбинација њене адресе, тј **броја блока** (**b**) и адресе локације у блоку, тј. **помераја** (**d**, **displacement**),

$$\mathbf{V} = (\mathbf{b},\mathbf{d})$$

Ако неки рачунар користи блокове од 1000 бајта по страници, онда преводилац конвертује адресу 3092 у блок 3 и померај 92. Адреса податка на локацији 327 транслира се у блок 0 и померај 327. Оперативни систем је одговоран за пуњење блокова у примарну меморију, он чува неки траг о томе који блокови су напуњени, а који су на секундарној меморији. Оперативни систем обавља ове функције посредством табела које се зову **мапе блокова**, које указују на локацију сваког блока програма, слика 9.16.

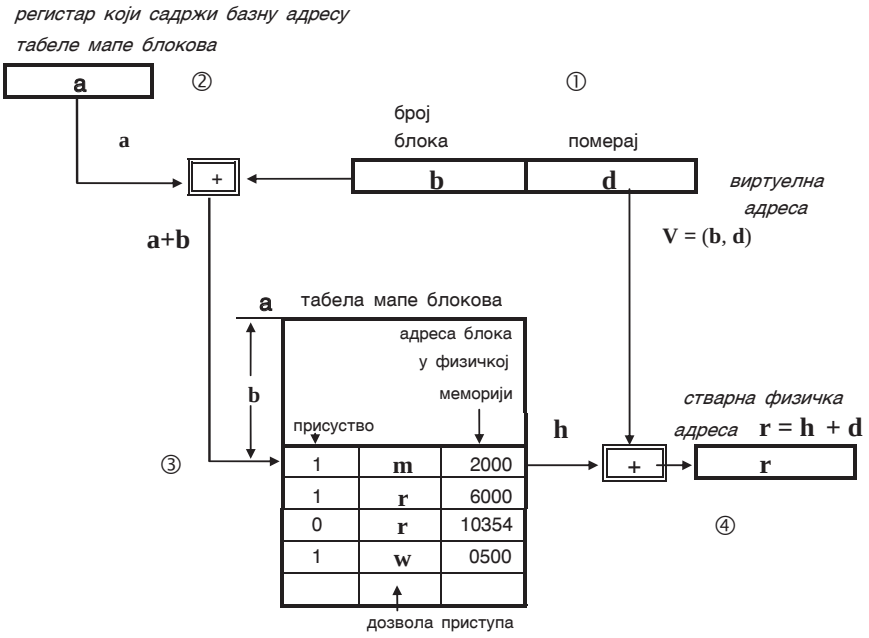
Колона присуства (**presence**) означава да ли је блок у примарној меморији (јединица) или у секундарној (нула). Ако је у примарној, онда податак у пољу адресе указује на физичку адресу блока (први блок почиње на локацији 2000). Ако блок није у примарној меморији, онда та адреса указује

на локацију у секундарној меморији, и то би у овом случају био 103 цилиндар, површина 5, сектор 4 на скривеном диску. Колона дозвола се користи у шемама за заштиту меморије, и означава које су операције над тим локацијама дозвољене; **r** означава да је дозвољено само читање (**read**), **w** упис (**write**), а **m** означава да се тај блок може и модификовати (читати, уписивати и мењати постојећи садржај).

присуство	*адреса*	*дозвола приступа*
1	2000	m
1	6000	r
0	10354	r
1	0500	w

Слика 9.16. Једноставна табела мапе блокова

Израчунавање физичке адресе на основу виртуелне, врши се уз помоћ базних адреса табеле мапе блокова које се налазе у контролном блоку сваког процеса, као што показује слика 9.17.



Слика 9.17. Израчунавање физичке адресе

Поступак израчунавања тече следећим редоследом:

1. *Програм који се изводи тражи податак са локације*: **V = (b,d).**

2. *Систем додаје број блока,* **b**, *на локацију своје табеле мапе блокова*, **a**, *и израчунава померај* , **a+b**, *у табели мапе блокова*.
3. *Чита се садржај са израчунате локације у табели, и ако је блок у примарној меморији чита се стварна адреса блока* **h.**
4. *Адреса блока* **h** *додаје се на померај* **d** *из виртуелне адресе, и одређује се физичка адреса*, **r = h+d .**

Напомена: Ако блок није у главној меморији, он се мора најпре напунити, пре него што се процес настави.

Да би овај поступак био могућ потребан је одговарајући хардвер, тј. систем мора имати специјални регистар почетка табеле блокова, који држи локацију табеле мапе блокова. Но, за израчунавање физичке адресе, на овај начин, потребно је најмање два приступа меморији. Ово је значајно успоравање рада рачунара, па се примењују такозване асоцијативне меморије.

9.6.4.2. Израчунавање адресе применом асоцијативних меморија

Неки рачунари табелу мапе блокова смештају у специјалну меморију која се зове меморија са адресирањем по садржају. Овој меморији се приступа на бази целог, или делимичног садржаја који је уписан у неку њену локацију, а не на бази адресе локације. Овај део садржаја ћелије, на основу којег се она идентификује, назива се кључем. У меморији може постојати и више ћелија са истим кључем. Меморије код којих се идентификација ћелија врши кључем, а не адресом, називају се још и асоцијативним. Асоцијативне меморије захтевају специфичан управљачки систем, а обично и специфичне меморијске елементе. Асоцијативне меморије су нарочито корисне када је потребно убрзати проналажење жељеног податка у неком скупу података, на пример табели парова (пресликавања), када се један елемент пара сматра кључем.

Асоцијативне меморије су знатно скупље од обичних. Сложеност и цена не дозвољавају велике капацитете асоцијативних меморија. Ради јефтинијег решења, асоцијативни приступ се некада остварује и у обичној меморији на тај начин што се сукцесивно приступа њеним ћелијама у циљу проналажења податка који задовољава услов тражења. Овакво тражење је споро. При упису се тражи прва такозвана слободна ћелија која не садржи податак од неког интереса, па се у њу уписује жељени податак са својим кључем.

Асоцијативне меморије раде на следећи начин. Софтвер задаје жељени меморијски садржај (редни број блока **b** из виртуелне адресе), а хардвер тражи тај садржај у меморији. Претраживање асоцијативне меморије представља истовремено (симултано) испитивање садржаја свих локација, а не једне по једне (тј. секвенцијално). На овај начин се може уштедети и до 90% времена за одређивање адресе физичке локације. Када се пронађе

задати садржај добија се информација о адреси податка у физичкој меморији. Поступак је релативно сложен и излази из оквира почетних разматрања.

9.6.5. СТРАНИЧЕЊЕ И СЕГМЕНТАЦИЈА

Проблем појаве фрагмената и обраде великих програма решава се помоћу поделе на блокове. Постоје две врсте блокова које се стварно користе у рачунарским системима, а то су: странице и сегменти. Иако има доста разлика међу њима у основи им је иста идеја. Систем страничења дели расположиву меморију и процес на једнаке блокове, чија је величина 2^n, често 512 или 1024 бајта, и сл. Сегментација користи блокове неједнаке величине коју одређује програмер. Један сегмент одговара једном потпрограму, групи функционално врло повезаних потпрограма или низу података.

Основне разлике између сегментације и страничења јесу следеће:

- *сегменти су логичке целине, а странице физичке целине одређене архитектуром рачунарског система,*
- *сегменти су промењљиве дужине, а странице фиксне дужине,*
- *сегменти су видљиви за програмера, док су странице потпуно транспарентне,*
- *код сегмента је интерна фрагментација (фрагментација процеса) потпуно елиминисана а екстерна фрагментација (фрагментација меморије) може постојати. Код страница је обрнуто, јер је екстерна фрагментација потпуно елиминисана, а интерна фрагментација је занемарива.*

9.6.5.1. Додела меморије у страницама

При алокацији меморије у страницама управо се постиже да су делови процеса који се уносе у меморију по величини једнаки празним местима у меморији, па тако након смештаја не настају никакви нови фрагменти. Проблем фрагментације на тај начин је у потпуности решен.

Дељење меморије у блокове фиксне величине није прописано стандардима већ је зависно од концепције појединих произвођача рачунарског система. Али, оно што је значајно, јесте да је и сам процес, такође, подељен у делове исте такве величине. Те делове називамо још и страницама. Процеси су смештени, по правилу, на неку екстерну меморију где чекају да буду унети у главну меморију. Чим се ослободи онолико страница оперативне меморије колико дотични процес има, или му је неопходно за смештање перманентног (сталног) дела, биће омогућена алокација тог

процеса, без обзира на то што ослобођене странице централне меморије нису све у једном континуираном простору меморије, него су разбацане по читавој централној меморији.

Потребан је посебан дигитални механизам који ће аутоматски израчунавати адресе за сваки појединачни случај, тј. пресликавати релативне адресе (**V= (b,d)**), у стварне физичке адресе **(r = h + d).** Тај механизам пресликавања адресног простора у меморијски простор мора бити хардверски изведен због потребне брзине.

Ако посматрамо реалну ситуацију, у којој постоји више процеса у једном систему, јасно је да сваки од процеса мора имати своју табелу страница. Оперативни систем мора знати почетну адресу сваке табеле страница (**a**), јер се на бази те адресе и броја блока (**b**) проналази адреса **h** почетка блока података у стварној, физичкој меморији.

При сваком адресирању имамо два приступа у меморију. Један је приступ у табелу страница да би се утврдила стварна адреса у меморији, а други је приступ на израчунату адресу. Због тога је могуће, ради постизања веће брзине, за табелу страница текућег процеса користити посебне регистре за адресирање, које називамо регистри страница или регистри за пресликавање страница (**page map registers**). Табеле страница, упркос томе, морају и даље постојати али се третирају као околина процеса, што значи сваки пут када процес добије контролу извођења, садржај табеле страница се пребацује у регистре за странице (и не само ове регистре, него и све остале) који се даље користе за адресирање, све док траје извођење тог процеса, односно до следећег прекида.

Неки врло сложени оперативни системи реализују идеју о пуњењу (у меморију), само активних делова процеса, док неактивни делови остају на екстерној меморији, или скривеном диску. Међутим, током извођења процеса долази до промене ситуације и појединачне странице постају непотребне, док друге садрже податке или делове програма које треба обрађивати. Због тога треба те странице, које су постале неактивне, избацити из меморије, а на њихово место прекопирати странице које постају активне. Због тога се овакав поступак зове **замена страница на захтев**. Процес се изводи све до часа кад се жели приступити податку чија се адреса налази на неактивној страници. Неактивна страница је на екстерној меморији, па због тога процес тражи услугу од оперативног система, тако што генерише захтев за пуњење управо те странице на којој се налази тражена адреса. То значи да долази до посебне врсте прекида који се назива прекид због странице (**page interrupt, page fault**). Обрада тог прекида састоји се управо у пуњењу тражене странице. Овде може адресни простор бити већи од меморијског простора. Таква, привидно повећана, меморија назива се виртуелном меморијом.

9.6.5.2. Додела меморије у сегментима

При дељењу процеса на блокове једнаке дужине не поштује се логичка структура програма, па се може десити да је један део структуре у главној меморији а други на диску. У том случају би могло доћи до сувише честих измена страница, и великог успоравања процеса. Сегментација полази управо од логике коју процес у себи садржи, и логичке целине процеса узимају се као посебни делови које треба смештати и позивати у меморију. Да би се извршило пресликавање програмских (виртуелних) адреса у стварне адресе, употребљава се, опет, техника записивања почетних адреса појединих сегмената у табели. Ту табелу називамо табелом сегмената.

Величина сегменат је различита, па је табела мапе блокова, која се сада зове табела сегмената, модификована и садржи и величину сегмента, као на слици 9.18.

присуство сегмента у меморији	*дозвола приступа*	*величина сегмента*	*ред. бр. употр. сег.*	*измена садржаја*	*адреса*
1	w		3	1	
1	r		4	0	
0			–	–	
1	n		2	0	
1	e		1	1	

Слика 9.18. Табела сегмената са битовима за коришћење и измену садржаја

При избацивању блока (странице или сегмента) мора се водити рачуна да ли је било измена садржаја у блоку. Ако није (0), блок се једноставно пребрише. Ако јесте (1), онда се прво стари блок смешта на диск, а тек онда се на његово место уписује нови блок.

Сегмент је логична целина и може се, као такав, посебно употребљавати, па нам омогућава провођење врло ефикасних, једноставних метода заштите. Ако неки сегмент може послужити само као рутина која се може изводити, онда таквом сегменту можемо дати ознаку **E** (**execute**) која значи да се тај сегмент може само изводити, а не може се у њега уписивати неки нови садржај, нити се може прочитати шта сегмент садржи. Знак **R** (**read**) ограничава приступ само на читање, знак **W** (**write**) допушта упис у сегмент, док ознака **N** спречава било какав приступ корисничком процесу у тај део меморије. На тај начин можемо допустити неком процесу да користи сегменте из неког другог процеса, јер их не може уништити, ни променити, па чак ни прочитати. Ова техника заштите зове се техника ограниченог приступа, и о њој ће бити више речи код метода заштите датотека.

9.6.6. ЈЕДИНИЦЕ ЗА УПРАВЉАЊЕ МЕМОРИЈОМ

Имплементација система за управљање меморијом захтева и значајну хардверску подршку. То могу бити асоцијативне меморије, уређаји за транслацију адреса, ултра брзе меморије за памћење табела мапе блокава или мапе страница (сегмената). Код микрорачунарских система, хардвер за упављање меморијом обично је реализован у облику једног чипа (**Memory Management Unit, MMU**). Овај чип прихвата логичку адресу добијену од микропроцесора и генерише одговарајућу физичку адресу користећи своје интерне табеле. Посебне табеле могу бити придодате за корисничке просторе и системски простор. Ове табеле, такође, могу садржати и информације за заштиту меморије и ограничавање приступа. Табеле пуни сам микропроцесор, а такође и обавештава јединицу за управљање меморијом када треба да пређе из једног корисничког простора у други кориснички простор или у системски простор. Већ 16–то битни и 32–битни процесори имају јединице за управљање меморијом.

9.7. УПРАВЉАЊЕ УЛАЗОМ И ИЗЛАЗОМ

Управљање улазом–излазом је функција оперативног система чији циљ је да управља, контролише и координира рад улазно–излазних јединица и операција, као и да омогући комуницирање процеса са спољашњом околином рачунарског система. Улазно–излазне јединице чине три групе периферијских уређаја, а то су:

- *прави улазни уређаји, који могу послужити само за улазне операције* (тастатуре, оптичке оловке, мишеви, итд.),
- *прави излазни уређаји, који могу послужити само за излазне операције* (плотер, штампач, итд.),
- *уређаји улазно–излазног типа, који могу бити или улазни, или излазни, или и улазни и излазни.*

Очито је да се ради о мноштву различитих уређаја, од којих сваки има своје хардверске карактеристике, и своје карактеристичне операције. Између ових уређаја постоје:

- *разлике у операцијама које поједине јединице могу изводити. Те разлике постоје не само међу уређајима различитих типова, већ и међу уређајима истог типа (траке се премотавају, а дискови не),*
- *разлика у брзинама којима поједине јединице изводе операције,*
- *разлике у кодовима који се употребљавају за представљање података на појединим уређајима,*
- *разлике у преносу података, величини блокова, брзини којом се подаци преносе.*

Да би се улазом и излазом могло управљати на јединствен начин, потребно је премостити велике разлике које међу јединицама постоје, и на неки начин софтвер учинити независним од конкретног уређаја. Независност од појединачне јединице истог типа и независност од типа јединица може се добрим делом спровести тако да процес не оперише са стварним јединицама него са такозваним виртуелним јединицама. Када је процесу потребно да изврши операцију улаза или излаза он позива оперативни систем, који даље преузима контролу над преносом. Но, да би оперативни систем могао да обави захтевану операцију, кориснички програм мора да задаје одређене параметре, на основу којих је могуће ближе описати операцију коју треба извести, као и уређај који ће се при томе користити. Треба имати у виду да у једном рачунарском систему може постојати више уређаја истог типа, а, такође, и више процеса који равноправно могу конкурисати за било који од уређаја. Такође, треба знати да се може користити само слободан уређај, или треба сачекати да се потребни уређај ослободи. Због тога се у програму не наводи конкретан уређај, већ врста уређаја, као и тип операције која треба да се изврши (улаз, излаз, улаз–излаз). Програмер мора само да одреди где се тачно налазе подаци који треба да буду пренети, где треба да буду пренети и колико тих података има, односно количину података. Уколико је у могућности, програмер треба да одреди и остале параметре као, на пример, брзину преноса, али неће бити трагично ако то не уради.

Оперативни систем на бази захтева и параметара добијених од корисничког програма, повезује јединицу коју је програмер замислио да ће користити (виртуелну јединицу) са неком конкретном јединицом из састава рачунарског система. Повезивање врсте јединице са типом јединице изводи се путем стандардних спецификација или стандардних вредности (**default**). То значи да ако се неки параметар не зада (ако рецимо програм није конкретно именовао неку стварну јединицу) онда се аутоматски узима нека унапред задата и систему позната вредност за тај параметар (у овим примеру то би била јединица која може обавити задату операцију, и при томе је слободна). Стандардна спецификација (**default**) је, дакле, одређење које се узима у недостатку праве експлицитне вредности (то је у ствари имплицитна вредност).

Сваки процес има свој контролни блок (**PCB, Process Control Block**) у који се поред осталог уписују и информације о стању улаза–излаза, као и информације о уређајима који су процесору додељени. Сваки уређај са своје стране, такође, има свој контролни бкок (**UCB - Unit Control Block, Device Descriptor - DD**) који садржи карактеристике јединице:

– *идентификатор јединице* (физичко име),
– *ознаку стања јединице* (слободна, заузета, у квару итд.)
– *показивач реда захтева за улаз–излаз на тој јединици*,
– *показивач контролног блока процеса који користи јединицу,*
– *показивач захтева који се опслужује,*

– *показивач табеле за конверзију кода,*
– *ред за регистровање захтева назван* ***захтев-постоји***
– *ред за регистровање завршетка* **U/I** *операција назван* **I/O–завршен.**

Повезивање виртуелне јединице (тј. њеног контролног блока) са оперативним системом (тј. са системским контролним блоком) врши се онда када се први пут затражи извођење поједине U/I операције за ту јединицу, а тај поступак се назива отварање виртуелне јединице, и обрнуто, када се процес заврши, јединице се затварају. То је имплицитно отварање и затварање виртуелних јединица.

Приликом повезивања процеса са улазно–излазном јединицом упоређују се параметри које је задао програмер, са карактеристикама јединице, и ако су унутар скупа могућих вредности оне се прихватају. Ако неки параметар није специфициран, узима се претпостављена вредност из контролног блока јединице (**default specification**). Ако задати параметар излази из оквира који су прихватљиви за ту јединицу, оперативни систем сигнализира грешку, и предлаже решење (тј. даје скуп могућих вредности).

9.8. УПРАВЉАЊЕ ДАТОТЕКАМА

Магнетни оптички и полупроводнички уређаји који се користе као масовне меморије садрже милијарде бајтова информација. Да би помогао, убрзао и олакшао приступ овим информацијама, оперативни систем дели запамћене информације у датотеке. Датотека није ништа друго до *именовани*, *логички скуп података који је на познат начин смештен на физичку меморијску јединицу*. Физичка организација података може бити или континуирани низ бајтова, или нека од сложенијих структура података. Да би се оперативни систем ефикасно користио, он мора знати структуру датотека.

Неки оперативни системи подржавају само секвенцијалне датотеке, неки други само оне са случајним приступом, а трећи индексиране датотеке. Оперативни системи који подржавају широк спектар типова датотека су врло сложени, а такође се усложњавају и захтеви који се односе на секундарне меморије. Управљање датотекама је део једног општијег питања, а то је руковање и управљање информационим целинама (**information management**). У рачунарском систему то је посебно сложено јер се изворне информационе целине на различите начине трансформишу, преносе, меморишу, рашчлањују, састављају, анализирају и користе за генерисање нових информација. Управљање информационим целинама можемо посматрати као три посебна система, који не искључују један другог, али се разликују по својој софтверској реализацији, по различитим начинима третирања информационих целина, и по различитим информационим целинама којима оперишу. Ти системи су:

- **системи за управљање подацима** (**data management systems**) *који представљају део оперативног система, а који брину о подацима у главној меморији. Управљање подацима садржи скуп правила за организовање и приступање структурама података не улазећи у њихов информациони садржај,*
- **системи за управљање датотекама** (**file systems, filing systems***), који представљају део оперативног система, а њихов је задатак да брину о неструктурираним информационим целинама (не улазећи у њихов садржај) које се налазе на спољашњим меморијама.*
- **системи за управљање базама података** (**Data Base Management Systems, DBMS**). *То је засебан софтвер из групе услужних програма који оперише са структурираним, али и интерпретираним информационом целинама, тј. води рачуна о информационом садржају. У пракси, системи за управљање подацима и датотекама међусобно се допуњују у својим функцијама и раду, па се код многих реалних система и поистовећују.*

Да би се подаци на спољним меморијама могли ефикасно и безбедно користити, систем за управљање датотекама мора водити евиденцију о свим датотекама које постоје, где их треба меморисати, како их треба меморисати и ко (који процес) може те датотеке користити и на који начин.

Са становишта корисника, датотека је организован скуп слогова података који су сврстани по одређеном редоследу. Те слогове називамо логичким слоговима. Насупрот томе, спољне меморије су подељене на блокове који чине јединице за меморисање података које називамо физичким слоговима. Један од основних задатака система за управљање датотекама јесте да смести логичке слогове података у физичке блокове спољне меморије (при томе најчешће мислимо на диск).

И овде ће нам, као и увек до сада, за решење проблема послужити посебне табеле које називамо *директоријуми*, *каталози* или *имениц*и датотека (**File Directory, File Dictionary, Volume Table of Contents** (**IBM**), **File Catalogue, File Folder**).

Директоријум, односно именик датотека, *је табела која служи за пресликавање симболичких имена датотеке у физичку адресу*, али, такође садржи низ информација о томе где је почетак датотеке, колико је велика, на којим блоковима је записана, итд. Директоријум се смешта на почетак спољне меморије којој припада, тако да и сам директоријум постаје датотека.

9.8.1. ДИРЕКТОРИЈУМИ

Датотеке, обично, не егзистирају на диску или траци појединачно, усамљене. Ако су изоловане, онда сам програмер (корисник) мора одредити и запамтити где је свака од датотека смештена. У овом случају је практично

немогућ вишекорисnички систем, јер како ће више програмера моћи да зна где се налази датотека коју је неко други направио и сместио. Оперативни систем обезбеђује средства за организацију датотека, а то је систем **каталога**, **именика** (**directory, file folder**). Систем директоријума организује датотеке на секундарној меморији. У својој најједноставнијој форми директоријуми се могу реализовати као табеле симбола које садрже списак имена датотека и информације како да се свака од њих пронађе. Већина директоријума садржи:

– *информације за идентификацију* (име датотеке, лозинке, итд.),

– *информације о физичким карактеристикама* (број додељених блокова, почетну адресу, начин повезивања блокова итд.),

– *информације о логичким карактеристикама датотеке* (врсту слогова, дужину слога, формат слога, број слога, фактор блокирања, организацију датотеке, број кључева, позиције кључева итд.),

– *информације о приступу и заштити датотеке* (допуштени приступ, допуштено дељење, корисници итд.),

– *информације административног типа* (датум креирања, време чувања, датум и корисник последњег приступа, датум последњег ажурирања, подаци о грешкама итд.).

Ови подаци варирају с обзиром на разне врсте система у хардверском и софтверском смислу, на произвођаче, кориснике и филозофије изградње читавог рачунарског система.

9.8.1.1. Команде за рад са директоријумима

Оперативни системи имају велики број команди за рад са директоријумима. Неке команде омогућавају креирање и уништавање директоријума (**mkdir, rmdir**, у **DOS**–у). Друге команде омогућавају кориснику претраживање директоријума и тражење потребне датотеке. **MS-DOS** обезбеђује ову функцију командом "**dir**", а **UNIX** користи "**ls**". Ове команде допуштају кориснику да листа све датотеке у директоријуму, и да нађе одређену датотеку, или групу датотека које имају неке заједничке карактеристике. У оперативном систему **MS-DOS** постоје следеће варијанте наредби за листање директоријума:

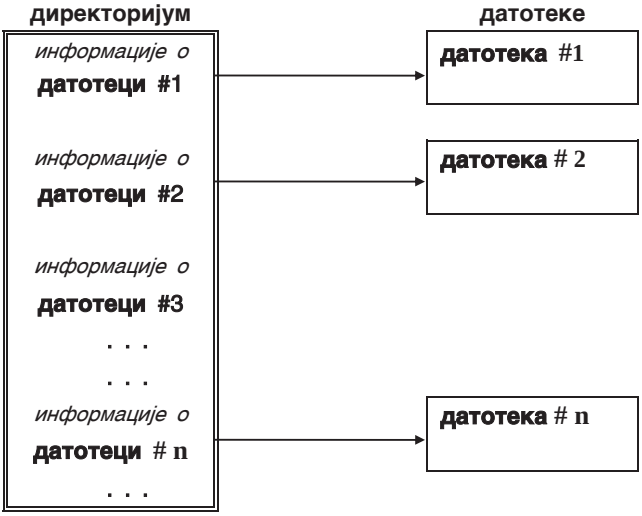
dir	*листа садржај текућег директоријума*,
dir a:	*листа садржај диска са именом* **a**,
dir pet*	*листа датотеке чије име почиње словима* **pet**,
dir *.exe	*листа датотеке са екстензијом* **exe**, *итд.*

9.8.1.2. Организација директоријума

Датотеке се на масовне меморије готово никада не смештају изоловано, свака за себе, већ се путем директоријума, на неки начин, повезују. Системи директоријума могу бити пројектовани тако да организују датотеке на више разних начина. Најпростија форма се састоји од једног директоријума.

9.8.1.2.1. ДИРЕКТОРИЈУМИ СА ЈЕДНИМ НИВООМ

Могуће је у једној табели држати информације о свим датотекама на спољашњој меморији, слика 9.19. Овакав приступ је коришћен у оперативним системима у великом броју микрорачунара као што су **CP/M, Apple DOS 3.3** и прве верзије **MS-DOS**–а. Али, коришћен је и у неким магистралним системима као што је **UNISYS OS 1100**.



Слика 9.19. Системи са једним нивоом директоријума

Сваки систем директоријума мора бити способан да на јединствен начин идентификује сваку датотеку. Због ограничених могућности именовања у системима са једним нивоом директоријума оперативни систем **CP/M** покушава да превазиђе овај проблем помоћу функције корисника (**user**), која омогућава да се све датотеке сместе у једну од 16 група.

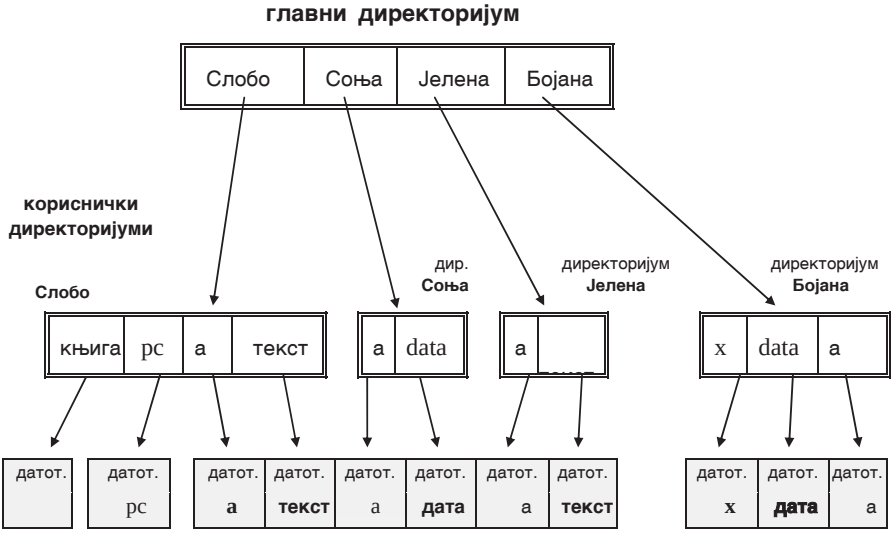
Кориснички број је део имена датотеке која се обично не приказује. Због корисничког броја једно исто име може се користити до 16 пута. То значи

да може постојати датотека **dat1.TXT** за корисника 1, и **dat1.TXT** за корисника 6. **Unisys**–ов **OS 1100** развија ову идеју даље обезбеђујући једно проширено име које укључује број рачуна власника, пројекат и име датотеке. Овом шемом се добија практично неограничен број датотека са истим именом, јер свака има јединствену комбинацију броја рачуна и пројекта.

Претраживање директоријума са једним нивоом може захтевати много времена, нарочито ако треба секвенцијално испитати хиљаде записа (уписа). Ово се може превазићи сортирањем директоријума или применом поступка остатка дељења, но оба ова поступка имају и своје мане.

9.8.1.2.2. ДИРЕКТОРИЈУМ СА ДВА НИВОА

Због поменутих проблема, неки оперативни системи имају имплементирану структуру са два нивоа, слика 9.20.



Слика 9.20. Систем директоријума са два нивоа

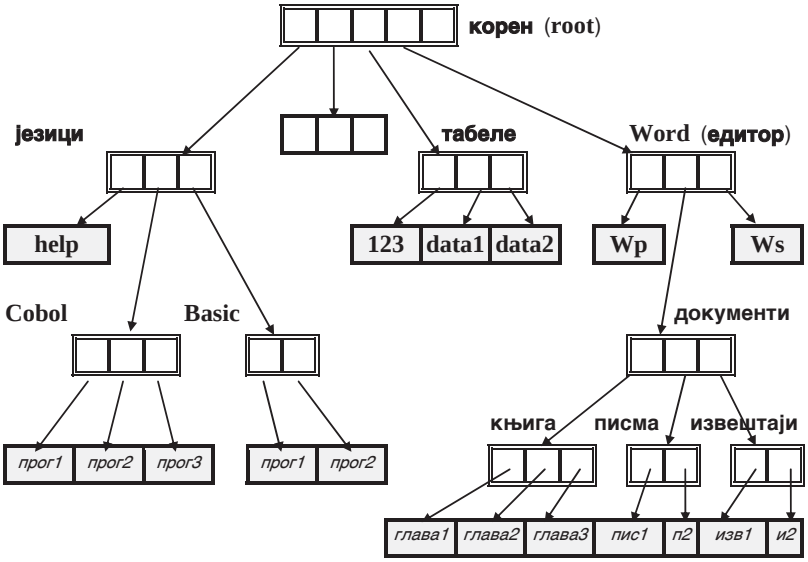
Овај систем садржи главни директоријум датотека (**MFD, Master File Directory**) и корисничке директоријуме. **MFD** садржи информације о корисничким директоријумима. Када корисник ***Бојана*** захтева приступ датотеци **дата**, оперативни систем потражи у **MFD** локацију *Бојаниног* директоријума, оперативни систем тада приступа том директоријуму и одређује локацију датотеке **дата** и на крају јој приступа. Видимо да

директоријуми у два нивоа допуштају дуплирање имена датотека; сваки кориснички директоријум има датотеку **а**, постоје две датотеке **дата** и **текст**. Ово је могуће јер је појављивање **а**, **дата** и **текст** датотеке у ствари јединствено, али у корисничком директоријуму.

Неки системи директоријума у два нивоа чине могућим дељење датотека између два корисника (тј. заједничко коришћење датотека). Ови системи придруже потребне директоријуме корисничком процесу, а спрече приступ другим директоријумима.

9.8.1.2.3. ДИРЕКТОРИЈУМИ СА СТРУКТУРОМ СТАБЛА

MS-DOS 2.0 и надаље, **UNIX** и неколико других оперативних система, користе директоријуме са структуром стабла, слика 9.21. Почетни, главни директоријум се зове корен (**root**), а његови записи указују, или на датотеке, или на помоћне директоријуме у следећем нивоу, тј. на поддиректоријуме (**subdirectories**), а ови пак указују на датотеке или на неке друге поддиректоријуме. Директоријум у којем се тренутно налазимо назива се текући.



Слика 9.21. Директоријум са структуром стабла

Тачна локација датотеке је одређена низом директоријума који мора бити пређен да би се дошло до ње. До датотеке **data 1** се долази преко **корена** у **табеле** и тек се онда дође до **data 1**. До једне датотеке **прог1** долазимо

из директоријума **корен**, преко директоријума **језици** у директоријум **Cobol**, а онда приступамо датотеци. Друга датотека **прог1** се проналази преко **корена** и **језика** до **BASIC** директоријума, а онда приступамо датотеци.

Низ директоријума који су пређени да би се приступило датотеци зове се **ПУТ** (**path**). У **UNIX** оперативном систему се пут до прве датотеке **прог1** обележава:

/ језици / BASIC / прог1.

Прва коса црта (**/**) указује на корен, а остале раздвајају имена поддиректоријума и име датотеке.

У **MS-DOS**–у би то било:

C:\ језици\ BASIC\ прог1.

Слова **C:, D :, E**: и **F**: представљају ознаке за тврде дискове. Оба ова пута казују да се датотека **прог1** налази на поддиректоријуму **BASIC**, који је на поддиректоријуму **језици**, на директоријуму **корен** на диску **C**. Ова структура допушта корисницима да деле датотеке под надзором лозинки и приступних права. При томе се мора увек знати пут од ***корена*** или од текућег директоријума до потребне датотеке.

Оперативни системи **MS-DOS** и **UNIX** користе тачку (**.**) за означавање текућег директоријума, а две тачке (**..**) за означавање претходног директоријума у односу на текући. Претходни директоријум од **COBOL**–а је **језици**, а од језика **корен**. Ако је текући директоријум **књиге**, да би приступили датотеци **изв1** у **UNIX**–у треба написати ***пут*****:** или као **/word/документи/извештаји/изв.1**, или **../извештаји/изв**1. У првом случају крећемо из **корен**а, а у другом из претходника, а претходни директоријум од **књиге** је **документи**.

Сваки корисник има свој текући директоријум који служи као препоручени (**default**) директоријум за сва обраћања тог корисника датотекама. Већина оперативних система који користе структуру стабла одређују сваком кориснику текући директоријум у току његовог повезивања на систем (**logon**). Претпоставимо да је директоријум **језици** текући. Онда се датотеци **"help"** може приступити једноставним позивањем њеног имена, тј. не треба нам да листамо пут од корена до датотеке.

9.8.2. ОПЕРАЦИЈЕ СА ДАТОТЕКАМА

Оперативни систем подржава велики број операција са датотекама, као на пример: креирање нових и брисање постојећих датотека, приступ и заштиту датотека од недозвољене врсте приступа, обраћање датотекама симболичким именима, обезбеђивање интегритета датотека, дељење тј. коришћења датотеке између више корисника (**sharing**),

9.8.2.1. Прављење датотеке

Креирање датотека је процес током којег оперативни систем прави упис потребних података у директоријум да би указао на постојање датотеке. Неки системи у том тренутку указују и на физички простор за смештање, док други креирају само запис у директоријуму. У сваком случају при креирању датотеке мора јој се дефинисати име које се уноси у директоријум. Сваки оперативни систем има своја правила за дефинисање имена датотека. У оперативном систему **MS-DOS,** име се састоји од два дела који су међу собом раздвојени тачком (.). Први део се зове име (име у ужем смислу речи), и сме бити дужине до осам карактера. Други део се зове проширење, екстензија, дужине до три карактера, и одређује тип датотеке које на известан начин одређује дозвољене врсте приступа. Неки типови датотека се сматрају стандардним па им је проширење имена фиксно да би били препознатљиви за оперативни систем. Тако се за извршне програме у машинском језику користе екстензије **exe** и **com**, за програме у **BASIC**–у екстензија **BAS**, а за **C**–програме **C**, као на пример:

program.asm	*програм у асемблеру*,
program1.c	*програм у* **C**,
pismo.txt	*текст*,
knjiga.tex	*текст*,
knjiga.bak	*резервна копија текста, итд.*

Неки оперативни системи имају специјалне команде за креирање. **Unisys 1100** има команду **@CRE** и **@ASG,U,** док **DOS** и **UNIX** то раде имплицитно, помоћу едитора или копирањем.

DOS команда: **copy file1 file2** копира садржај датотеке **file1** у **file2**. Датотека **file2** ће бити креирана ако није постојала. **UNIX** команда **cat file1>file2** копира садржај **file1** у **file2**, при чему се датотека **file2** креира у том часу.

9.8.2.2. Уништавање датотеке

Сви оперативни системи имају путеве за уклањање датотека са меморије и ослобађање простора који су оне заузимале. Команде које то раде се зову **delete, del, kill, remove, rm, erase, unsave** и слично. Стварни учинак наредбе за уништавање зависи од система. У већини случајева, простор који је датотека заузимала, враћа се оперативном систему, али се стварно неће ништа брисати или мењати док се тај простор не додели другој датотеци. Ово омогућава опоравак датотеке која је случајно уништена. Други системи само обележе датотеку као уништену, али ништа не бришу док се не пошаљу још неке команде, па је и овде је могућ опоравак датотеке.

9.8.2.2. Отварање и затварање датотека

Скуп података који описују начин смештања датотеке на спољној меморији, назива се контролни блок датотаке (**FCB, file control block**), а основни подаци који се налазе у контролном блоку датотеке су:

– *име датотеке,*

– *адреса контролног блока јединице на којој је датотека,*

– *адреса првог блока датотеке,*

– *адреса следећег блока,*

– *врста приступа у датотеку.*

Отварање датотеке **(OPEN**) је поступак који отвара пут ка самој датотеци. Тај поступак је врло сложен и повезује процес, оперативни систем, саму јединицу спољне меморије, и директоријум. Повезивање се састоји у томе да се током извођења **OPEN** поступка креира контролни блок датотеке **(FCB**) у главној меморији да би био лако доступан оперативном систему.

Процедура **OPEN** обавља следеће операције: претражи и пронађе у директоријуму податке који се односе на ту датотеку, провери да ли је приступ који се тражи допуштен, провери да ли је датотеку већ отворио неки други процес, па ако је отворена провери слажу ли се врсте приступа (ако је отворена за уписивање не може бити отворена и за читање и обрнуто). На крају се креира њен контролни блок (који трасира "пут" за коришћење датотеке). Тако се сваки следећи приступ у датотеку спроводи употребом контролног блока, што значи да се не мора увек поново претраживати главни и помоћни директоријум. Да би се датотека отворила, морају се задати два основна параметра, а то су име датотеке и врста приступа датотеци:

OPEN (**име**, **приступ**).

Инверзна процедура је затварање датотеке, и она брише овај "пут" када коришћење датотеке више није потребно, а позива се на следећи начин:

CLOSE (**име**).

Затварање датотеке информише систем да та датотека више није потребна. Систем ажурира директоријум, односно оне податке који се тичу коришћења датотеке. Грешка у затварању датотеке често резултира у губљењу свих података уписаних у датотеку.

9.8.2.4. Приступање датотеци

Пре него се употреби, датотека мора бити отворена. Извршавањем команде за отварање датотеке, оперативни систем лоцира жељену датотеку на

масовној меморији и чита њен опис из директоријума и преноси га у оперативну меморију.

Након отварања, датотеци се може приступити са циљем читања, уписа, поновног уписа, тј. ажурирања (**read, write, rewrite, update** команде). Упис подразумева додавање нових записа, док ажурирање изазива модификацију постојећих записа. Читање помера податке из датотеке у програм. Неки системи имају и операцију додавања (**append**), која додаје записе на крај постојеће датотеке. Други системи имај операцију убацивања (**insert**) којом се нови записи убацују између постојећих записа (само код индексираних датотека, док је за секвенцијалне немогуће).

Скоро сви оперативни системи обезбеђују пут за копирање датотека. Код неких оперативних система као што су **DOS, UNIX** постоје команде, а код неких постоје програми. Неки системи имају могућност да помере датотеку са једног места на друго (**move**). Већина система даје могућност кориснику да промени име датотеци **(rename**).

Многи оперативни системи имају један показивач који указује на бајт или запис који ће бити следећи прочитан. У систему постоји, често, и команда репозиционирања којом се мења локација на коју показује овај показивач. Померање на почетак датотеке на диску или на траци зове се **rewind**– премотавање, слика 9.22., јер физички то захтева постављање траке на почетак. Могуће је померање за један запис уназад (**backspace**).

Слика 9.22. Основне обраде записа датотеке на траци

9.8.3. ПРАВА ПРИСТУПА ДАТОТЕКАМА И ДИРЕКТОРИЈУМИМА

Многи системи са директоријумима садрже механизме заштите и информације о приступним правима. У заштити датотека најчешће фигуришу лозинке (**passwords**) које морамо предати оперативном систему пре него нам он потврди приступ. Неки системи захтевају једну лозинку за читање и разне друге за модификацију датотека. Приступна права указују коме је допуштено да зна за постојање датотека и шта може да ради са њима. **UNIX**, на пример, дели потенцијалне кориснике датотека у три категорије: *власник*, *члан групе власника* (сувласник) и *остали*. Свакој класи корисника могу бити придружена права приступа са циљем читања, уписивања и извођења. Права приступа садржана су у директоријуму и могу спречити неовлашћене кориснике да уопште знају за постојање датотека. Други

системи користе хијерархијску организацију приступних права, попут оне са слике 9.23.

Слика 9.23. Хијерархија приступних привилегија корисника датотека

Корисници на најнижем нивоу не знају чак ни да датотеке постоје, што је врло важно, јер у неким осетљивим апликацијама и само знање имена датотеке може бити опасно по безбедност.

Ниво обавештености допушта представљање (приказивање) списка датотека кориснику и ништа више. На нивоу извођења, корисници могу да захтевају извођење преведеног програма, али немају права да листају, копирају, модификују или бришу садржај датотеке.

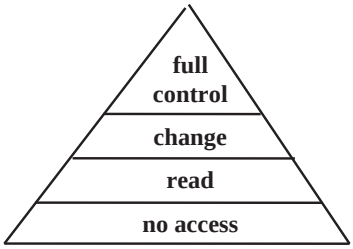
Ниво читања даје кориснику право приступа појединим записима у датотеци. Кориснику на овом нивоу, у општем случају, допушта се и копирање записа у другу датотеку.

Нови податак може бити додат у постојећу датотеку на нивоу додавања, али постојећи податак не може бити ажуриран нити избрисан. Корисник који има право да модификује постојеће податке налази се на нивоу ажурирања.

На највишем нивоу налази се корисник који може да уништи датотеку.

Приступна права на слици 9.23. су хијерархијска. Наиме, корисник на било ком нивоу, нормално поседује права за тај ниво, али и сва друга права која су прописана за све ниже нивое. Корисник којем је дозвољено да чита датотеку (програм), нормално је може позвати на извршавање, али не може додавати записе у њу.

На сличан начин питање права приступа дељеним директоријумима (**share folder**) решава и оперативни систем **Windows NT**. Наиме, за кориснике који су повезани у рачунарску мрежу постоји хијерархија дељених овлашћења од најрестриктивнијих до највећих, приказана на слици 9.24.



Слика 9.24. **Share** овлашћења у **Windows NT**

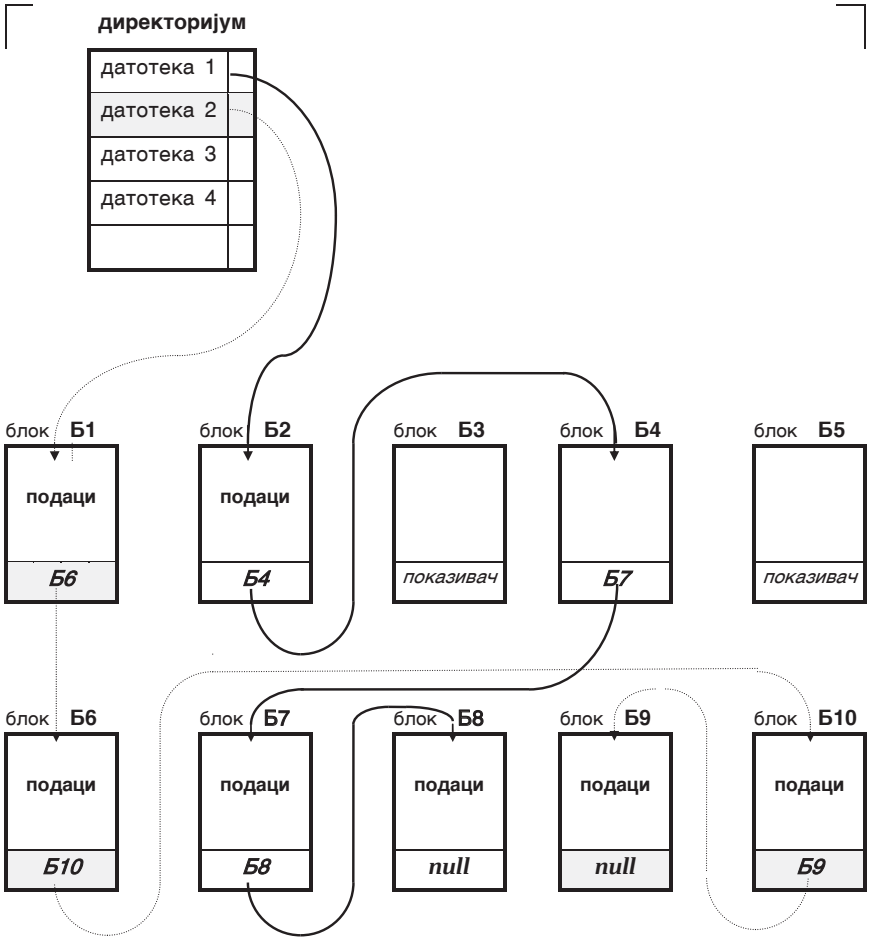
9.8.4. СМЕШТАЊЕ ДАТОТЕКА И ДИРЕКТОРИЈУМА

9.8.4.1. Континуално смештање

Овакав поступак се може применити за држање података на диску, али он захтева у тренутку креирања датотеке, доделу максимално могућег простора, који ће датотека икада користити. Не само да треба да постоји довољно слободног простора за датотеку, него тај простор мора бити непрекидан (континуиран), као један блок. И што је такође лоше, једном додељен простор може користити само та датотека, па ће велики део простора остати неискоришћен за дуже време, тј. до формирања коначне форме датотеке.

9.8.4.2. Повезано смештање

У овом приступу, директоријум садржи показивач на први блок датотеке. Тај блок заузврат показује на следећи блок, и тако редом, до блока који садржи специјални крај–листе индикатор (**end-of-the-list**), слика 9.25. Датотеци **дат1** додељени су блокови **Б2, Б4, Б7, Б8,** а датотеци **дат2** блокови **Б1, Б6, Б9, Б10.** Прављење датотека из елемената типа повезане листе не захтева доделу непрекидног простора на диску, већ блокови могу бити било где. То даље значи да се датотеци придружује само онај простор који она у том тренутку и заузима, а када датотека расте, додељују јој се нови блокови. Брисањем датотеке враћају се оперативном систему њени блокови, и он их може доделити другим датотекама. То значи да оперативни систем мора водити евиденцију о слободном простору у меморији (**free list**). То се, такође, може радити повезаним листама или, чешће, мапирањем меморије помоћу *мапе битова* (**bit map**). Овде сваком блоку одговара један бит; ако је јединица блок је искоришћен, а ако је нула блок је слободан. Када датотеци треба још простора, оперативни систем нађе слободан блок у *бит–мапи*, упише уместо нуле јединицу, и повеже тај блок као последњи у листу датотеке. Брисањем датотеке само се на нулу постављају они битови у бит–мапи који одговарају блоковима те датотеке.

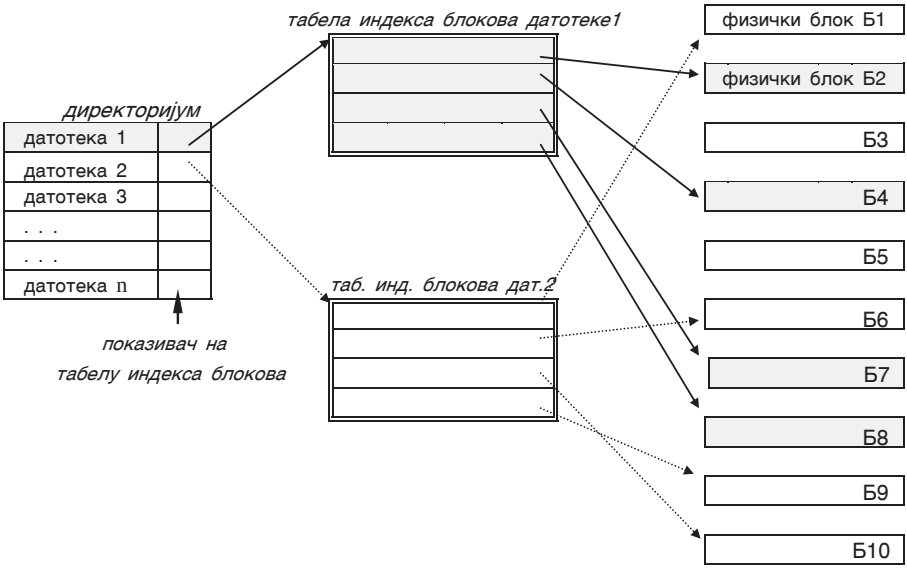


Слика 9.25. Структура повезане листе

9.8.4.3. Индексирање

У овом поступку директоријум садржи име датотеке и показивач на **табелу индекса блокова**. Индекс није ништа друго до скуп показивача поређаних оним редоследом који одговара редоследу додељивања блокова на диску.

Први податак у индексу блокова је показивач на први блок датотеке, други је показивач на други блок, и тако редом. Слика 9.26. приказује табеле индекса блокова за пример са слике 9.25.



Слика 9.26. Индекс блокова структуре типа датотека

9.8.4.4. Мапирање блокова

Могуће је доделити простор датотеци уз помоћ табеле која за сваки блок има једно поље. Број првог блока датотеке записује се у директоријум. За пример са слике 9.25 први блок датотеке **дат2** је физички блок број **1** (**Б2**). У мапи блокова, тј. табели смештања датотека, у поље **1**, уписан је број следећег блока **дат2**, а то је блок број **6**, слика 9.27.

директоријум

датотека 0	15
датотека 1	2
датотека 2	1
датотека 3	11
. . .	

табела смештања датотека

1	6
2	4
3	free
4	7
5	free
6	10
7	8
8	null
9	null
10	9
11	25

блокови на диску

1	**први блок дат.2**
2	**1. блок дат.1**
3	
4	**2. блок дат.1**
5	
6	**2. блок дат.2**
7	**3. блок дат.1**
8	**последњи блок дат.1**
9	**последњи блок дат.2**
10	**3. блок дат2**
11	први блок дат.3
12	

Слика 9.27. Табела смештања датотека

У пољу **6**, у табели смештања датотека, уписан је показивач на физички блок у меморији где је смештен следећи блок датотеке **дат2**, а то је **10**. То значи да је трећи блок датотеке уписан у блок број **10** на диску. Број у пољу **10** табеле смештања указује на следећи блок меморије који је додељен датотеци **дат2**, и то је блок **9**. У поље **9** уписан је показивач **null** (**end-of-the-file**), и он показује да је то последњи блок датотеке **дат2**.

MS-DOS користи табелу смештања датотека (**FAT, File Allocation Table**). Ова табела има 1 **byte** за сваки блок на диску. Код великих дискова у **FAT**–у се референцирају кластери уместо сектора. Кластер има 2,4 или 8 сектора, зависно од капацитета диска (1 кластер је 2 сектора за дискету од 360 **KB**, а 4 сектора за дискету од 1.2 **MB**), а 8 сектора за већину **hard** дискова).

9.9. ЗАКЉУЧАК

Са појавом рачунара отпочео је и развој оперативног система, најпре као помоћ у извођењу улазно–излазних операција, а потом преузимањем многих контролних функција од оператера и програмера. То је имало за последицу да се поред корисничког процеса у меморији рачунара нужно нађу још и процеси који припадају оперативном систему.

Снага рачунара у многоме зависи од оперативног система. Оперативни систем се састоји од низа програма који међу корисницима и програмима расподељују ресурсе рачунарског система. Део оперативног система остаје стално у меморији, и он представља језгро оперативног система које је одговорно за управљање примарном и секундарном меморијом, за креирање процеса, додељивање процесора и обраду прекида.

Процес је програм у поступку извођења. Ако је један програм два или више пута активиран (постоји више копија програма у исто време), он увек представља нови процес (вишепозивни програми).

Након што је процес креиран, он се може наћи у неколико разних стања. У стању готовости, спремности, процес има све што му треба за рад, изузев **CPU**-а. Када процесор добије **CPU**, он је у стању извођења. Процес који чека на улазно–излазне или друге услуге налази се у блокираном стању или стању чекања. Процес који је привремено избачен из конкуренције за **CPU**, налази се у стању суспензије. Оперативни систем користи низ табела за чување информација о активним процесима, улазно–излазним операцијама и датотекама. Ове табеле су често међу собом спојене у повезане листе које чине магацине и редове.

Како многи рачунари имају само једну **CPU**, само један процес може бити активан у једном тренутку времена. Прелазак са једног процеса на други захтева чување садржаја свих регистара старог процеса и њихово пуњење новим информацијама везаним за нови процес. То исто је потребно и при обради прекида. Прекиди могу настати на много разних начина, па се и рутине за обраду разликују, а зову се анализатори прекида.

Мултипрограмски оперативни системи имају неколико процеса у стању спремности који чекају да добију процесор. О томе брину распоређивачи на високом, средњем и ниском нивоу.

Развој рачунарског хардвера и оперативног система довео је до реализације система способних да једновремено подржавају много процеса, а да не угрозе њихову целовитост. То значи да се у једној јединственој оперативној меморији налази много процеса па је неопходно заштитити сваки процес од случајног или недозвољеног утицаја других. То се постиже одговарајућим хардвером (гранични регистри) и софтвером (делом оперативног система који се зове управљање меморијом). Гранични регистри меморије ограничавају део меморије који је додељен процесу, а битови за контролу приступа указују које операције се могу извршавати.

Систем виртуелне меморије даје привид програмеру да располаже знатно већим меморијским простором од стварног. На тај начин величина програма се може повећавати преко величине оперативне меморије, а програмер се ослобађа бриге око смештања програма и података и израчунавања стварних, физичких адреса у оперативној меморији. У ту сврху користе се одговарајуће табеле. Кориснички програми и програми оперативног система могу бити померени у било који део меморије, а да се то не одрази на њихов правилан рад, све док се њихове табеле коришћених блокова (страница или сегмената) ажурирају сваки пут када дође до померања – релокације.

Но, све има своју цену, па и концепт виртуелне меморије. Неопходан је додатни хардвер у који се смештају адресе табела блокова или коришћење асоцијативног претраживања. Пре него што се дође до податка мора се више пута приступити меморији да би се одредила адреса податка. То додатно троши време.

Одређивање блока који треба избацити из меморије да би се унео нови блок у меморију, представља један од најсложенијих проблема. Но, многи проблеми губе на значају како расте брзина рада процесора и капацитет оперативне меморије (данас постоје рачунари са оперативном меморијом већом од 100 мегабајта).

Подаци смештени на јединицама секундарне меморије обично су организовани у датотеке. Датотеке су именовани скупови битова који су обично логички спрегнути, као што су повезане све инструкције које чине неки програм, или подаци који чине платни списак.

Организација датотека и приступ зависе од оперативног система. Неки системи допуштају само секвенцијално смештање, док други допуштају случајан приступ и комбинацију оба приступа. Подацима у датотекама може се приступити секвенцијално, путем кључа или индекса, или директно задајући њихову адресу. Расположиве методе зависиће од оперативног система.

Оперативни системи морају подржавати прављење и уништавање датотека. Он мора допустити приступ датотекама ради операција као што су читање, упис, додавање и ажурирање (модификација). Помоћу оперативних система могу се променити имена датотека, извршити њихово копирање, померање са једног места (директоријума или диска) на друго и слично.

Приступ датотеци обично се обавља путем директоријума (именика) који садржи листу имена датотека, показиваче на њихову локацију (најчешће почетак),

информације о величини, када је која датотека креирана, модификована или коришћена, и, често, и информације о приступним правима.

Приступна права одређују ко може користити датотеку и шта сме чинити са њом. У приступна права спадају: сазнање да датотека постоји, могућно извршавање датотеке (програма), читање њених записа, упис нових записа, модификација постојећих записа и, коначно, брисање датотеке.

Директоријуми могу бити организовани у један, два и више нивоа. Систем директоријума структуиран као стабло даје низ погодности и омогућава логичку организацију датотека. Ови системи обично оперишу командама као што су: креирање и уништавање директоријума, померање из једног директоријума у други, приказивање садржаја било ког директоријума (поддиректоријума). Датотеке и директоријуми смештају се на диск на много разних начина: у континуални (непрекидни) меморијски простор, посредством повезаних листа или мапа битова. Основно разумевање датотечког система и система директоријума треба кориснику рачунара да да представу како се подаци могу смештати, како им се и са којим циљем може приступити и како се могу заштити.

9.10. ПИТАЊА

1. У чему је разлика између програма и процеса?
2. Да би настао процес, и да би се он извршио, која су разна стања у којима се процес може наћи у току свог боравка у рачунару?
3. Како процес прелази из једног стања у друго?
4. Које се све информације чувају у контролном блоку процеса PCB?
5. Како централни процесор "прелази" са једног процеса на други?
6. Који су основни типови прекида?
7. Да ли се све врсте прекида на исти начун обрађују?
8. Описати улогу граничног регистра у заштити меморијског простора.
9. Које су предности и мане мониторских система?
10. Укратко објаснити како техника преклапања омогућава извођење великих програма у рачунарима са малом меморијом.
11. Објаснити употребу фиксних партиција у вишепрограмским системима.
12. У чему се састоји проблем фрагментације код фиксних партиција?
13. Који се проблеми јављају при додељивању меморије у фиксним партицијама?
14. У чему је разлика између система са фиксним и променљивим партицијама?
15. Шта омогућавају системи са виртуелном меморијом?
16. Како се израчунавају стварне адресе у системима са виртуелним меморијским простором?
17. Која је основна разлика методе страничења од метода сегментације?
18. Како се остварује веза између имена датотеке и њене физичке адресе?
19. Описати основне операције са датотекама: креирање, уништавање, отварање, затварање, читање, уписивање, додавање, промена имена и сл.

20. Које су основне операције са директоријумима?
21. Које информације садржи директоријум?
22. Како се помоћу приступних права врши заштита датотека?
23. Које су три основне методе за организовање директоријума?

9.11. КЉУЧНЕ РЕЧИ

- адресни простор (**address space**)
- анализатор прекида (**interrupt handler**)
- асоцијативне меморије (**associate memories**)
- бит–мапа (**bit map**)
- блокирање (**blocking**)
- датотека (**file**)
- директоријум (**directory**)
- додавање (**append**)
- додељивач (**dispatcher**)
- **FIFO (First-In First-Out)**
- гранични регистар (**fence register**)
- излазно стање (**stop, output state**)
- извођење (**run**)
- језгро, нуклеус (**kernel, nucleus**)
- кластер (**cluster**)
- контролни блок процеса (**PCB, Process Control Block**)
- копирање (**copy**)
- корен, главни директоријум (**root, master directory**)
- **LIFO (Last-In First-Out)**
- мапе блокова (**block maps**)
- маскирање (**masking**)
- немаскирани прекид (**nonmaskable interrupt**)
- отварање (**open**)
- партиције (**partition**)
- поддиректоријум **(subdirectory)**
- прекид (**interrupt**)
- преклапање (**overlay)**
- премотавање **(rewind)**
- претпражњење, испражњивање (**preemption**)
- процеси (**processes**)
- процесорски ограничени задаци, послови (**processor bounded jobs**)
- права приступа (**access rights**)
- пунилац (**bootstrap**)
- пут (**path**)
- распоређивач (**scheduler**)
- сегментација (**segmentation**)
- сектор (**sector**)
- спремност (**ready**)
- стања **(status**)
- страничење (**paging**)
- стратегија замене (**replacement strategy**)
- супервизор (**supervisor, supervisor call**)
- суспендован (**suspended**)
- табела сегмената (**segment table**)
- текући директоријум (**current directory**)
- улазно стање (**start, input state**)
- улазно–излазно ограничени послови, задаци (**I/O - bounded jobs**)
- упис (**write)**
- векторски прекид (**vectored interrupt**)
- виртуелна меморија (**virtual storage**)
- виртуелна адреса (**virtual address**)
- заштита датотеке (**file protection**)
- затварање (**close**)
- чекање (**wait**)
- читање (**read)**

10. РАЗВОЈ РАЧУНАРСКЕ ТЕХНИКЕ

10.1. ТЕХНОЛОШКЕ ЕРЕ

Када и где је настао први рачунар никада нећемо сазнати. Једно је извесно: абакус је као помоћно средство за рачунање у употреби више од 2000 година. На бази средстава помоћу којих се обавља рачунање, рачунари су прошли кроз неколико технолошких епоха. Те епохе су:

– *механичка ера до 1930. године,*
– *електромеханичка ера до 1944. године,*
– *електронске цеви од 1946. до 1954. године,*
– *дискретни полупроводници од 1955. до 1964. године,*
– *интегрисани полупроводници од 1965. па до данас.*

Упоредо са развојем технологије развијале су се и могућности рачунара. Једна од битних карактеристика рачунара је брзина рада, тј. број основних операција извршених у секунди. На слици 10.1 приказан је утицај технологије на брзину рачунара.

технологија	**датум**	**број операција у секунди**
механичка	1930.	1
електромеханичка	1940.	10
вакуумска цевна	1945.	10^3
транзисторска	1960.	10^6
интегрисана кола	1970.	10^8
интегрисана кола	2002.	10^{10}
интегрисана кола	2008.	10^{12}

Слика 10.1. Технологија и брзина рачунара

Прва машина способна да аутоматски обавља четири аритметичке операције направљена је 1623. на Универзитету у **Tubingenu**. Направио ју је професор **Wilhelm Schickhard**, али нема правих писаних докумената о тој машини.

Знатно већи утицај имала је машина коју је направио **Blaise Pascal** 1642. године, слика 10.2. То је био механички бројач за извођење сабирања и одузимања. Да би се остварило одузимање негативни бројеви су били представљени комплементима. Око 1671. године **Gottfried Leibniz** је конструисао калкулатор који је могао аутоматски да множи и дели.

година	изумитељ	могућности рачунања	техничка иновација
1642	**Pascal**	сабирање, одузимање	*аутоматски пренос, приказ бројева у комплементу*
1671	**Leibniz**	сабирање, одуз.имање множење, дељење	*корачни механизам за множење и дељење,*
1827	**Babbage**	метод коначних разлика	*аутоматске операције са више корака*
1834	**Babbage**	израчунавања опште намене	*механизам за аутоматску контролу низа операција (програм)*
1941	**Zuse: Z3**	израчунавања опште намене	*први рачунари опште намене*
1944	**Aiken: Harvard Mark I**		

Слика 10.2. Механичка ера у развоју рачунара

Једна од најзначајнијих особа у историји рачунских машина је **Charles Babbage**. Он је конструисао два рачунара:

– *диференцну машину 1823. године, и*

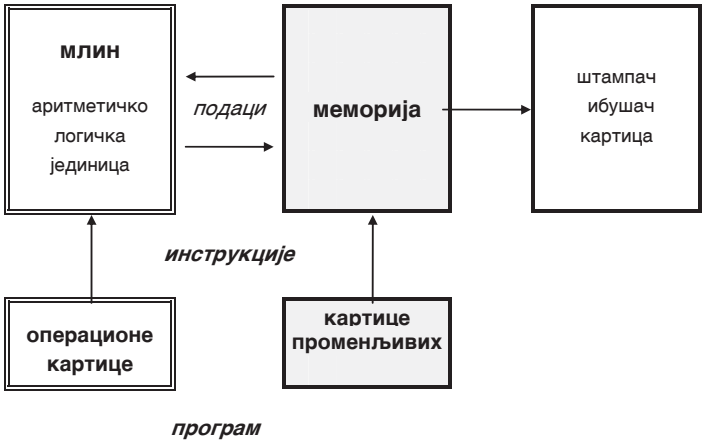
– *аналитичку машину 1834. године, која никад није завршена.*

Прва машина из 1823. године омогућавала је аутоматско рачунање у математичким табелама са штампањем резултата. Све операције су биле засноване искључиво на сабирању. Аналитичка машина је имала за циљ аутоматско извођење свих математичких операција. На слици 10.3. дата је општа структура коју је предочио **Babbage**. Она има два главна дела: **меморију** састављену од сета бројача и **"млин"** који одговара модерној аритметичко–логичкој јединици. За контролу низа операција **Babbage** је предлагао коришћење бушених картица.

Картице које чине програм подељене су у две групе:

1. **операционе картице**, *које бирају и контролишу свака по једну од четири могуће операције* (+, –, *, :), *у сваком кораку програма, при чему једна картица одговара једном програмском кораку,*
2. **картице променљивих**, *које бирају меморијску локацију која ће бити коришћена у појединим операцијама као извор операнада или као одредиште резултата.*

Константе су се, такође, могле задавати или помоћу бушених картица, или су се ручно уносиле у бројачка поља.



Слика 10.3. Структура **Babbage**–ове аналитичке машине

У току 19. века настао је низ комерцијалних механичких рачунских машина. Један од успешних у том послу био је Американац **Herman Hollerith**, који је 1896 основао **Tabulating Machine Company** за производњу ових уређаја. Године 1911. ова компанија се спојила са неколико других па је тако настала **Computing-Tabulating-Recording-Company**, која је 1924. године променила име у **Inernational Businese Machines Corporation (IBM).**

После **Babbage**–ове машине није било значајних покушаја за прављење дигиталних рачунара опште намене све до 1930. године. Тада су независно конструисани такви рачунари у Немачкој и Америци. Године 1938. **Konrad Zuse** је направио рачунар **Z1** који је користио бинарну уместо декадне аритметике. Исти научник је 1941. године направио први програмски контролисани рачунар опште намене помоћу релеа са бројевима у покретном зарезу.

Howard Aiken са универзитета **Harvard** предложио је 1937. израду електромеханичког рачунара опште намене. Посао је покренут у сарадњи са **IBM**–ом. Рачунар је касније добио име **Harward Mark I** (1939. год.), а прорадио је 1944. године. Био је сличан са **Babbageovom** машином, а користио је инструкције формата:

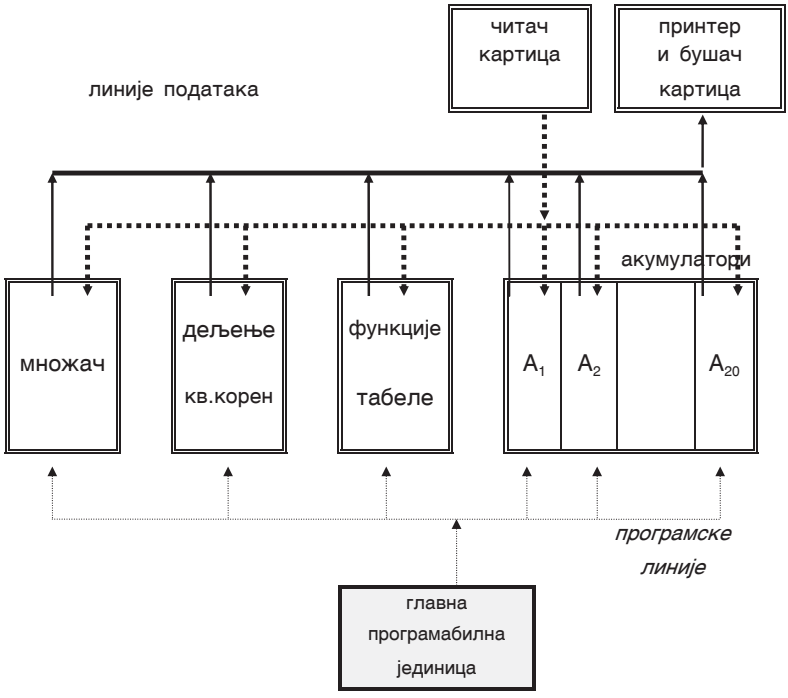
A1 A2 OP ,

где је **OP** операција, **A1** и **A2** су регистри операнада, а **A2** се уједно користио за смештање резултата.

Први покушај да се направи електронски рачунар са електронским цевима био је на државном универзитету **Iowa State** 1930. године од стране **John Atanasoffa**. То је била машина специјалне намене за решавање линеарних

једначина. **Atanasoff** је 1937–38 направио концепт електронске цифарске рачунске машине, тј. рачунара, на основу којег му је Врховни суд Америке 1973. године признао право патента, и сматра се да је он творац концепта модерних електронских рачунара.

Први електронски рачунар опште намене вероватно је био **ENIAC** направљен на универзитету **Pennsylvania** под вођством **John Manchly**–а и **Presper Eckerta**. Машина је прављена од 1943. до 1946. године и имала је 30 тона, преко 18000 вакумских цеви и трошила је око 200 **kW**. Структура **ENIAC-а** дата је на слици 10.4.



Слика 10.4. **ENIAC**

Године 1945., консултант на изради **ENIAC**–а, **John von Neumann**, предложио је да се програми и подаци смештају у исту меморију (**stored-program concept**). Главна предност овог концепта је могућност програма да сам модификује своје инструкције.

Први рачунар на овом принципу **EDVAC** разликовао се од својих претходника у неколико важних карактеристика. Имао је много већу меморију састављену од главне (капацитета од 1024 речи–бројева или инструкција) и секундарне (магнетна меморија капацитета око 20 K речи).

Користио је бинарни бројни систем и имао је серијску аритметичко–логичку јединицу.

Аритметичке инструкције су имале формат:

A1 A2 A3 A4 OP

где су **A1, A2** адресе операнада, **A3** адреса резултата, **A4** адреса следеће инструкције, дакле инструкције су биле четвороадресне.

Инструкција условног гранања имала је формат:

A1 A2 A3 A4 C

са следећим значењем: ако број на локацији **A1** није мањи од броја у **A2** узми инструкцију са локације **A3,** у противном узми инструкцију са **A4**.

EDVAC је почео са радом 1951. године.

John Von Neumann је, на институту **Princeton Institute for Advanced Studies**, са својим колегама 1946. године почео пројектовање новог рачунара који је назван **IAS**.

Машина је имала главну меморију направљену помоћу катодне цеви. За разлику од **EDVAC**–а ова машина је користила паралелне дигиталне мреже за реализацију елементарних операција. Свака инструкција садржала само једну меморијску адресу, а формат јој је био:

OP A .

Централни процесор садржао је неколико брзих регистара који су коришћени као имплицитна меморија. Све у свему, рачунар је имао веома модерну концепцију која је имала велики утицај на даљи развој рачунарске технике. На слици 1.3 приказана је типична архитектура рачунара из касних четрдесетих и раних педесетих година, тј. рачунара прве генерације.

Управљачка функција рачунара је централизована у једном процесору. Свака операција у систему, као на пример пренос једне речи између меморије и улазно–излазног уређаја, захтева директно учешће централног процесора.

У то време поред **IAS**–а, направљени су и многи други рачунари као на пример **WHIRLWIND I.** То је био први рачунар са главном меморијом направљеном од феритних језгара. **ATLAS** је био први рачунар са концептом меморије у једном нивоу која се данас зове виртуелна меморија. Компанија **Eckert-Mauchly Computer Corporation** је 1951. године направила рачунар **UNIVAC,** који је имао секундарну меморију у облику магнетне траке. **IBM 701** из 1953. године био је рачунар са главном меморијом у облику катодне цеви, и са магнетном траком и добошем као секундарним меморијама.

У првим рачунарима програми су били писани у бинарном коду, тј. на машинском језику. Писање машинских инструкција је било напорно, а оне саме су се јако тешко распознавале, тако да су се већ у раним педесетим годинама почеле да користе симболичке инструкције облика **ADD X1**. То је симболичко програмирање, а овај језик добио је назив **ASSEMBLER**.

Употреба асемблера знатно је олакшала програмирање, али је морао да постоји специјални системски програм назван асемблер чији је задатак био да преведе програм са симболичког на машински језик непосредно пре извршавања. Ове машине су биле једнопрограмске, па је централни процесор често био докон, чекајући да се изврше улазно–излазне операције на спорим периферијама.

10.2. ЕЛЕКТРОНСКИ ЦИФАРСКИ РАЧУНАРИ

10.2.1. РАЧУНАРИ ПРВЕ ГЕНЕРАЦИЈЕ

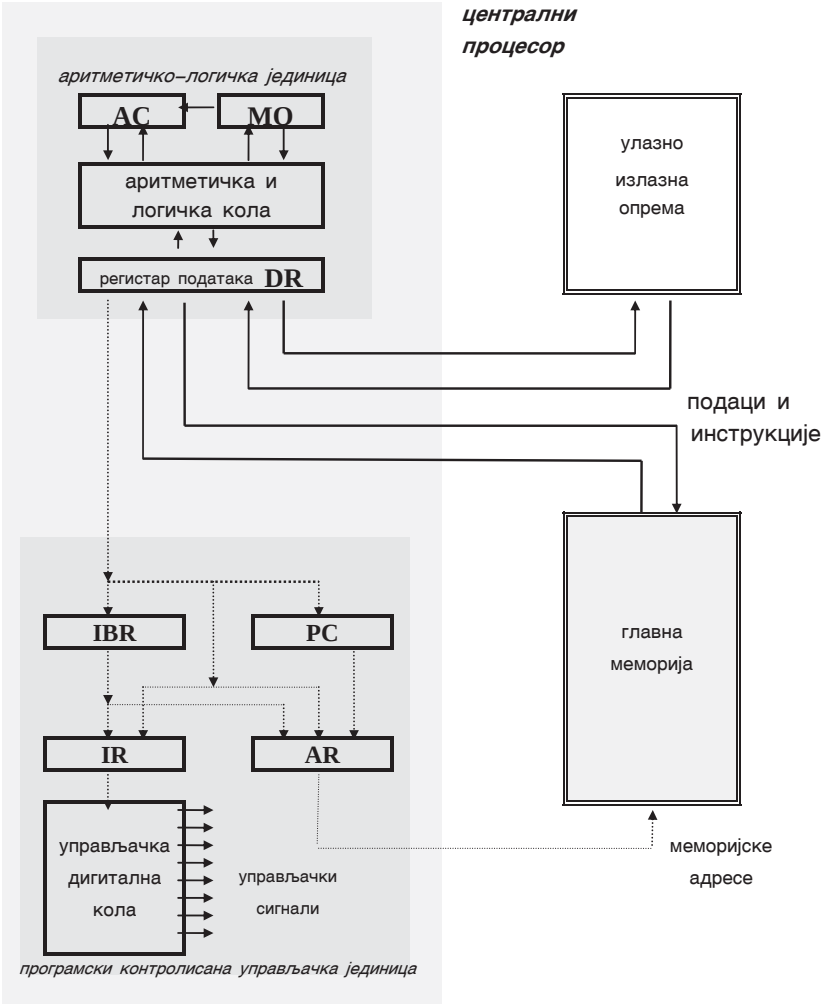
Од 1946. до 1948. године **A.W.Bruks, H.H.Goldstine** и **John Von Neumann** описали су логички концепт и програмирање **IAS** рачунара чија је шема приказана на слици 10.5.

IAS је имао реч дужине 40 бита која се у једном такту преносила из меморије у **CPU**. Имала је меморију од 2^{12} =4096 речи, а реч је могла садржати у себи податак или инструкцију, слика 10.6.

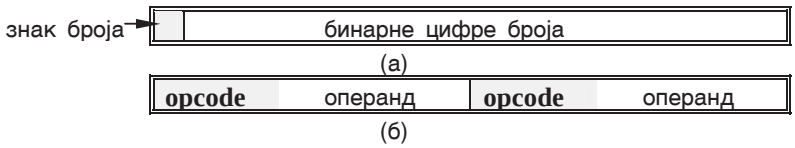
Основни тип података био је број у фиксном зарезу, при чему је бит највеће тежине био бит знака броја. Формат меморијске речи рачунара **IAS** дат је на слици 10.6 а). Коришћен је други комплемент да би се помоћу сабирача реализовало одузимање. Децимална тачка је била између битова 0 и 1, односно сви бројеви су били између –1 и +1 (бројеви су били скалирани). Једна реч је садржала две инструкције од по 20 бита, при чему је **код операције** био 8 бита, а преосталих дванаест бита представљали су адресу меморијске локације, слика 10.6 б).

Централни процесор је поред аритметичке и контролне јединице садржао и сет врло брзих регистара за привремено памћење инструкција, меморијских адреса и података. Главна меморија је коришћена за смештање програма и података, а пренос једне речи обављао се увек између четрдесет битног регистра података **DR** у **CPU** и било које локације у меморији са адресом **X**. Адреса локације којој се приступа уписивана је у 12–битни адресни регистар **AR. DR** је могао да се користи за смештање једног операнда током извођења инструкција. За привремено смештање операнада и резултата коришћена су два регистра: акумулатор **AC** и **multiplier-quotient** регистра **MQ**. Свака инструкција изводила се у две фазе: **фаза**

прибављања и **фаза извршења**. У току фазе прибављања из меморије се преносе две инструкције одједном и пребацују се у контролну јединицу. Инструкција која се неће одмах извршавати смештала се у прихватни регистар инструкција **(Instruction Buffer Register**, **IBR**). Код операције друге инструкције одмах се смешта у регистар инструкција (**instruction register**, **IR**), где се врши декодирање. Поље адресе текуће инструкције се преноси у меморијски адресни регистар (**AR**). Овај рачунар је имао још један адресни регистар који се звао инструкцијски адресни регистар или програмски бројач (**PC**) и користи се за смештање адресе следеће инструкције која ће бити извршена.



Слика 10.5. Општа шема рачунара **IAS**



Слика 10.6. Формат меморијске речи рачунара **IAS**

Сет инструкција био је подељен у пет група:
– *инструкције преноса података,*
– *инструкције безусловног скока,*
– *инструкције условног скока,*
– *аритметичке инструкције и*
– *инструкције за модификацију адреса.*

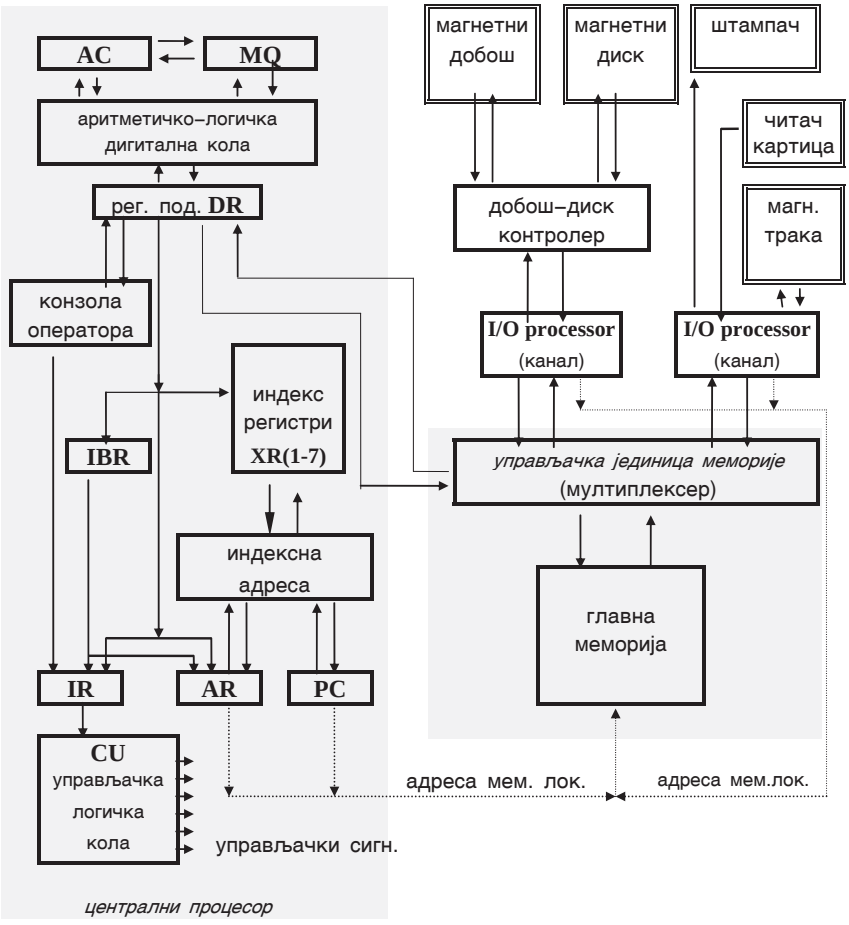
10.2.2. РАЧУНАРИ ДРУГЕ ГЕНЕРАЦИЈЕ

Рачунари ове генерације настали су у периоду 1955.–1964. године, а главна карактеристика је прелазак са вакуумских цеви на транзисторску технологију. Транзистори су измишљени 1948. године у **Bell Telephone Laboratories** и одмах су нашли широку примену управо у рачунарској техници. Транзистори су експериментално примењени први пут у рачунару **TX-0** који је прорадио 1953. године. Но, сем употребе транзистора, развијено је и неколико значајних новина у рачунарској техници и софтверу, које се могу систематизовати у следећем:

1. *транзистори се користе за израду прекидачких кола,*
2. *масовна употреба феритних језгара и магнетних добоша за реализацију главних меморија,*
3. *употреба индекс регистара и хардвера који подржава аритметику у покретном зарезу,*
4. *развој машински независних програмских језика високог нивоа као што су:* ***ALGOL, COBOL*** *и* ***FORTRAN****,*
5. *надзор улазно–излазних операција препуштен је специјалним* ***I/O*** *процесорима који ослобађају* **CPU** *од спорих рутинских послова,*
6. *произвођачи рачунара почињу да нуде системски софтвер као што су компајлери, библиотеке потпрограма и монитори пакетне обраде.*

Године 1956 направљен је **IBM 704,** цевни рачунар који је имао индексне регистре и хардвер за аритметику у покретном зарезу. То је уједно била прва комерцијална машина са контролним програмом који је представљао зачетак оперативних система. Каснији модели овог рачунара и његов наследник **709** имали су улазно–излазне процесоре који су, касније, названи

каналима. Говорећи о другој генерацији рачунара неопходно је говорити о рачунарским системима, јер се користи већи број: меморијских јединица, процесора, улазно–излазних уређаја. Типичан пример друге генерације рачунара је **IBM 7094**, чија блок шема је дата на слици 10.7.



Слика 10.7. Блок шема рачунара **IBM 7094**

Његов централни процесор, слика 10.7, разликовао се од **IAS**–а углавном због употребе седам индекс регистара и аритметичких кола која су подржавала операције у фиксном и покретном зарезу. Све улазно–излазне операције контролисао је скуп улазно–излазних процесора који су имали директни приступ главној меморији. Коришћене су методе индексног и

индиректног адресирања. **7094** је имао више од 200 инструкција које се могу класификовати на следећи начин:

1. *инструкције за пренос података којима се преноси једна информацијска реч између **CPU** и меморије, или између два регистра у **CPU**,*
2. *аритметичке инструкције у фиксном зарезу,*
3. *аритметичке инструкције у покретном зарезу,*
4. *логичке (ненумеричке) инструкције,*
5. *инструкције за модификацију индекс регистара,*
6. *инструкције условног и безусловног гранања,*
7. *улазно–излазне операције (неке је извршавао **CPU**, а већину **IOP**).*

Улазно–излазни пренос података заснивао се на употреби специјалних контролних сигнала који су се звали прекиди, а који су омогућавали да **CPU**, када је то потребно, брзо решава проблеме улаза и излаза које потом препушта **IOP** процесорима. Главна карактеристика режима рада била је пакетна обрада. У то време направљени су велики рачунарски системи популарно названи супер рачунари, а најпознатији су били **LARC,** фирме **UNIVAC** и **STRETCH (7030)** фирме **IBM**. Они су користили технике **паралелне обраде** (**parallel-processing**) у облику истовременог прибављања и извршавања разних инструкција једног програма (**мултипроцесирање**) и истовременог извршавања инструкција више разних програма (**мулти-програмирање**).

10.2.3. ТРЕЋА ГЕНЕРАЦИЈА РАЧУНАРА

Почиње од 1965. године и траје и данас. Наиме, 1964 године појавила су се интегрисана кола малог степена интеграције–**SSI** (**Small Scale Integration**). То је било логичко коло направљено у једном чипу. Већ 1968 године појавила су се интегрисана кола средњег степена интеграције–**MSI** (**Medium Scale Integration**), односно, регистар у једном чипу. Године 1971. на тржишту су се појавила интегрисана кола високог стапена интеграције–**LSI** (**Large Scale Integraton).** То су били: динамичка **RAM** меморија од **1 Kbit**, серијско–паралелни претварач **UART** (**Universal Asynchronous Receiver-Transmiter**) и први четворобитни микропроцесор **Intel 4004**. Но, прави почетак ере микропроцесора представља, у ствари, 1972. година када је фирма **Intel** изнела на тржиште микропроцесор **8008**. То је уствари био неуспели резултат развоја по наруџбини фирме **Datapoint**, јер је постављене пројектне задатке обављао десет пута спорије од захтеваног. Ова фирма је отказала уговор, а **Intel** је свој промашај изнео на тржиште, уместо да га баци на ђубриште. Показало се да је управо таква једна компонента недостајала тржишту, јер је имала вишеструку примену, а омогућила је реализацију врло јефтиних микрорачунара. Неки аутори сматрају да су

рачунари развијени у **LSI** технологији уствари посебна генерација–**четврта генерација** рачунара. Њихове основне особине би биле: мала потрошња, захтевају мало одржавања, просторије не морају бити климатизоване, јефтини су и погодни за реализацију малих и средњих рачунара.
Неколико година касније појављују се интегрисана кола врло високог степена интеграције–**VLSI (Very Large-Scale Integration)** са више од **450000** транзистора у једном чипу, што је еквивалент више од **100000** логичких кола. То је почетком 1980. године омогућило појаву динамичке меморије од **256 Kbit-a** по чипу, са временом приступа мањим од 20 наносекунди, као и 32–битних микропроцесора који су радили на 16 **MHz**. Већ средином 1980. године појавили су се чипови са више од милион транзистора, док се данас производе, и на слободном су тржишту, чипови са више од 30 милиона транзистора. Неки аутори сматрају четвртом генерацијом рачунаре произведене помоћу интегралних кола високе интеграције, а петом генерацијом рачунаре на бази интегралних кола врло високе интеграције. Неки аутори такође сматрају да су рачунари реализовани у овој технологији посебна генерација рачунара–**пета генерација**. Ова генерација је још увек у развоју, а њене основне карактеристике би биле: нова, високо паралелна архитектура (стотине и хиљаде процесора који паралелно раде и још већа брзина, и при томе ће рачунар (попут човека) бити свестан информационог садржаја података које обрађује, тј. омогућиће обраду знања.

Природно, нове технологије су битно утицале на развој рачунарских система, смањиле су њихову цену и самим тим проширile области њихове употребе. Где су границе развоја рачунарске технике тешко је рећи ако се има у виду да су већ реализована **CMOS** кола која раде на напону напајања од **400 mV,** дисипирају снагу од **1 W**–а, раде на учестаности већој од **100 MHz** и садрже преко **100 милиона логичких кола** у чипу чије су димензије 5 **X** 5 **cm**. Но, треба рећи да се већ почетком осамдесетих технологија приближила природним ограничењима. Код врло малих транзистора не може се са сигурношћу очекивати кретање електрона по планираној путањи, него ће прескочити на суседну (релација неодређености).

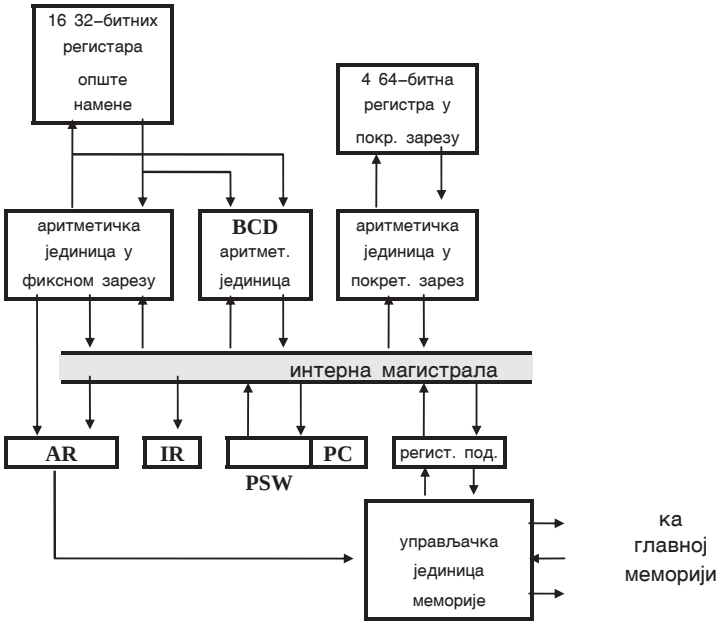
У Америци је у току реализација програма **VLSI** кола врло високе брзине–**VHSI (Very High-Speed Integration**), а у Јапану држава подржава индустријски програм развоја који је познат под именом **пета генерација рачунарских система,** док Европска заједница има свој сопствени програм назван **ESPRIT**.

Главне карактеристике рачунара треће генерације су:

1. *масовна употреба интегралних кола чиме се смањују физичке димензије и цена рачунара,*
2. *доминирају полупроводничке оперативне меморије,*

3. *употреба микропрограмирања ради поједностављења процесора и повећања флексибилности,*
4. *велики развој техника конкурентног и паралелног процесирања попут обраде на текућој траци, мултипрограмирања и мултипроцесирања,*
5. *развој разних метода за аутоматску деобу ресурса рачунарског система (снажан развој оперативних система).*

Типичан представник рачунара треће генерације је **IBM 360** чији је централни процесор приказан на слици 10.8. Овај рачунар је углавном реализован помоћу дигиталних кола **RCA** серије 70 из 1964 (**Radio Corporation of America**). На сличан начин су реализовани и рачунари из серије 370. Напоменимо да су неки модели ових рачунара имали микропрограмирану управљачку јединицу и од тада се овај концепт јако много користи. Овај концепт је први пут обрађен још 1951. године у радовима **Maurice V. Wilkesa**, и примењен је био већ у неким рачунарима прве и друге генерације, али без већег успеха.



Слика 10.8. Централни процесор рачунара **IBM 360**

Најзначајнији представници рачунара на почетку ере треће генерације поред **IBM** 360 и 370 су вишепроцесорски рачунари **CRAY-1** и **CDC Cyber 205** (најбржи рачунари до сада), затим **CDC 6600** и **CDC 7600** (1964. и 1969. године) које карактерише употреба великог броја улазно–излазних процесора (периферијских процесора) са великим степеном аутономије. **CDC**

STAR-100 је имао централни процесор са архитектуром текуће траке. Рачунар **ILLIAC IV** је имао 64 одвојене аритметичко–логичке јединице (које је надзирала једна контролна јединица) које су могле паралелно и независно да обрађују податке.

Средином 60–тих година почео је развој, а касније и масовна примена мини и микрорачунара чија је главна карактеристика кратка реч од 8 до 32 бита, ограничен хардвер и мале физичке димензије. Ови рачунари су имали врло ниску цену и омогућили су велику примену рачунара у индустријским постројењима и процесној техници, и врло брзо су потиснули контролере реализоване помоћу дигиталних логичких кола, јер су имали низ предности.

Посебан напредак у развоју рачунарске технике изазвала је појава микропроцесора, и на бази њих развијених микрорачунара врло ниске цене. Типична конфигурација таквог микрорачунара дата је на слици 10.9.

Слика 10.9. Типична конфигурација микрорачунара

Појава микропроцесора је знатно утицала и на архитектуру великих рачунарских система доводећи до појаве мулти–микропроцесорских рачунара и дистрибуираних система обраде који у свом раду обједињују и више стотина микрорачунара који су између себе врло удаљени. Основни проблеми реализације ових система су комуникације базиране на серијском преносу података. У ову сврху развијени су специјални микропроцесори који омогућавају брз серијски пренос. Употреба микропроцесора обезбеђује једноставно уношење и анализу свакодневних података, комуницирање са

рачунаром или неким мерним инструментима. Микропроцесори су омогућили знатно једноставнију реализацију управљања и праћења стања разних уређаја, смањили су комплексност разних дигиталних логичких мрежа које су се у ту сврху користиле и знатно растеретили централни рачунар од обављања разних функција у периферијским уређајима. На овај начин дошло је до дистрибуције функција између централног процесора и микропроцесора који управљају периферијским уређајима. С друге стране, појавом микропроцесора омогућен је суштински заокрет у филозофији приступа изградњи и управљању у разним областима технике: кућним апаратима, аутомобилима, аудио и видео техници, регулацији, телекомуникацијама и слично.

Примена микрорачунара има одређене карактеристике које их, углавном, разликују од примене великих рачунара. Микрорачунар се посматра као део неког великог система који треба да обавља одређене фиксне функције. Највећи број примена је такав да одговори микрорачунара морају да се добију у фиксном интервалу времена, тј. раде у реалном времену. Микрорачунари углавном обављају разне врсте надзора и управљања процесима, а релативно мали број примена захтева обављање сложених аритметичких операција. Примена микрорачунара је мање стандардизована од примене великих рачунара.

10.3. МИКРОПРОЦЕСОРИ И МИКРОРАЧУНАРИ

Последњих деценија праву револуцију у примени рачунара изазвали су микропроцесори. Први комерцијални микропроцесор **INTEL 4004** појавио се 1971. године. То је био четворобитни паралелни процесор. Убрзо након тога су се појавили осмобитни и шеснаестобитни микропроцесори који су нашли огромну примену која је резултирала у масовној производњи и врло ниској цени. Глвни представници осмобитних микропроцесора су **INTEL 8080**, направљен 1973. године, затим 1974. **ZILOG Z80** и **MOTOROLA 6800**, итд. Преглед развоја микропроцесора и меморија, као и њихових карактеристика у првој деценији њиховог постојања дат је у табели на слици 10.10. Док су први микропроцесори били сасвим једноставне архитектуре са организацијом око једне магистрале, са развојем технологије и повећањем степена интеграције у микропроцесоре су пресликавана решења из централних процесора великих рачунара. Данас микропроцесори садрже велики број регистара специјалне и опште намене; неки имају архитектуру текуће траке; неки подржавају вишепрограмски и вишекорисnички рад. Микропроцесори се по својим карактеристикама приближавају, па чак и превазилазе процесоре некадашњих рачунара средње снаге. Захваљујући врло ниској цени, рачунари базирани на микропроцесорима или микрорачунари, нашли су велику практичну примену и већ има неколико десетина милиона корисника ових рачунара. На

тржишту су најраспрострањенији једнокорисnички–лични микрорачунари **PC IBM** (**Personal Computer**) и њима компатибилни рачунари који раде у окружењу оперативних система **DOS** (**Disk Operating System**). Повећање степена интеграције довело је до реализације личних рачунара врло малих димензија, величине акт–ташне или мало веће бележнице (**laptop** и **note book**, слика 10.11). За **PC** рачунаре је у последњих десетак година развијено мноштво корисних програма у разним областима примене, који су створили основу за даље ширење поља примене микрорачунарских система.

тип	карактеристика	1972	1974	почетак 1978	крај 1978	1979	1981
CPU	1. типични представник	**i8008**	**I8080 Z80 MC6809**	**i8086**	**Z8000**	**MC68000**	**iAPX 432**
	2. дужина речи (бит)	8	8 (16)	сви 16	16 и 32	16 и 32	32
	3. брзина (такт **MHz**)	0.5–0.8	2–3	5–8	4–10	8	8
	4. број регистара	7 (8 бита)	7 (8 бита)	8 (16 бита)	16 (16 бита)	16 (32 бита)	16 (32 бита)
меморије	5. физички адресни простор у бајтима	16 K	64 K	1 M	8 M	16 M	16 M
	6. број сегмената	–	–	4 (64K) фиксно	64 **MMU** (до 64K)	64 **MMU** (до 64K)	
физичке	7. технологија	**PMOS**	**PMOS/ NMOS**	**NMOS**	**NMOS**	**NMOS**	**HMOS**
карakt.	8. транзистора/ чипу	2000	4500	20000	17500	68000	100000
	9. кашњење (**nsec**)	30	15	3	1–10	1–10	1–10

MMU Memory Menagement Unit -јединица за управљање меморијом

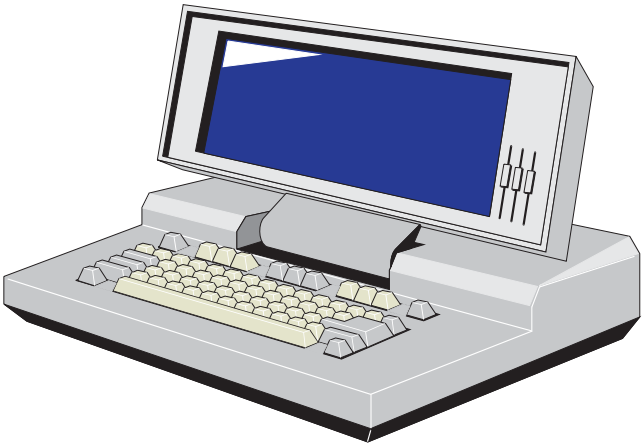
Слика 10.10. Развој главних карактеристика микропроцесора

Програми за **PC**–рачунаре су тако пројектовани да омогућавају употребу рачунара корисницима који апсолутно не морају да знају ишта о рачунарима или о програмирању, то јест створено је такозвано **user friendly** окружење. Но, увек треба имати на уму да су то, по правилу, недорађени, преобимни програми, урађени на вишим програмским језицима, намењени великом броју корисника па су препуни разних могућности које се тешко памте. Због тога смо сведоци да се свакодневно појављују нове верзије тих истих програма.

Микропроцесори (и микрорачунари) су већ прошли кроз неколико фаза свог развоја које такође називамо генерацијама.

Прву генерацију представљају микропроцесори **Intel 4004 & 4040**, **Fairchild PPS-4**, **National IMP-4, Rockwell PPS-4** и **Microsystem MC-1**. Били су, четворобитни, са 2^{12} адресним простором, паковани у 16–пинска кућишта (са 16 ножица), па су се многе ножице користиле у разне сврхе у разним

тренуцима времена (временско мултиплексирање). Затим су уследили осмобитни: **Intel 8008, IMP-8, PPS-8, AMI 7200**, али и 16–битни **IMP/16** и **National PACE.** Коришћени су у једноставним индустријским применама, као и у уређајима широке намене.



Слика 10.11. Лични рачунар као бележница

Другу генерацију карактерише повећање димензија чипа на 40–пинско паковање, за знатно већим меморијским адресним простором, са могућношћу прикључења до 256 улазно–излазних склопова, са већим скупом инструкција, потпрограмима, прекидним механизмом, итд. Типични представници осмобитних микропроцесора били су: **Intel 8080/8085, Fairchild F-8, National CMP-8, RCA COSMAC, MOS tech 6500** (највише произведени микропроцесор до сада, коришћен за ТВ игре), **Zilog Z80, Motorola 6800/6809**. Прављени су и 12–битни микропроцесори попут **Intersil 6100** и **Toshiba TLCS-12,** као и **16**-битни**: TI TMS 9900, DEC MCP-1600** и слично**.** У ову генерацију спадају и такозвани бит–одрезак (**bit-sliced**) микропроцесори. Прављени су у **TTL** биполарној технологији која има велику дисипацију, али и огромну брзину. Карактеристика ових процесора је да им је функција управљања и обраде подељена у два разна чипа. Аритметичко–логичка јединица није фиксне архитектуре већ је сам корисник прави из делова који садрже неколико регистара и део за обраду (налик на малу **ALU**). Свака јединица за обраду врши независну обраду једног дела податка. На бази ових микропроцесора реализовани су врло квалитетни професионални уређаји специјалне намене са речима велике дужине и флексибилне архитектуре. Типични представници су: **Motorola MC 10800, Intel 3000, Fairchild 9400, Texas Instrumants SPB 0400.**

Трећу генерацију започиње 1978 године **Intel** својим 16–битним процесором 8086, а продужавају: **Zilog Z80, Motorola 68000, National NS16016** и **Texas Instruments TMS 99000.** Паковања су 48–пинска и 64–пинска, изузев **Z**8000 сви су микропрограмирани, **ALU** обавља све четири елементарне операције, а могу да адресирају од 1**MB** до **16MB**, имају виртуелну меморију од **48 MB** (мегабајта), још моћнији механизам прекида, врло флексибилан улаз–излаз података као и привилеговане инструкције. Ови микропроцесори су били основа за израду микрорачунара који су могли да се користе у пословне сврхе, али и за обраду података, управљање у реалном времену, дистрибуирану обраду у рачунарским мрежама, итд.

Четврта генерација су 32–битни микропроцесори, који се израђују почев од 1981. са појавом на тржишту **Intel iAPX 432** система, мада су рачунари са **Motorola 68000** већ били 32–битни, па их можемо посматрати као међукорак. Касније су уследили многи други 32–битни микропроцесори: **National Semiconducter NS32032** (68 пинова), **Zilog Z80000** (68 пинова) **Intel 80386** (132 пина), **Motorola MC 68010 / 68020 (114** пинова**),** затим **i 80486** и **MC68040** (друга генерација 32–битних микропроцесора). Карактеришу се даљим повећањем могућности рачунања (аритметика у покретном зарезу), већим паралелизмом у раду, повећањем адресних могућности, адресним простором од 4**G** бајта, повећањем броја регистара, потпуно развијеним хардвером који подржава вишепрограмски и вишекориснички рад и већом брзином рада (такт на преко 300 **MHz**).

Пету генерацију чине 64–битни микропроцесори који су се појавили на тржишту 1995 и касније. У овом периоду развијена је читава серија **RISC** микропроцесора од стране разних произвођача: **Hewlet Packard, Inmos, Acron, Motorola MC88100, DEC alpha** (**21064**), и **Intel i860** фамилија. Ови микропроцесори имају 128–битну унутрашњу магистралу, основни такт код неких је преко 3 **GHz**, све инструкције се извршавају у једном такту. Већ постоје преводиоци за скоро све више програмске језике, па ће ови микропроцесори све више преузимати тржиште, а већ су на бази неких од њих реализовани персонални микрорачунари попут **Intel**–овог **Pentium** и **Power Mac**. Број транзистора у процесорима расте из дана у дан, па тако данашњи **MMX Pentium** има преко 4,5 милиона, а предвиђа се да ће процесори ускоро имати преко милијарду транзистора, радиће на фреквенцији од 10**GHz**, са брзином од око 1 000 000 **MIPS**–а (данашњи **Pentium IV** ради на 2,50**GHz**, са 321 **MIPS**).

На почетку ере, микропроцесор се употребљавао као један од мањих подсистема у терминалима, калкулаторима и комуникационим уређајима. Микропроцесор је нашао велику примену у управљачким и информационим системима. Уграђује се у комуникационе уређаје, уређаје за прикупљање

података, у системе за даљинско управљање, у интелигентне терминале, у системе за управљање саобраћајем, у уређаје за обраду текстова, управљачке склопове периферних јединица, у уређаје у системима заштите и безбедности, у опрему за тестирање и испитивање, у медицинске инструменте, користе се у TV играма итд. Микропроцесор се користи за реализацију микрорачунарски контролисаних система када постоји потреба за реализацијом више управљачких функција и функција доношења одлуке, када се захтева флексибилност, када функције нису до краја дефинисане и очекује се придодавање нових функција, када се рукује са много података и када се контролише велики број логичких стања. Посебну класу микрорачунара базираних на микропроцесорима чине лични (персонални) рачунари.

10.4. ЗАКЉУЧАК

Основна карактеристика развоја рачунарских система је да рачунари постају све бржи, мањих димензија и све јефтинији. Створене су многе нове технологије, развијена нова архитектурна решења, реализовани високо интегрисани дигитални логички склопови и меморије и створен велики број периферијских уређаја.

Рачунари су у почетку првенствено били намењени решавању сложених научно техничких проблема које је било тешко, или чак немогуће, решити до тада постојећим средствима. Током времена рачунари су постепено почели да улазе и у друге области рада, али и живота, где год је било потребно да се великом брзином обраде велике количине података. То су најчешће разни облици пословне обраде података, надзор и управљање индустријским процесима, вештачка интелегенција, али и писање књига, писама, вођење евиденције и слично. Како су се ширили правци примене рачунарских система, тако се све више људи укључивало у коришћење рачунарске опреме.

Посебан значај у развоју и ширењу примене рачунара имају микрорачунари и микропроцесори. Микропроцесори су омогућили реализацију врло јефтиних микрорачунара који имају велику брзину рада и велики опсег примена. Истовремено, микропроцесори су омогућили стварање моћних рачунарских система, базираних на истовременом кооперативном раду више стотина микропроцесора, тј. створени су мултипроцесорски системи у којима се размена информација између процеса, који се извршавају на појединим микропроцесорима врши путем магистрале и коришћењем заједничке меморије. Последњих година у снажном су развоју рачунарски системи са дистрибуираном обрадом код којих су поједини рачунари међу собом физички знатно удаљени.

10.5. ПИТАЊА

1. Кроз које су све технолошке ере прошли рачунари у свом развоју?
2. Како је технологија утицала на карактеристике рачунара?

3. У чему је значај **Babbage**–ове аналитичке машине?
4. У чему се **EDVAC** разликовао од претходних рачунара?
5. Који је рачунар **I** генерације и зашто имао пресудни утицај на развој рачунара?
7. Које су карактеристике рачунара **II** генерације?
8. Чиме се карактеришу и које су се технологије користиле у реализацији рачунара **III** генерације?
9. Како су микрорачунари и микропроцесори утицали на развој и примену рачунара?
10. Кроз колико технолошких периода су прошли микропроцесори?
11. Где су се у почетку углавном користили микропроцесори?
12. Да ли се микропроцесори користе само за израду микрорачунара?

10.6. КЉУЧНЕ РЕЧИ

- акумулатор (**accumulator, AC**)
- Бабиџова анлитичка машина (**Babbages analitical machine**)
- биполарна технологија (**bipolar technology**)
- бит–одрезак (**bit-sliced**)
- **CDC (Control Data Corporation)**
- **DOS (Disk Operated System)**
- друга генерација (**second generation**)
- други комплемент (**two`s complement**)
- **EDVAC (Electronic Discrete Variable Computer)**
- електромеханички рачунари (**electromechanical computers**)
- електронске цеви (**vacuum tubes**)
- електронски рачунари (**electronic computers**)
- **ENIAC (Electronic Numerical Integrator And Calculator)**
- **IAS**
- **ILLIAC IV**
- **IBM (International Business Machines)**
- интегрисана кола (**Integrated Circuits, IC**)
- лични, персонални рачунар (**Personal Computer, PC**)
- мали степен интеграције (**Small-Scale Integration, LSI**)
- механичке рачунске машине (**mechanical computing machines**)
- меморијски адресни регистар (**Address Register, AR**)
- микропроцесор (**microprocessor**)
- микрорачунар (**microcomputer**)
- програмски језици високог нивоа (**High-Level Languages**)
- прва генерација (**first generation**)
- регистар **MQ** (**Multiplier-Quotient**)
- регистар инструкција (**Instruction Register, IR**)
- средњи степен (ниво) интеграције (**medium-scale itegration, MSI**)
- **STAR-100**
- транзистори (**transistors**)
- трећа генерација (**third generation**)

- високи степен интеграције (**Large Scale Integration, LSI**)
- врло високи степен интеграције (**Very Large Scale Integration, VLSI**)

ЛИТЕРАТУРА

[1] Alexandridis, N.A., "Microprocessor Systems-Architecture and Engineering", Infortech State of the Art Report on Microeletronics, Pergamon Infotech, series 8, no. 2, 1980.

[2] Alexandridis, N. A., "Microprocessor System Design Concepts", Computer Science Press, Rockvile, 1984.

[3] Crook, C., "Microcomputer Architecture-A Survey Report", Infortech State of the Art Report on Microelectronics, Pergamon Infotech, series 8, no. 2, 1980.

[4] Davis, S., "Microprocessors", EDN, August 5, 1979.

[5] Hayes J. P., "Computer Architecture and Organization", McGraw-Hill International Book Company, 1983.

[6] Hilburn J.L. and E. Teja, " As You Get to Know the 8086, Use Your 8-bit Experience", EDN, January 20, 1979.

[7] Intel Corporation, 8080 Microcomputer System User`s Manual, Santa Clara, CA, September 1975.

[8] Јауковић М. "Увод у информационе системе", Техничка књига, Београд, 1992.

[9] Johnson, R.C., "32-bit Microprocessors Inherit Mainframe Features", Electronics, February 24, 1981.

[10] Кватерник Р. "Увод у оперативне системе", Информатор, Загреб, 1988.

[11] LeMair, I., and R. Nobis, " Complex Systems are Simple to Design (with the MC68000)", Electronic Design, 18, September 1, 1978.

[12] Luce T. "Computer Hardware, System Software, and Architecture" Mitchell Publishing, Inc. Watsonville, CA , 1989.

[13] Madnick S. and Donavan J., "Operating Systems", McGraw-Hill, New York, 1974.

[14] Мијалковић М. "Програмирање MSC96 серије микроконтролера", Виша електротехничка школа, Београд, 2002,

[15] Motorola, Incorporated, 16-Bit Micrprocessing Unit, Austin, TX, 1980.

[16] Николић М & Николић М. "Диск оперативни систем МС ДОС 5.0" Техничка књига, Београд, август 1992.

[17] Osborne, A. "An Introduction to Microcomputers, Volume II, Some Real Products", Berkeley, CA: Adam Osborne and Associates, Inc., 1976.

[18] Рибарић С. "Архитектура микропроцесора", Техничка књига, Загреб, 1985.

[19] Смиљанић Г. "Основе дигиталних рачунала", Школска књига Загреб, 1978.

[20] Станковић М. и Станковић М. "Рачунарски системи", Завод за уџбенике и наставна средства, Београд, 1990.

[21] Стојковић В. и Тошић Д. "Програмски системи I део", Научна књига, Београд, 1979.

[22] Stritter, E., and T. Gunter, " A Microprocessor Architecture for a Changing World: The Motorola 68000", Computer, February 1979.

symbol име карактера	ASCII Octal	Hex	Dec	symbol име карактера	ASCII Octal	Hex	Dec	symbol име карактера	ASCII Octal	Hex	Dec
NULL	00	00	00	,	54	2C	44	V	126	56	86
SOH	01	01	01	- minus	55	2D	45	W	127	57	87
STH	02	02	02	.	56	2E	46	X	130	58	88
ETX	03	03	03	/	57	2F	47	Y	131	59	89
EOT	04	04	04	0	60	30	48	Z	132	5A	90
ENG	05	05	05	1	61	31	49	[	133	5B	91
ACK	06	06	06	2	62	32	50	\	134	5C	92
BELL	07	07	07	3	63	33	51	]	135	5D	93
BS	10	08	08	4	64	34	52	^	136	5E	94
HT	11	09	09	5	65	35	53	- underline	137	5F	95
LF	12	0A	10	6	66	36	54	`	140	60	96
VT	13	0B	11	7	67	37	55	a	141	61	97
FF	14	0C	12	8	70	38	56	b	142	62	98
CR	15	0D	13	9	71	39	57	c	143	63	99
SO	16	0E	14	:	72	3A	58	d	144	64	100
SI	17	0F	15	;	73	3B	59	e	145	65	101
DLE	20	10	16	<	74	3C	60	f	146	66	102
DC1	21	11	17	=	75	3D	61	g	147	67	103
DC2	22	12	18	>	76	3E	62	h	150	68	104
DC3	23	13	19	?	77	3F	63	i	151	69	105
DC4	24	14	20	@	100	40	64	j	152	6A	106
NAK	25	15	21	A	101	41	65	k	153	6B	107
SYN	26	16	22	B	102	42	66	l	154	6C	108
ETB	27	17	23	C	103	43	67	m	155	6D	109
CAN	30	18	24	D	104	44	68	n	156	6E	110
EM	31	19	25	E	105	45	69	o	157	6F	111
SUB	32	1A	26	F	106	46	70	p	160	70	112
ESC	33	1B	27	G	107	47	71	q	161	71	113
ES	34	1C	28	H	110	48	72	r	162	72	114
GS	35	1D	29	I	111	49	73	s	163	73	115
RS	36	1E	30	J	112	4A	74	t	164	74	116
US	37	1F	31	K	113	4B	75	u	165	75	117
SP, space	40	20	32	L	114	4C	76	v	166	76	118
!	41	21	33	M	115	4D	77	w	167	77	119
“	42	22	34	N	116	4E	78	x	170	78	120
#	43	23	35	O	117	4F	79	y	171	79	121
$	44	24	36	P	120	50	80	z	172	7A	122
%	45	25	37	Q	121	51	81	{	173	7B	123
&	46	26	38	R	122	52	82	|	174	7C	124
‘ apostrof	47	27	39	S	123	53	83	}	175	7D	125
(	50	28	40	T	124	54	84	~	176	7E	126
)	51	29	41	U	125	55	85	<DEL>	177	7F	127
*	52	2A	42								
+	53	2B	43								

Табела окталних, хексадецималних и декадних **ASCII** кодова

	00	01	10	11
0000	NULL	DLE	DS	
0001	SOH	DC1	SOS	
0010	STX	DC2	FS	SYN
0011	ETX	TM		
0100	PF	RES	BYP	PN
0101	HT	NL	LF	RS
0110	LC	BS	ETB	UC
0111	DEL	IL	ESC	EOT
1000		CAN		
1001		EM		
1010	SMM	CC	SM	
1011	VT	CU1	CU2	CU3
1100	FF	IFS		DC4
1101	CR	IGS	ENQ	NAK
1110	SO	IRS	ACK	
1111	SI	IUS	BEL	SUB

00

	00	01	10	11
0000	blanc	&	-	
0001			/	
0010				
0011				
0100				
0101				
0110				
0111				
1000				
1001				`
1010		!	|	:
1011	.	$	,	#
1100	<		%	@
1101	(	)	-	‘
1110	+	;	>	=
1111			?	“

01

	00	01	10	11
0000				
0001	a	j	~	
0010	b	k	s	
0011	c	l	t	
0100	d	m	u	
0101	e	n	v	
0110	f	o	w	
0111	g	p	x	
1000	h	q	y	
1001	i	r	z	
1010				
1011				
1100				
1101				
1110				
1111				

10

	00	01	10	11
0000	{	}	\	0
0001	A	J		1
0010	B	K	S	2
0011	C	L	T	3
0100	D	M	U	4
0101	E	N	V	5
0110	F	O	W	6
0111	G	P	X	7
1000	H	Q	Y	8
1001	I	R	Z	9
1010				
1011				
1100				
1101				
1110				
1111				

11

Табела бинарних **EBCDIC** кодова

ПРИЛОГ - Б
ВРСТЕ РАЧУНАРСКИХ СИСТЕМА

Од првих дана развоја рачунарске технике, рачунари су се разврставали у три групе: **mainframe**, ***минирачунаре*** и ***микрорачунаре***. Међутим, нове технологије доводе до појаве нових рачунара који се могу назвати суперрачунари, који додатно компликују класификацију рачунара. Данашњи микрорачунари су базирани на примени микропроцесора чија меморијска реч може бити 8, 16, 32 или 64 бита, са оперативном меморијом чија се величина креће од неколико десетина **KB** до неколико десетина **MB** (на пример, 64 **MB**). Обзиром да се снага микропроцесора из дана у дан повећава (већ раде на 500 **MHz**), а главна меморија им се може проширити до 4 **GB**, јасно је да ће границе између појединих врста рачунара бити још мање стриктне. Њихова цена је реда од једне до неколико хиљада долара. Мини и супермини рачунари имају меморијску реч дужине најмање 32 **bita**, а магистрала података може одједном да пренесе целу реч између главне меморије и **CPU**–а. Ови рачунари имају капацитет примарне меморије од најмање 32 **Mbyte**, а могу бити основа за реализацију система који подржава од 16 до 256 радних станица (терминала или микрорачунара). Њихова цена се креће од 20.000 $ до 50.000 $.

Mainframe рачунари су најчешће велики рачунарски системи. Они имају меморију великог капацитета и подржавају рад великог броја корисника. Најгрубље речено данашњи **mainframe** рачунари пре би могли бити названи суперрачунари. То су у датом тренутку најбржи рачунари, са највећим капацитетима меморија и највећом ценом. Данашњи суперрачунари користе технике текуће траке (**pipeling**) и паралелне обраде (више процесора који раде истовремено под контролом једне централне управљачке јединице). Такође се користе и друге технике мултипроцесирања. Оно што још карактерише ове системе јесте да истовремено опслужују велики број радних станица (више стотина).

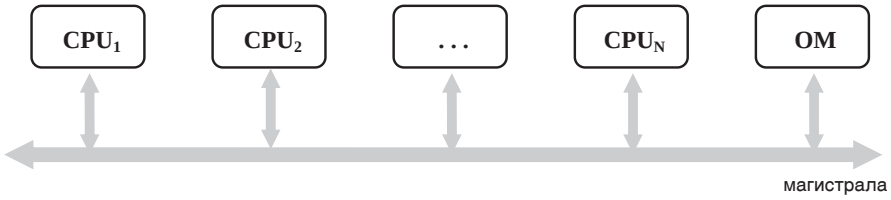
Уобичајена класификација рачунара на **mainframe**, **mini** и **micro** рачунаре данас је практично превазиђена јер се на тржишту могу наћи микрорачунари који по својој брзини и капацитету главне меморије превазилазе могућности некадашњих mainframe рачунара. Међутим, нису ово једине карактеристике које треба узимати у обзир када се говори о врстама рачунара. Уместо ове уобичајене поделе, због брзог развоја рачунарске технике, можда данас има више смисла подела према броју централних процесора у једном рачунарском систему. Рачунарски систем са једним **CPU** назива се **simplex**, са два процесора **duplex**, а системи са **n CPU**–а се зову **n-plex** системи. Системи са више од једног **CPU** називају се мултипроцесорски системи.

Б - 1. SIMPLEX рачунарски системи

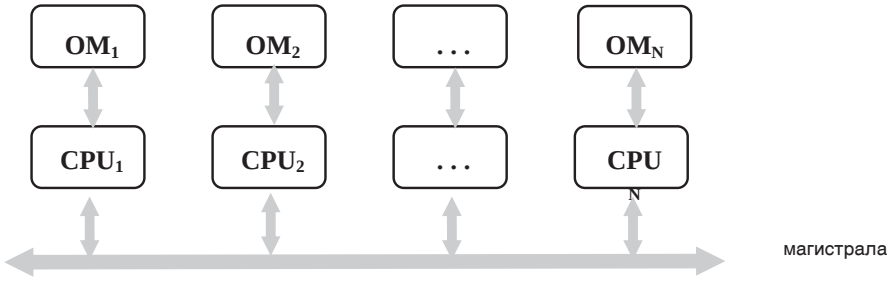
Већина досадашњих рачунарских система (од микрорачунара до **mainframe**) реализована је на бази једне **CPU**. За ове системе је, дакле, карактеристично да имају један централни процесор и једну или више меморијских јединица које су под контролом меморијске управљачке јединице (**memory controller, memory management unit**). За повезивање **CPU**–а и осталих делова система могу се користити канали или **I/O** адаптери. Међу најпознатије рачунаре ове врсте свакако спадају **IBM 360/370, VAX 11/780, PC** итд.

Б - 2. МУЛТИПРОЦЕСОРСКИ СИСТЕМИ

Постоје разне врсте мултипроцесорских система при чему су могуће разне варијанте: почев од тога да сваки од процесора ради независно, па до потпуног дуплирања њихових послова. Спрега појединих **CPU** остварује се преко заједничких компонената система, а то је најчешће главна меморија. Ако сви процесори у систему користе једну (заједничку) главну меморију онда су то чврсто (круто) повезани системи (**tightly coupled systems**), слика Б–1. Ако сваки **CPU** има своју засебну главну меморију онда су то лабаво спрегнути системи (**loosely coupled systems**), слика Б–2. У овим системима процесори непосредно комуницирају међусобно.



Слика Б–1. Круто спрегнути системи

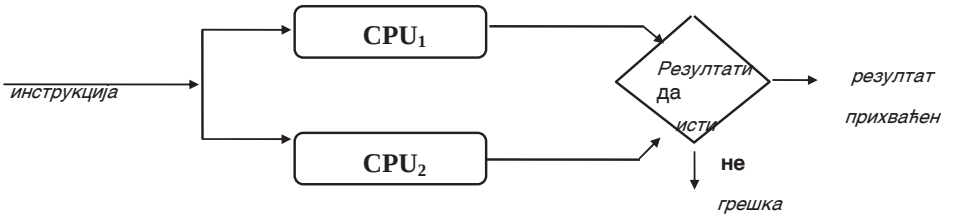


Слика Б–2. Лабаво спрегнути системи

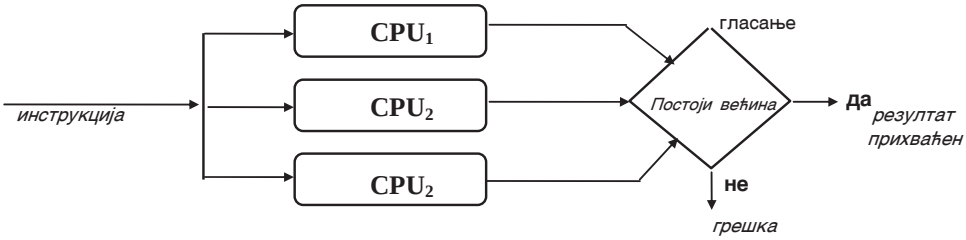
Посебну врсту вишепроцесорских система чине, такозвани, системи **имуни на грешку** (**fault tolerant**). Ови системи не само да користе више процесора, већ дуплирају (редундују) и друге компоненте система да би се добио поуздан систем који може да ради чак и у случају отказа неких компоненти. Ово омогућава

откривање и отклањање грешке или отказа појединих делова система. Код ових рачунарских система сви процесори (или бар два) извршавају исте операције. Ако су резултати свих процесора исти, обрада се наставља. Ако се резултати разликују, дијагностикује се грешка (или отказ) и предузимају се одређене активности за корекцију.

Најчешће се у пракси сусрећу системи са два независна процесора (**tandem**), слика Б–3, и три процесора – принцип већинског гласања (**voting**), слика Б–4. Код система са два процесора (**tandem**) грешка се детектује у случају када резултати оба процесора нису исти. Али неки **tandem** системи, осим два процесора, имају и две копије програма (за сваки процесор посебно). Један од процесора је главни, а други је резервни. Када се у главном систему открије грешка помоћни систем преузима контролу. Ако је детектован отказ неке хардверске компоненте, систем бира алтернативни пут или јединице да би се продужио поступак обраде до отклањања грешке. Код система са три процесора, систем упоређује три резултата, ако су сви исти, или се само један разликује, узима се као тачан резултат већине процесора, тј. процесори "гласају" и побеђује резултат већине.



Слика Б–3. Тандем



Слика Б–4. Принцип већинског гласања

Б - 2.1. ПОМОЋНИ ПРОЦЕСОРИ

У првим рачунарским системима **CPU** је био задужен за извршавање свих операција у рачунару. Да би се повећала брзина обраде података, неке специјализоване операције су поверене на извршавање посебним процесорима. Тако су улазно–излазне операције поверене каналима (**I/O processors, peripheral proces-**

sors), обрада низова података је поверена векторским процесорима (**array processors**), а обрада база података такозваним **database** машинама. Ови процесори се називају помоћни или процесори за подршку (**support processors**, **attached support processors**). На овај начин централни процесор је знатно растерећен, па је самим тим повећана ефикасност обраде. Ови помоћни процесори класификују се као **front-end** и **back-end** процесори. **Front-end** процесори су смештени између корисника и **CPU**. У ову групу спадају канали и комуникациони процесори. **Back-end** процесори извршавају специјалне послове (уместо **CPU**–а) који не захтевају директну интеракцију са корисником. У ову групу спадају векторски процесори и процесори за обраду база података.

Б - 2.1.1. ВЕКТОРСКИ ПРОЦЕСОРИ

Векторски процесори обично имају архитектуру текуће траке, посебно пројектовану за брзо извршавање аритметичких операција које су издељене на велики број сегмената (на пример, сабирања и операција померања) који се морају извршавати секвенцијално. Неки од ових процесора могу одједном обрађивати читаве матрице.

Б - 2.1.2. РАЧУНАРИ ЗА ОБРАДУ БАЗА ПОДАТАКА

За рад са базама података (за пословну обраду) развијене су посебне врсте процесора за подршку. Ови процесори су посебно конструисани за ненумеричке обраде и извршавају дугачке и компликоване операције као што су сортирање и претраживање. На овај начин се растерећује централни процесор који у то време решава неке друге проблеме. Они у принципу користе паралелну обраду за реализацију и приступ базама података.

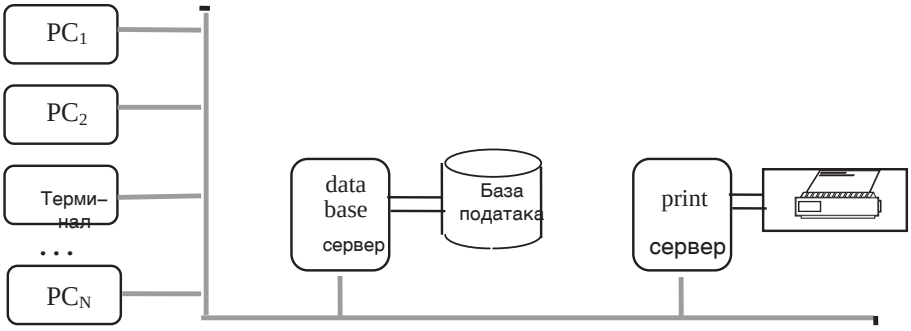
Б - 3. ДИСТРИБУИРАНА ОБРАДА

Системи са дистрибуираном обрадом (**distributed processing**) представљају лабаво спрегнуте групе процесора и периферијских јединица које су повезане разним техникама. Омогућавају значајно повећање брзине обраде јер више процесора истовремено ради на истом проблему. Ови системи додатно користе предности мултипроцесорских система као што су: поузданост и имуност на отказ и грешку (попут оних у **fault tolerant** системима). Они такође омогућавају заједничко коришћење скупих или критичних ресурса. На пример, на слици Б–5. више корисника (PC_1, PC_2, . . ., PC_n) има приступ једном штампачу или једној бази података.

Б - 3.1. ОСТВАРИВАЊЕ ВЕЗЕ

Код система са дистрибуираном обрадом повезују се терминали, микрорачунари, велики рачунари и периферијске јединице које могу бити у разним просторијама неке зграде или у разним зградама, градовима или државама, па чак и континентима. Ове везе се остварују посредством разних медијума, као што су: а) упредене парице (**twisted pair**) уз употребу модема и телефонских линија (**phone**

line), б) коаксијални кабл (**coaxial cable**), ц) оптички кабл (**optical fiber**), микроталасне (**microwaves**), радио (**radio**) и сателитске везе (**satellite**), као што је приказано на слици Б–6. Упредене парице, коаксијални и оптички кабл се у литератури називају **hardwire**, док се медијуми као што су ваздух или вакуум (етар) називају **softwire**.



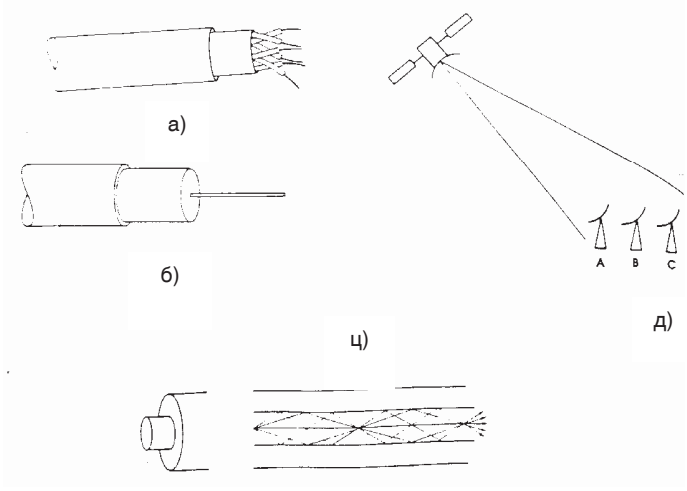
Слика Б–5. Више корисника дели један штампач и базу података

Упредене парице, (**twisted pair**) су јефтине и лаке за инсталацију, па су због тога врло често примењиване, али су врло осетљиве на спољне сметње ако нису оклопљене. То су такозвани **UTP** каблови (*неоклопљене упредене парице,* **Unshielded Twisted Pair**). Знатно имуније на сметње су *оклопљене упредене парице*, такозвани **STP** (**Shielded Twisted Pair**). То су две изоловане бакарне жице упредене равномерним кораком упредања. Ово упредање смањује индуковане електромагнетне сметње "преслушавање" између суседних парова. Користе се за брзине преноса до 100 **Mbps** (**UTP** категорија 5), јер им се пропусни опсег креће око 250 **kHz**. Представљају основу класичног телефонског система (али се на растојању од 2 до 10 **km** морају налазити појачавачи), а данас најчешће чине основу локалних мрежа унутар једне зграде где **UTP** сегменти морају бити краћи од 100 **m**.

Коаксијални кабл се најчешће користи за локалне рачунарске мреже јер је врло имун на сметње. Наиме, један бакарни проводник је направљен у облику цеви унутар које се налази проводна жица (оклопљена је и заштићена од сметњи). Пропусни опсег им је око 350 **MHz**, па самим тим омогућава реализацију брзих мрежа (500 **Mbps**), али су број чворова на једном сегменту и удаљеност врло ограничени. Да би се неутралисало слабљење, растојање између појачавача може бити од 1 до 10 **km**. Дебели коаксијални кабл (**thick coax**) ограничава дужину сегмента на 500**m** са највише 100 чворова. Најкраће растојање између суседних чворова мора бити веће од 2,5**m**, а растојање чвора од магистрале мора бити краће од 50**m**. Рачунар се на магистралу прикључује помоћу примопредајника (**transiver**). Танки коаксијални кабл (**thin coax**) је јефтинији и лакши за инсталацију. Подржава до 30 чворова по сегменту који не сме бити дужи од 185**m**. Растојање између два чвора је најмање 0,5**m**. На крајевима кабла–магистрале

налазе се завршни отпорници од 50Ω (терминатор). Рачунар се на магистралу прикључује директно помоћу **BNC–T** конектора.

Оптички кабл је медијум будућности јер представља квалитативно и технолошки виши ниво од претходних. Прави се у облику флексибилних стаклених или пластичних влакана мале масе и димензија (50 до 100 **μm**), дуж којих се преноси светлосни сигнал. Око влакна се налази омотач који потпуно рефлектује светлост (тотална рефлексија). Дакле, на месту предајника електрични сигнал се конвертује у светлост која се на месту пријема опет конвертује у електрични сигнал. Имају велики пропусни опсег (преко 3**GHz**), па самим тим и велику брзину преноса података (1 **Gbps**), изузетно су безбедни у односу на сметње (нарочито електричне) и имају изузетно мало слабљење (размак између појачавача може бити од 10 до 100 **km**). Ови оптички комуникациони системи ће у будућности потиснути у великој мери остале медијуме управо због својих предности. За сада су скупи и захтевају додатну опрему за прикључивање радних станица, па то ограничава њихову примену.



Слика Б–6. Комуникациони медијуми

Микроталасне комуникације се користе углавном за пренос података од тачке до тачке (користе параболичне "тањире" са врло усмереним дијаграмима зрачења). Погодне су за пренос између зграда или за удаљене комуникације. У поређењу са кабловима осетљивије су на ометање и не гарантују безбедност порука јер је преносни медијум етар (**ether**), али су у том погледу знатно боље од радио–комуникација.

Када се за успостављање микроталасне везе између две станице на земљи, као микроталасна станица, користи један или више комуникационих сателита онда кажемо да се ради о сателитској мрежи. Сателит прими аналогни сигнал на једној учестаности, појача га, обнови дигитални сигнал (**repeater**) и утисне га у

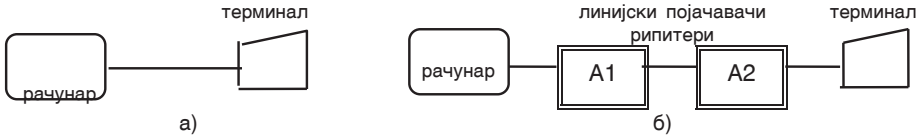
аналогни сигнал на другој фреквенцији. Обично се користи у опсегу од 1 до 10 **GHz**. За сада се углавном користи за телефонски, телекс и телевизијски сао–браћај, али од недавно се користи и у **Internet** рачунарској мрежи.

Радио комуникације се од микроталасних разликују по томе што радио антена зрачи у свим правцима, а покривају опсег фреквенција од 30 **MHz** до 1 **GHz**. У свету данас постоји велики број пакетских радио мрежа у које је укључено и не–колико сателита. Као и претходне две мреже и ова се базира на **X.25** прото–колу. Тачније пакетски радио се базира на примени **AX.25** протокола.

Већина великих мрежа комбинује два или више медијума, тј. састоји се из мањих мрежа које се базирају на различитим медијумима. Но како се целокупни рад мреже одвија под контролом одређених протокола (стандарда), нема никаквих проблема за њихово муђусобно повезивање.

Б - 3.1.1. НЕПОСРЕДНО ПОВЕЗИВАЊЕ

При реализацији LAN мрежа могу се користити неколико сигнализационих техника. Која од њих ће бити употребљена зависи од околности и пројектних захтева у погледу брзине, врсте употребљених каблова, цене хардверских компо–ненти итд. Делови рачунарског система са дистрибуираном обрадом могу се повезати и непосредно, што се посебно односи на повезивање терминала (мик–рорачунара) и рачунара (слика Б–7. а). То се најчешће ради посредством RS 232 стандарда (по **EIA Electronic Industries Associattion**) или по **V.24** стандарду (**CCITT**, **Consultative Committee of International Telephone and Telegraph**). Ови начини повезивања су ограничени на растојања до **15m**, јер квалитет сигнала брзо опада са растојањем. Ово ограничење се може превазићи додавањем линиј–ских појачавача (**line drivers**) који појачавају сигнал, слика Б–7. б). На овај начин компоненте система могу бити удаљене више стотина метара.



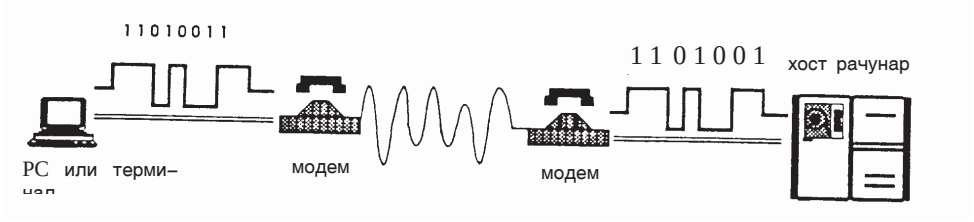
Слика Б–7. Повезивање блиских и удаљених рачунара

Структура сигнала је таква да је бит података чија је вредност логичка нула (**0**) представљен позитивним напоном од 12 волти (тј. неким напоном у опсегу од 5 до 15 волти), а бит података чија је вредност логичка јединица (**1**) представљен је напоном –12 волти (тј. неким напоном у опсегу од –5 до –15 волти).

Највећа брзина промена напонских нивоа сигнала, тј. напонских нивоа, уједно је и највећа могућа брзина преноса података и назива се ***бод*** (**baud rate**). Ако је брзина промене сигнала 300 **baud** онда је брзина података 300 **bps** јер свака промена сигнала представља један **bit** података.

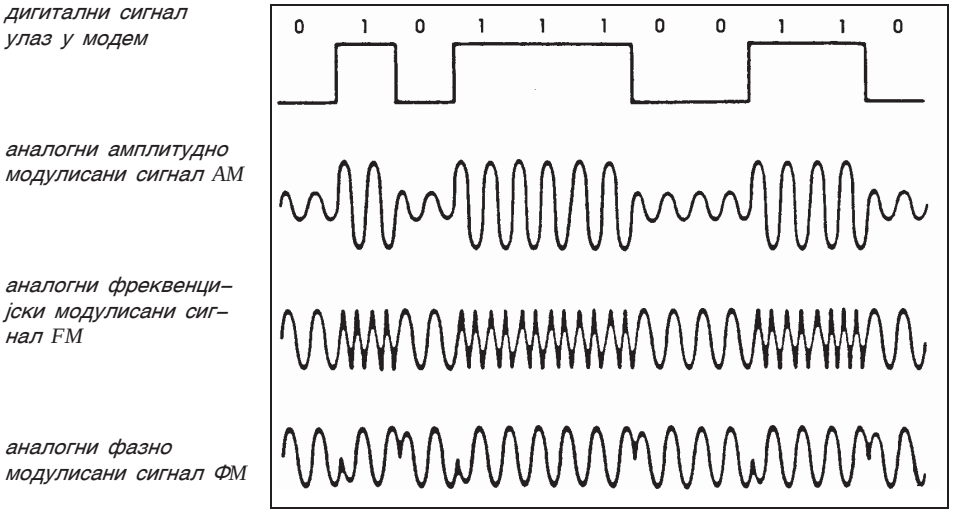
Б - 3.1.2. МОДЕМИ

Телефонске линије су биле, јесу и биће најчешћи медијум за повезивање рачунара. С једне стране, већина телефонских линија преноси аналогне или непрекидне сигнале (говор). С друге стране подаци у рачунарима су дигитални, дискретни битови. Пре него се дигитални подаци пренесу преко стандардне телефонске линије морају бити конвертовани у аналогне сигнале. Тај поступак се обавља модулацијом. Примљени подаци на другој страни (да би их прихватио рачунар) морају се претходно поново вратити у дигитални облик поступком који се зове демодулација. Уређај који то ради зове се **modem** (**modulator–demodulator**). На слици Б–8. приказан је поступак конверзије сигнала и њихов пренос преко телефонске линије.



Слика Б–8. Остваривање везе помоћу телефонске линије

Сам процес претварања дигиталног сигнала у аналогни, може се извршити на више разних начина: променом амплитуде–**амплитудна модулација AM** (**amplitude shift keying**, **ASK**), променом фреквенције–**фреквентна модулација FM** (**frequency shift keying**, **FSK**) или променом фазе наизменичног сигнала–**фазна модулација ФМ** (**phase shift keying**, **PSK**), као на слици Б–9.



ССлика Б–9. Врсте модулација

Амплитудна модулација (**AM, ASK**) се састоји у томе да се амплитуда носећег сигнала мења у ритму промена дигиталног податка. Логичка јединица (1) се преноси једним (обично вишим) нивоом амплитуде наизменичног носећег сигнала, а логичка нула (0) другом (нижом) амплитудом, при чему се учестаност и фаза наизменичног сигнала не мењају.

Фреквентна модулација (**FM, FSK**) се састоји у томе да се логичка јединица преноси помоћу носећег наизменичног сигнала константне амплитуде и фазе чија је фреквенција f_1, а логичка нула помоћу сигнала чија је фреквенција f_0. Обично је $f_1 > f_0$.

Фазна модулација (**ФМ, PSK**) се састоји у томе да носећи сигнал има константну амплитуду и фреквенцију, али се сваки пут при преласку са нуле на јединицу и обрнуто, фаза наизменичног сигнала мења за 180^0. Чешће се користе технике у којима се кодирају по два бита, па комбинацији 00 одговара фаза 0^0, комбинацији 01 фаза од 90^0, комбинацији 10 фаза од 180^0, комбинацији 11 фаза од 270^0,

Али врло брзо су се појавила два нова стандарда X.21 и X.21 bis. X.21 се односи на пренос дигиталних сигнала преко дигиталних телефонских система који су данас готово комплетно заменили аналогне системе. Ово омогућава већи степен функционалности него стандард V.24/RS232C. Стандард X.21 bis само премошћава празнину која настаје између стандарда X.21 и V.24 и обезбеђује повезивање V.24 модема на дигитална кола. Коришћењем X.21 повезивања, уређај за пренос података (DTE) генерише дигиталну информацију која се преноси кроз дигиталну телефонску мрежу све до одредишта. Капацитет је, у смислу максималне брзине преноса података у преносном медијуму, ограничен његовим пропусним опсегом. Ове брзине су ограничене могућношћу тачног препознавања садржаја сигнала на неком удаљеном крају.

Одабирање

Велики број техника преноса се заснива на идеји да се правоугаони таласни облици могу представити помоћу синусних таласних облика исте фреквенције. Треба уочити да се узимањем одбирака тачно на средини сваког временског интервала (периоде) увек добија добра презентација жељене вредности. Веће сигналне брзине повећавају фреквенцију ових таласа. Ако тачка одабирања предњачи или касни у односу на средишњу, очитана вредност је мање тачна, и под одређеним условима може бити чак и погрешно протумачена.

Наравно, стварни правоугаони таласни облик биће представљен као комбинација основног синусног таласа и још неких његових виших хармоника (талас чија је учестаност цео број пута већа од основне). Јасно да што је више хармоника садржано у таласном облику утолико ће интерпретација и апроксимација правоугаоног таласног облика бити тачнији. Тачка одабирања може тада више одступати од средишње, а да интерпретација очитане вредности буде тачна. У стварности, потребно је да граничне фреквенције кабла и међусклопа буду бар пет пута веће од сигналне брзине. Скуп фреквенција које могу бити успешно пренете

преко хардвера (кабла) чине пропусни опсег. Ако сигнал заузима цео пропусни опсег хардвера онда је то пренос у основном опсегу. Разни трансмисиони медијуми имају различите граничне фреквенције и самим тим могу обезбедити различите брзине преноса. Наравно, што је виша брзина сигнала то је временски интервал у коме се може вршити одабирање краћи, па је самим тим добро одабирање теже остварити.

Оквири

Најважнија ствар која мора бити остварена при преносу сигнала јесте да се на пријемном месту може техником одабирања добити вредност која ће бити тачно интерпретирана. Наравно, битно је и да пријемник може да организује долазећи сигнал, да га прими као низ битова и да препозна знак или информацију коју тај сигнал садржи. Када говоримо о преносу карактера најчешће се користе две технике уоквиривања–преноса (**framing**):

- **Асинхрони пренос** без заједничког такта између крајњих тачака. Карактери се шаљу у било ком тренутку времена.
- **Синхрони пренос** са заједничким тактом, па пријемна тачка "зна" када треба да врши одабирање.

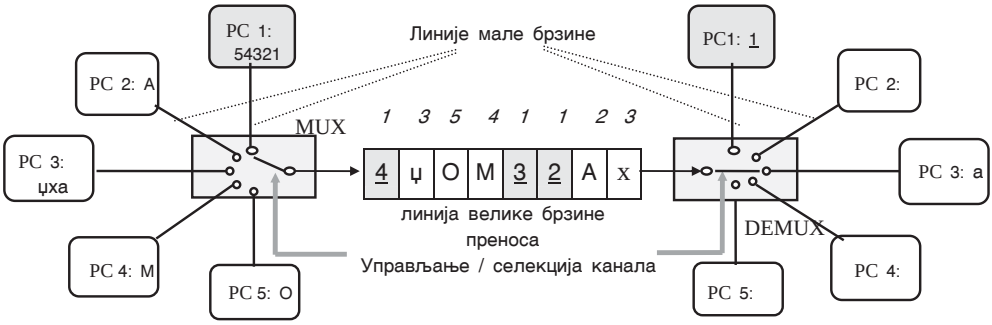
Асинхрони пренос типично ради на малим брзинама (на пример, 300 до 2400 **bps**), мада су могуће и веће брзине. Пријемни крај се припрема за долазак карактера помоћу сигнала који се шаље на почетку пре самог знака који се шаље. Овај сигнал је познат под именом ***старт бит*** (**start bit**).

За веће брзине преноса као што захтевају везе између два рачунара (веће од 9600 **bps**), оба краја морају бити синхронизована да би се обезбедило да њихови тактови буду потпуно једновремени (што ефективно казује шаљи податке сада или одабирај податке сада, зависно о ком крају је реч). Подаци се шаљу непрекидно са повременим (периодичним) слањем синхронизационих карактера, **SYN**. У локалним мрежама где су брзине знатно веће од оних у традиционалним комуникацијама помоћу ПТТ веза, проблем одржавања синхронизације пријемног чвора или станице са предајном станицом знатно је тежи. То је последица тога што веће сигналне брзине захтевају већу брзину такт сигнала. Синхронизација између предајника и пријемника је од тако великог значаја да не би дошло до губитка информација. Рецимо, ако предајник шаље сигнал два пута у секунди а пријемник врши одабирање само једном у секунди, онда ће половина података бити изгубљена. Али значајно је и да и предајник и пријемник раде у исто време–синхронизовано, јер ако пријемник дозволи да се тренутак одабирања помера (**drift**) у односу на средишњу тачку може доћи до погрешног очитавања долазећег низа карактера.

Б - 3.1.3. КОНЦЕНТРАЦИЈА ПОДАТАКА

Непосредно повезивање сваке периферије са рачунаром је скупо и неефикасно. То је последица чињенице да терминали генеришу релативно ретко и мало по–

датака у поређењу са уобичајеним брзинама обраде података. То је прескупо, па се решење може наћи у томе да се више уређаја повеже на исту комуникациону линију. Мултиплексери су уређаји који омогућавају да се сигнали из више разних извора преносе посредством једне комуникационе линије. Две основне технике које се користе при мултиплексирању података: фреквенцијски мултиплекс (**FDM**, **Frequency Division Multiplexing**) и временски мултиплекс (**TDM**, **Time Division Multiplexing**). Временско мултиплексирање дели време преноса података у интервале који се зову **slot** и додељује сваки интервал једној јединици која тада преноси своје податке, слика Б–10. Видимо да је један део поруке са PC 1 већ стигао (број 1), а да остали делови пристижу преко линије велике брзине (осенчена поља). На линију управо ступа податак 4 са PC1, док на пријемни демултиплексер управо стиже податак **x** са PC3.

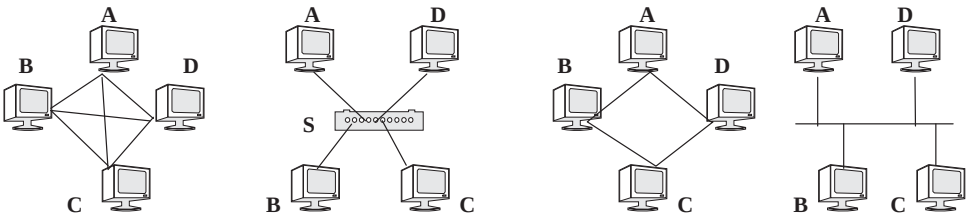


Слика Б–10. Порука са PC1 се шаље преко брзе линије (осенчена поља)

Фреквенцијски мултиплекс је повезан са појмом ширине пропусног опсега линије преноса (**band width**). Пропусни опсег се односи на опсег фреквенција које могу бити пренете посредством неког медијума. Фреквенцијски мултиплекс дели расположиви пропусни опсег медијума на одређени број потканала: фреквентних опсега (**frequency ranges**). Сваки потканал се додељује једном уређају, и при томе сви подканали истовремено преносе своје податке. Између два подканала налазе се опсези фреквенција које се не користе (**guard bands**), који помажу да се на пријему изврши раздвајање података. Рачунарске мреже које користе само део пропусног опсега кабла могу обезбедити вишеканални пренос, познате су под именом широкопојасне, а сваки опсег фреквенција у мултиплексираном систему обезбеђује један канал. Ако нису сви канали искоришћени онда нису у потпуности искоришћене могућности трансмисионог медијума. Наравно има више разних приступа за представљање сигнала из основног опсега у широкопојасним системима. Али треба имати у виду да је у већини случајева сигнал у основном опсегу дигиталан а у широкопојасном је аналоган. Постоје такође и системи познати као системи са носиоцем који су негде између ова два система иако су у суштини аналогни али заузимају цео расположиви пропусни опсег (јефтинија верзија широкопојасног).

Б - 3.2. ТОПОЛОГИЈА ДИСТРИБУИРАНИХ СИСТЕМА

Дистрибуирани систем је мрежа састављена од процесора и периферијских јединица. Сваки уређај у мрежи зове се чвор (**node**). Сви чворови једне мреже могу бити лоцирани на релативно ограниченом простору и имају једну те исту организацију. У том случају мрежа је локална и назива се **LAN** (**Local Area Network**), мрежа локалног подручја. Исто тако мрежа може бити распрострањена на знатно ширем простору и може имати више разних организација и користити више државних или приватних комуникационих система. Оваква мрежа се зове **WAN** (**Wide Area Network**), мрежа пространог подручја. Начин на који су уређаји који чине једну мрежу међу собом повезани назива се топологија (**topology**). Неке од могућих топологија дате су на слици Б–11. Термин топологија најчешће се користи за опис конфигурације веза између радних станица. То су пре свега магистрала (**bus**), прстен (**ring**), звезда (**star**), делимично повезане (**mash**, **partiali connected**), потпуно повезане (**fully connected**) и хијерархијске (**hierarchical**). Свака од ових топологија може се користити у реализацији локалне мреже са различитим степеном успеха јер свака од њих има неке предности и мане. То је пре свега последица различитих ограничења или физичких могућности система који се инсталира, или окружења у коме систем ради.



Слика Б–11. Могуће топологије рачунарских мрежа

Делимично или потпуно повезана топологија

Код потпуно повезаних топологија свака радна станица или уређај повезан је засебним каблом са сваком другом станицом (уређајем). Делимично повезани системи имају станице које су истовремено повезане са више од две друге станице или уређаја. Повезивање чворова *од тачке до тачке* (**point to point**), или како се још зове *потпуно повезани систем* (**fully connected system**) је врло поуздан систем, јер има много разних путева да се подаци пренесу с једног места на друго (слика Б–11. а). Ако се, на пример, директна веза од **A** до **B** прекине, порука може бити послата преко чвора **C** или **D: A-C-B** или **A-D-B** или **A-C-D-B**, и слично. Предности ових топологија су: могућност избора више разних рута (путања) и робусност и могућност опстанка и у случају прекида каблова. Мане ових топологија су: потребно је знатно више каблова и додавање више станица доводи до изузетног поскупљења.

Звезда

Топологија звезде (**star**), захтева да све поруке пролазе кроз централни прекидачки склоп, комутатор, преусмерач (**switcher**), слика Б–11. б). У овој топологији сваки уређај је повезан са централном комутационом станицом. Дакле, нема дељења медијума и свака нова станица која се додаје повезује се засебним каблом са централном тачком. Ако се обезбеди робусна централна тачка у којој се подаци преусмеравају ка одредишту, ова топологија даје знатно већу поузданост од магистрале и прстена у односу на проблеме са кабловима. Обично, брзина преноса података није сувише велика. Посебна погодност је могућност коришћења постојећих телефонских каблова за реализацију рачунарске мреже, што знатно може смањити трошкове инсталирања, а улогу комутатора има централа. Ова конфигурација омогућава комуникацију између било која два чвора једноставним слањем поруке прекидачком склопу **S**. Лако се додаје нови чвор, али отказ прекидачког склопа значи и отказ целог система. Такође постоји опасност од загушења прекидачког склопа, што опет доводи до пада целог система.

Предности употребе топологије звезде су: оштећење кабла има смањени утицај, могућност употребе постојећих каблова и лака детекција грешке и њено изоловање. Мане топологије звезде су: централна тачка која може оборити читаву мрежу и мање брзине преноса.

Прстен

У *конфигурацији прстена* (**ring network**), сваки чвор је повезан са два друга чвора (слика Б–11. ц). Подаци се увек крећу у једном истом смеру (на пример, у смеру казаљке на сату **A-B-C-D-A**) да не би дошло до интерференције. Сваки уређај прикључен на прстен, мора бити способан да преузме поруку која је њему послата и да даље проследи оне поруке које су послате неком другом чвору. Неке мреже су направљене помоћу два комплета каблова да би се омогућило порукама да буду послате и у супротном смеру када је чвор или веза у квару. Али, уочимо да порука коју чвор **A** шаље чвору **D** мора пре тога да прође кроз чворове **B** и **C**, тј. само суседни чворови су директно повезани.

Сви чворови (радне станице и међусклопови) су повезани у круг, и сваки од њих има способност да регенерише сигнал. На овај начин се осигурава квалитет сигнала у свакој тачки прстена, а критични фактор је растојање између два чвора. Додатно на перформансе мреже утиче и број чворова који имају активну улогу у мрежи, јер сваки чвор уноси и одређено кашњење. То је последица чињенице да сваки чвор има улогу рипитера и регенерише низ битова. Док прихвата наредни (долазећи) бит и препознаје га као нулу или јединицу, чвор шаље претходни бит. Дакле, препознавање и прослеђивање поруке уноси најмање један бит кашњења. Што је више активних чворова у прстену утолико је веће кашњење при преносу података, јер сваки активни чвор уноси додатно кашњење. Дакле перформансе зависе од броја активних чворова. Што је можда још важније, повећање броја прикључених уређаја повећава вероватноћу да ће у неком тренутку времена нека станица вршити пренос података. То значи да ће све остале станице морати да сачекају да престане текући саобраћај па тек онда оне могу да дођу на ред.

Приступ било које станице на прстен обично се контролише помоћу **жетона** (**token**) или **прореза** (**slot**), и ове технике гарантују перформансе чак и у условима врло великог саобраћаја. Структура прстена има могућност да тренутно открије било који квар у кабловској структури или у преношеним подацима. Јасно, овде нема проблема рутирања. Свако на прстену узима податке, али само адресирани чвор може да копира податке. Веће мреже могу бити направљене као група повезаних прстенова. Наравно, прстенови се повезују преко уређаја за повезивање, на пример, мостова.

Предности мрежа са топологијом прстена су: трансмисиони капацитет је равномерно расподељен међу повезаним станицама, мала могућност грешке, приступ је загарантован чак и у условима великог оптерећења и једноставно рутирање у сваком појединачном прстену

Мане мрежа са структуром прстена су: теже додавање нове станице у чистом прстену него у другим топологијама. Ово је нешто олакшано у неким системима са другачијим кабловским структурама, потребан је рипитер у сваком чвору, тј. интерфејсу за станицу и неисправност у једном међусклопу изазива прекид рада целог прстена.

Магистрала

Конфигурација магистрале (**BUS**) слика Б–11. д) садржи једну комуникациону линију коју деле сви повезани уређаји, а сви чворови могу директно да комуницирају међу собом. Свака од повезаних јединица приступа медијуму преко мрежног међусклопа (**interface**) који представља тачку у мрежи која има одређену хардверску адресу. Подаци (поруке, **message**) се преносе између уређаја тако што идентификују жељено одредиште преко његове хардверске адресе. Да би се осигурало да баш жељени одредишни уређај преузме поруку, а не неки други, свака станица или међусклоп морају бити на јединствен начин одређени. Дакле, хардверске адресе морају бити јединствене унутар сваке мреже која има исто управљање (администрирање).

Највећа дозвољена дужина кабла зависи од од више чинилаца:

- врста сигнала (дигитални, аналогни)
- метод приступа (како се приступа дељеном, заједничком трансмисионом систему)
- брзина сигнализације (брзина преноса података)
- употребљени кабл (физичке могућности)

Како ови параметри утичу на систем може се показати на примеру једне просте особине. Највећа дужина кабла одређује колико се сигнал може изобличити а да се и даље може сигурно препознати у свакој тачки медијума. У противном ће се десити грешка при читању сигнала. Ово је строго ограничено и сваки од поменутих фактора даје свој удео у томе. На пример, ако се у етернет мрежи користи танки коаксијални кабл највећа дужина једног сегмента може бити 185 m, а код дебелог коаксијалног кабла је 500 m. Наравно, ова полазна ограничења могу бити превазиђена употребом уређаја за просту регенерацију облика, тј.

појачавача (**amplifier**) и рипитера (**repeater**). Ови уређаји повремено враћају сигналу почетни облик.

Рипитери се користе у дигиталним мрежама, а код система са аналогним сигналима користе се појачавачи. Проширење могућности мреже повећањем максималне дужине сегмента кабла довело је до идеје о прављењу мрежа са више сегмената (**multiple segment topology**), што даље доводи до појаве топологије стабла, тј. хијерархијске топологије (**tree, hierarchical topology**).

Већина мрежа са топологијом магистрале ради на пасивној основи, тако што свака од повезаних станица "чује" када податак пролази поред ње, али не игра никакву улогу у преносу података. Наравно, први део сваког низа података носи неки облик адресе одредишта и када станица препозна своју адресу она узима (снима) податке са магистрале и пропушта их даље ка осталим уређајима.

Порука се може упутити и већем броју (свим станицама) тако што су све препознале своју адресу што је подржано специјалним форматима адреса. Приликом пријема неке поруке од стране станице или интерфејса подаци прелазе у тај уређај. Дакле, тај уређај мора да прекине претходни процес да би записао или обрадио долазеће податке. То значи да ће све станице које прихватају податке са магистрале бити повремено неупотребљиве за корисника, тј. неће одговарати на његове команде или ће прекинути на неко време локалне активности. Ово понекад може бити незгодно за корисника.

Предности коришћења топологије магистрале су: једноставност реализације и мала цена потребних уређаја, лако повезивање и додавање нових уређаја (и радних станица), лоцирање прекида кабла је релативно лако (лако одржавање), систем идеалан за примену код саобраћаја типа "један према више" и систем погодан за примену код саобраћаја који се скоковито мења (неравномеран).

Мане топологије магистрале су: нема аутоматске потврде пријема као својства мреже (мада протоколи на вишим слојевима могу осигурати успешан пријем), потенцијално смањена безбедност јер свако може да слуша поруку и међусклопови морају да поседују "интелигенцију".

Логички изглед мреже

У опису топологије мора се обратити пажња на разлику између физичког и логичког положаја каблова. Оно што личи на звезду може стварно бити прстен и обрнуто. То је нарочито изражено при имплементацији прстена са жетоном који је реализован уз помоћ јединице за приступ више станица (**Multistation Access Unit, MAU**) која служи за повезивање сваке појединачне станице на прстен. Она је такође лоцирана у центар система и мрежа има исти физички изглед као звезда. Повезивање уређаја у прстен своди се на просто убацивање у слободан прикључак у **MAU**. Показује се да је структура каблова врло важна посебно када је реч о одржавању, а добра структура омогућава бољу еластичност и даје већи ниво управљања мрежом.

Б - 3.3. УПРАВЉАЊЕ ДИСТРИБУИРАНИМ СИСТЕМИМА

Једна мрежа садржи у себи различите уређаје од којих већина има неке своје особености у погледу комуникација. Исто тако више различитих локалних мрежа може бити међу собом повезано у једну **LAN** или **WAN** мрежу. Врло често се у мрежи јавља потреба за заменом старог уређаја новим (и другачијим), али то свакако не би требало да захтева измене у апликативном софтверу. На крају највећи број корисника мреже не мора да зна ништа о мрежама и рачунарима, нити како се проналазе подаци који су њима потребни.

Комуникација између великог броја различитих уређаја може да изазове озбиљне проблеме јер различити уређаји нужно постављају различите захтеве. Комуникациони софтвер је задужен да изађе на крај са овим разликама. Искуство је показало да је најбољи приступ третирање овог проблема као низа потпроблема. Сваки од ових мањих проблема решава се на различитом нивоу хијерархије. Овакав приступ доводи до потребе да се рачунарски систем моделира на одређени начин.

Б - 3.3.1. СТАНДАРДИ

Б - 3.3.1.1. Идентификација проблема

Поред медијума, сигналних и прекидачких техника постоји још читав низ проблема који се тичу успостављања и одржавања дијалога између два рачунара. На пример, како да обезбедимо да структура јединице података која је послата буде препозната на пријемном крају? Да ли су презентације података исте на оба краја (ASCII/EBCDIC)? Да ли постоје методе за опоравак података код дугачких порука у случају грешке? Да ли треба да користимо мере безбедности? Које услуге треба да обезбедимо између нека два рачунара домаћина (**host**)? Да ли постоји исти "**с краја на крај**" протокол на оба рачунара домаћина који би омогућио опоравак података на исти начин, или да ли се користе исте технике адресирања?

Наравно ово није коначна листа, али је довољна да укаже на бројност проблема које треба решити пре него што се обезбеди оптимално окружење за кориснике. Закључак који се намеће је да је идеално оно окружење у којем корисник може да изабере да разговара са било којом јединком широм света без обзира на њену опрему, да лоцира било коју јединку када је то најпогодније и да може да размени потребне информације са било ким без икаквог знања програмирања.

Са тачке гледишта корисника веза која постоји између уређаја A и B није од значаја. То може једноставно бити кабл, локална мрежа (LAN), мрежа широке распрострањености (WAN) или њихова комбинација. Оно што треба да одредимо, ефективно, јесу захтеви који специфицирају како да се подаци приспели са тачке A испоруче у тачку B и обратно. Ова спецификација чини суштину коју треба да уградимо у комуникациону мрежу. Ову проблематику треба размотрити шире од

захтева који се односе само на мреже локалне распрострањености (LAN) јер треба обезбедити средства за њихово повезивање са мрежама широке распрострањености (WAN).

Б – 3.3.1.2. **OSI референтни модел**

Са практичне и пројектантске тачке гледишта корисно је поделити велики проблем на више мањих, чији су захтеви боље постављени. Укупни проблем се онда решава склапањем појединачних мањих компоненти. Дељењем проблема на мање делове омогућено је да се мањи проблеми боље проуче, разумеју и да се пројектују боља решења у инжењерском смислу. Ово је у сагласности са правилима доброг програмирања где се велики софтверски систем дели на мање поступке (процедуре), које могу бити добро истестиране. Процедуре се онда повезују, тако да заједно представљају решење сложеног проблема.

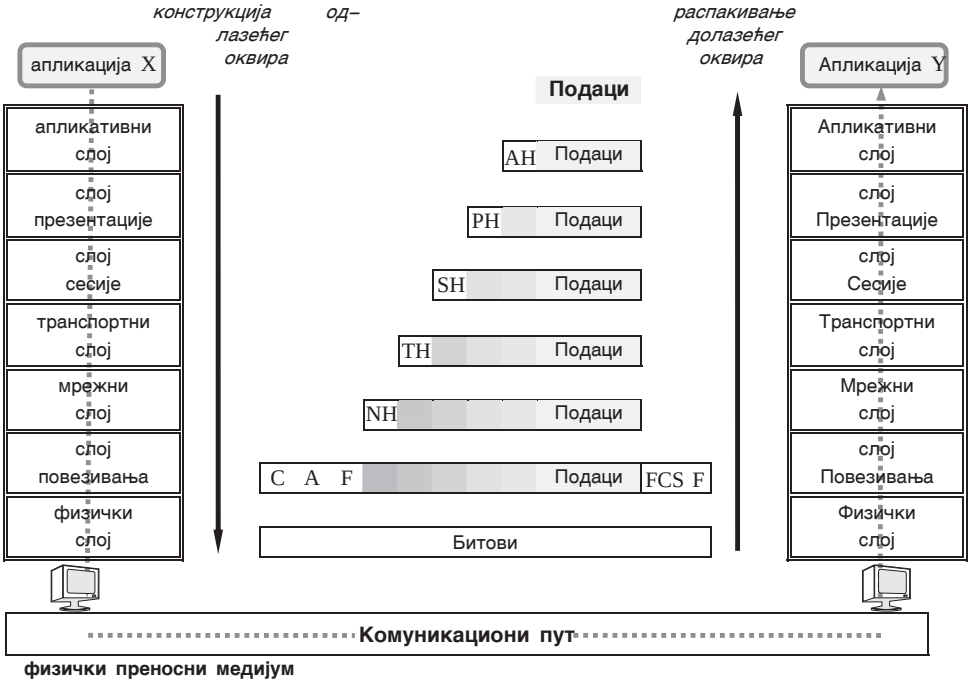
Пројектанти мрежа треба да повежу много различитих појединачних уређаја на ширем географском подручју, што представља велики проблем. Делећи захтеве који се постављају на коначан број делова, где је сваки елемент одговоран за једну појединачну област у функционалности повезивања, стварамо услове за повезивање комуникационе опреме широм света. На тај начин мрежа се дели на више функционалних слојева.

Многи испоручиоци опреме већ нуде системе са најчешће шест слојева. Сви они се могу уденути у платформу коју је израдила ***Међународна организација за стандарде за отворене системе повезивања*** (**International Standards Organisation's Open System Interconnection, ISO/OSI**) и која је позната под називом **референтни модел за отворену међусобну комуникацију** (**Open Systems Interconnection reference model**, **OSI**) који је развила међународна организација за стандарде. **OSI** модел дели проблеме везане за комуникацију на седам слојева, слика Б–12.

7. слој апликације	(**application layer**)	
6. слој представљања	(**presentation layer**)	апликационо оријентисани слојеви
5. слој сесије	(**session layer**)	
4. транспортни слој	(**transport layer**)	
3. мрежни слој	(**network layer**)	
2. слој повезивања	(**data link layer**)	слојеви који зависе од мреже
1. физички слој	(**physical layer**)	

Слика Б–12. Слојеви OSI референтног модела

Сваки слој извршава неке функције или услуге које су потребне нивоу, тј. слоју који је изнад њега, слика Б–13. Виши слојеви су растерећени функција које се обављају на нижем нивоу. Тако рецимо апликациони слој не треба да брине о формату оквира јер о томе брине слој за повезивање података (**data link**).



Слика Б–13. Промена формата поруке при проласку кроз разне слојеве

Важно је схватити основни концепт протока информација. Апликација на једном **host** рачунару размењује информације са одговарајућом апликацијом на другом рачунару, тј. са својим парњаком (**peer to peer**). Нека је то пренос датотека (**file transfer**). Чим је успостављена веза између апликација, два процеса размењују податке, али нема физичке везе на овом слоју. У стварности, само на најнижем слоју (физички слој) долази до физичке размене података. Дакле проток информација кроз апликациони слој се спушта наниже кроз слојеве све до ***слоја 1.*** где се збиља размењују битови података, а на другој страни се информација подиже навише. Размена се збиља обавља кроз ***физички слој*** јер је он једини слој у моделу где је веза дефинисана, где су успостављени нивои сигнала, тако да се низ података може пренети.

Овде се може поставити аналогија са писмом. Наиме, неко напише писмо, стави га у коверту, напише адресу и преда га у пошту. Када пошта испоручује писмо, прималац проверава адресу, отвара коверту и чита писмо. За физички пренос писма одговорна је пошта, али сам пошиљалац разматра пренос са аспекта преноса информација одговарајућем примаоцу–парњаку, без бриге о детаљима како ће писмо бити физички пренето (аутобус, воз, …). Дакле, проток информација се остварује између парњака на сваком слоју (**peer-to-peer**), али се размена података остварује само на физичком нивоу.

Слојеви 1, 2 и 3 су одговорни за успостављање везе–комуникације између чворова. Ниво 1, ***физички слој*** (**physical layer**), дефинише физичке и електричне захтеве за комуникациони медијум. Његов задатак је да пренесе појединачне битове између извора и одредишта и решава: колики ће бити напонски нивои логичке јединице и нуле, колико ће милисекунди (или микросекунди) трајати пренос једног бита, како се успоставља и прекида веза, колико прикључака има конектор итд . Овај слој се једини решава на нивоу хардвера (електричних уређаја), али и он поштује одређене стандарде за повезивање, као, на пример, **RS-232C**. На овом слоју су дефинисани и други стандарди као што су: **EIA 232–D**, **RS 449**, **CCITT X.21 bis**, **V.35** итд.

Слој 2, ***слој повезивања података*** тј. ***поуздане дигиталне везе*** (**data link**), описује како треба груписати битове у оквир (рам) и како да се приступи физичком медијуму који је одређен у физичком слоју (ниво 1). Обезбеђује да битови прихваћени од стране физичког слоја имају одређену структуру. Он је одговоран за пренос блокова података (дугачких информација). Откривање и исправљање грешака уграђени су у овај ниво (слој **data link**–а задаје кодове за детекцију грешака.), а постоји и могућност поновног слања података ако је то потребно. Присутно је управљање протоком података, односно на овом слоју се прилагођавају брзине преноса предајника и пријемника. **Token ring** и **Ethernet** протоколи су примери имплементације физичког и дела **data link** нивоа. На овом слоју дефинисани су стандарди као што су: **ISO HDLC** и **ANSI ADCCP**.

Мрежни слој (**network layer**) је одговоран за бирање логичког пута (**route**) између изворног и одредишног чвора. На овом нивоу се узимају из транспортног слоја блокови података величине једног пакета и пресликавају се на адресе из скупа мрежних адреса. Ако се захтева рутирање онда овај слој преузима одговорност за рутирање и поновно рутирање (**re-routing**). И на овом нивоу дефинисани су бројни стандарди: **EIA RS-2366A**, **CCITT X.25**, **X.75** (за аутоматско позивање) итд.

Транспортни слој (**transport layer**) дели дугачке поруке на мање целине, брине да ли су поруке пренете правилно и у правилном редоследу. Он користи разне механизме за проверу исправности (**error checking techniques**). На тај начин се обезбеђује квалитет било које поруке.

Слој сесије (**session layer**) успоставља позиве за размену низова битова података и обезбеђује синхронизацију размене података између апликација. Успоставља позиве за размену низова битова података. Он обезбеђује координацију између корисника избором међусобом прихватљивих протокола, и утиче на тачке за проверу које омогућавају опоравак података. **OSI** протокол дефинисан на овом слоју је **X.225**.

Слој презентације (**presentation layer**) пресликава различите репрезентације података у један спољашњи формат података који ће омогућити правилну интерпретацију информације на месту пријема. Он се бави проблемима синтаксе и семантике података. Пружа корисне, али не и неопходне, услуге као што су, на пример: промена кода (**ASCII** или **EBCDIC**), компресија, шифрирање /

дешифрирање података (**encryption/decryption**) које би могле бити потребне као средство за заштиту података. Он омогућава комуникацију између сасвим различитих рачунарских система.

Највиши ниво, ***слој апликације*** (**applications layer**) обезбеђује приступну тачку на комуникациони систем и утиче на размену информација између два апликативна процеса. Основне функције су му виртуелни терминал (прилагођава разне уређаје) и пренос датотека (разна правила за имена, прављење записа и сл).

Стварна комуникација захтева да сви чворови поштују одређене процедуре на свим нивоима. Сваки од ових слојева врши додатну модификацију података (како они иду наниже с једног на други слој у изворном чвору). Сви они, сем физичког слоја, додају своје заглавље, а **data link** заокружује поруку. Када су подаци примљени у одредишном чвору, мора се извршити инверзна модификација како подаци иду навише по нивоима у пријемном програму. Овај процес приказан је на слици Б–13.

Б – 3.3.1.3. IBM–ова SNA архитектура (SYSTEMS NETWORK ARCHITECTURE)

IBM има сопствени хијерархијски модел који се зове **systems network architecture**, **SNA**, и врло је сличан **OSI** моделу, слика Б–14. Кориснички интерфејс за апликационе програме у **SNA** моделу се зове **logical unit** (**LU**). Скуп логичких јединица, физичких јединица (рачунари, контролери) и комуникационих линија чине мрежну адресибилну јединицу (**network addressable unit, NAU**).

7. слој апликације	(**application**)
6. слој управљања функцијама	(**function management**)
5. слој протока података	(**data flow control**)
4. слој управљања преносом	(**transmission control**)
3. слој управљања путањом	(**path control**)
2. слој повезивања	(**data link control**)
1. физички слој	(**physical control**)

Слика Б–14. **SNA** референтни модел

Б - 3.3.1.4. Предности коришћења стандарда и слојева

Врло је корисно појаснити предности оваквог приступа. Идеја да се велики и сложен проблем подели на више једноставнијих делова је добро установљена. У њеној примени на **ISO** модел ствара се читав низ погодности укључујући:

- стандардне међусклопове (**interface**) између слојева, који допуштају да се они развијају сваки за себе независно. Дефинисањем веза између слојева, садржај слоја може да настави да се развија не утичући на остатак модела. Ако се развију неки нови и бољи алгоритми они се могу одмах увести у слој. Пажљивим управљањем може се постићи да обе верзије слоја истовремено живе све док траје тестирање нове верзије, а да живи подаци пролазе кроз текућу верзију. Када су поступци за тестирање и потврду завршени, нова

верзија се убацује у слој. Наравно када имамо неку верзију слоја која је јако распрострањена широм света (као на пример **TCP**) онда су нова унапређења ретка, јер би било потребно обновити милионе копија широм света да би обезбедили компатибилност повезивања мрежа, а то не би био прост посао.

- На сваком слоју се могу понудити услуге које једна другу замењују (алтернативе).
- Унутрашњи механизми у слојевима нису видљиви за друге слојеве.
- Слојеви могу бити потпуно уклоњени ако нису потребни, или се могу користити поједностављене верзије тамо где је то подесно.

Испоручивање пакета података мрежама широм света захтева постојање механизма адресирања заснованог на стандарду који одређује како се адресе додељују и интерпретирају. Без оваквих стандарда било би знатно теже подесити широм света распрострањену мрежу. На нивоу хардвера ово може бити адреса додељена некој мрежи и може бити коришћена на најнижем слоју OSI модела. Али да би имали једну међународну шему адресирања мора постојати концептуални механизам, а то захтева средства за доделу мрежних адреса из једне једине тачке. Ове адресе се додељују на мрежном нивоу који ради на **слоју 3** у OSI моделу.

У моделу са седам слојева протокола обезбеђује се да прималац прими податке које је послао пошиљалац. Протоколи на врху **слоја 3.** могу имати механизам за потврду пријема пакета података, али није неопходно да се они ту нађу. Међутим, постоји захтев у већини мрежа да имају одређено сазнање да су подаци успешно испоручени. То се постиже стављањем дела софтвера у међусклоп и код пошиљаоца и код примаоца да би подржали пролазак података на другу страну. Како софтвер остаје активан док су станице повезане на мрежу онда он треба да буду део софтвера на **слоју 4**.

Б - 3.3.2. СЛАЊЕ ПОРУКА НА МРЕЖУ

Прво уочимо да ако шаљемо неку јединицу података између две тачке увек постоји неки ниво шума, тј. постоји нека вероватноћа да се деси грешка. Што је већа група података која се шаље утолико је већа вероватноћа да се деси грешка у току преноса. Ако је јединица података сувише велика, вероватноћа појаве грешке постаће скоро једнака јединици. Дакле, никада нећемо бити сигурни да је пренос успешно обављен. Због тога се користе мање групе података да би се смањила вероватноћа грешке и тако се систем одржава у радном режиму.

Типично се поруке преносе ***од тачке до тачке*** (**point-to-point**) или техником ***прихвати и проследи*** (**store-and-forward**). Свака успутна тачка може бити способна да осигура интегритет података у тој тачки, или да одреди да ли се десила грешка током тог дела (крака) преноса. Дакле детекција грешке се може извршити на крају или у међувремену. Када би величина јединице података која се преноси била изузетно велика, на пример нека база података, вероватно би

дошло до појаве грешке при сваком покушају преноса. Да би се превазишао овај проблем јединица података која се допушта у једном преносу има ограничену дужину, па се веће јединице података деле на јединице мање величине које се зову ***пакети података***. Наравно да ће овај поступак изазвати појаву неких нових проблема као што је поновно састављање поруке на месту пријема и сл.

Обично се при преносу од 10^5 до 10^{10} битова јавља грешка у једном пренетом биту и обично се назива брзина грешке од 10^{-5} до 10^{-10}. Детекција грешке се обично остварује помоћу ***поља додатне цикличне провере*** (**cyclic redundanci check**, **CRC**) или софтверском провером дуж података. Пријемна станица је тада у могућности да потврди исправан пријем података (**Acknowledge**, **ACK**) или не потврди (**NAK**) ако су примљени подаци некоректни. Ако **ACK** или **NAK** не дођу до пошиљаоца током неког времена сматра се да је дошло до прекида (ако су подаци или **ACK** изгубљени) и подаци се поново шаљу. Наравно број покушаја поновног слања пакета је ограничен да не би дошло до тога да пошиљалац стално шаље један те исти пакет кроз прекинуту везу или непостојећем примаоцу. Ако неки део поруке није примљен након неколико покушаја читав пренос се одбацује, а станица пошиљаоца на крају преноса сигнализира грешку.

Путања поруке кроз мрежу

Без обзира да ли се користи **OSI**, **SNA** или неки други комуникациони софтвер мора се одредити пут од изворишног до одредишног чвора. Тај пут може бити кроз мрежу која користи везу од тачке до тачке (**point-to-point**) или магистралу. Мреже типа *од тачке до тачке* нормално захтевају да порука на путу до њеног одредишта прође кроз међучворове. Сваки од ових чворова, који се зову: **прихвати (анализирај) и проследи** (**store and forward nodes**), мора да прими поруку, одреди одредиште и да је проследи на одговарајућу линију. Како се види са слике Б–11. а) има много разних путева (начина) да се дође из једног места у друго у систему тачка по тачка. Али и у системима са магистралом порука може, у стварности, имати више разних могућих путања.

Користе се две главне врсте услуга (**service**) за пренос поруке са једног места на друго. У једном случају може се, пре слања било ког пакета, успоставити веза кроз мрежу кроз коју ће путовати сви пакети података. У прошлости ово је обично била физичка путања која се остваривала повезивањем мрежа као код телефонских система. Код рачунарских мрежа, развијена је техника комутације пакета, па дакле та путања која се остварује није посвећена и не мора бити физичка, али јесте логичка и чине је виртуелна кола и зове се *услуге привидних уређаја* – **virtual circuits service**. Она захтева успостављање логичке мреже, везе од извора до одредишта пре него што се пошаље нека порука. Логичка веза мора бити успостављена у току сесије, након тога се она ослобађа (када је сесија завршена). Ово је приступ који се користи у мрежама са телефонском везом. Поруке које путују дуж логичких кола се шаљу у одређеном поретку и примају се у том истом поретку. Када се обави пренос читаве поруке, тј. свих пакета, виртуелна кола се раздвајају.

Многи комуникациони системи користе процес који се зове пакетска комутација (**packet switching**) да би постигли што ефикасније коришћење мреже. Пакетна обрада је слична временском мултиплексу јер се порука дели на мање делове, пакете (**slot, packet**). Сваки пакет се засебно шаље. Код овог преноса информа–ција о путањи садржана у сваком пакету била би врло слична и указивала би на логичку путању коју треба следити. Самим тим, та информација не мора бити садржана у сваком пакету, дакле не мора сваки пакет да садржи све детаље о адреси одредишта. Путовање засновано на виртуелним колима успостављањем логичке путање дуж које плове сви пакети има још неке предности. Пакети ће стизати на одредиште у истом редоследу у ком су послати, па ће поновно ус–постављање дугачке поруке бити врло једноставно.

При успостављању виртуелне везе креира се једноставан идентификатор логичке путање. Овај идентификатор **ID** је једина информација која је потребна за праћење путање. Ово је знатно погодније него када сваки пакет мора да носи потпуну адресу одредишта када путања није успостављена. Ако пакет треба да буде прослеђен кроз неколико успутних тачака то се осигурава преко очитавања **ID**–а виртуелног кола, док се код других система (без успостављања везе) у сваком чвору мора доносити одлука о даљој путањи пакета. Овај приступ са виртуелним колима је погодан само за дугачке поруке када се веза успостави и држи дуже време. Ако је то цео дан онда је то стална веза (**permanent virtual circuits**), а ако се одржава краће време онда је позната под именом комутирана (**switched virtual circuits**). Једна од мана виртуелних кола је време потребно да се успостави веза на почетку преноса и раскине на крају сесије. Ако је потребан само краткотрајни пренос ово се може показати као неефикасно.

Друга могућност је да се не успоставља претходна путања и да сваки пакет има слободну путању независну од претходника или следбеника. Ово је пренос без успостављања везе или **datagram servises** и сваки пакет мора да носи потпуну адресу одредишта и један показивач на његову позицију у низу порука јер пакети који се шаљу кроз различите путање могу да стигну измењеним редоследом. Дакле одредиште мора да обезбеди складиштење приспелих пакета и ус–постављање оригиналног редоследа када они стигну погрешним редоследом. Овај систем је сличан класичном поштанском систему за испоруку и не гарантује: квалитет порука, путању којом ће две узастопне поруке ићи нити редослед примљених порука. Ово личи на начин на који се писма испоручују кроз услуге поште.

Изворни чвор испитује одредишни чвор и стање мреже и шаље поруку на одговарајућу везу ка одредишту. Два пакета једне исте поруке могу путовати разним путањама до њиховог одредишта. Поступак одређивања пута којим ће порука бити послата зове се рутирање (**routing**), и често се базира на принципу најјефтинијег пута. Рутирање садржи више корака. Прво, треба одредити које су опције на располагању. Друго, направити табелу рутирања до сваког чвора да би се добиле најбоље везе. Треће, конвертовати табеле у облик који може бити коришћен за слање долазећих порука ка одредишту.

Откривање/отклањање грешке

Пре него се граде методе за отклањање грешака морају постојати методе за откривање грешака. Историјски гледано, код асинхроног преноса података сваки пренети карактер има једноставну проверу парности техником парне или непарне парности. Бит парности се поставља на потребну вредност и пријемник проверава и потврђује да се број примљених битова подудара. У противном генерише се грешка.

Збирна провера

Повећавањем дужине поруке која се преноси, развијане су и друге технике провере. Типична је она по **CCITT** стандарду која генерише 16 или 32 бита **CRC** (допунска, сувишна, додатна циклична провера, **Cyclic Redundancy Check**). Она подразумева употребу унапред одређеног генераторног полинома. То је неопходно јер одредишна станица испитује примљени **CRC** да би била сигурна да је пренос исправан, она такође на бази примљених података израчунава сопствени **CRC**. Дакле и пошиљалац и прималац морају израчунавати **CRC** по истом унапред познатом стандарду. Податак се дели са генераторним полиномом и остатак представља CRC. Овај метод дељења користи аритметику по модулу 2 која се ефективно своди на искључиво ILI. Генерисање CRC је уграђено у хардвер који садржи генераторе полинома. Ово обезбеђује висок проценат откривања и идентификовања грешака.

Ову технику ћемо описати на једноставном примеру. Нека је дат стандардни полином типа: $x^4 + x^2 + 1$. Познат је низ битова који представља присуство или одсуство сваког тежинског елемента: постоји елемент тежине 4, не постоји елемент тежине 3, постоји елемент тежине 2 итд. Ово резултира у постојању дигиталне презентације 10101. Дигитална презентација полинома (делиоца) се сада користи за дељење податка (на пример низ 11011). Пре него што извршимо дељење додамо онолико нула на крај податка колики је степен полинома (у овом случају 4). Дакле дељеник је 110110000. Рачунска операција дељења по модулу 2 своди се на операцију **XOR**.

110110000 : 10101 = 1110 са остатком 1001 = **CRC**

дакле податак који се преноси је оригинални на који се додаје остатак: 11011**1001**. По **CCITT** стандарду за полином реда 16 се узима $x^{16} + x^{12} + x^5 + 1$, а 32–битни низ је још сложенији, али је примена врло једноставна. Јасно, ово је сложен поступак за детекцију грешке и понекад је неопходно уградити једноставније провере. Ово је нарочито случај када се пакети деле на врло мале јединице јер је то неопходно у неким мрежама дуж путање. Тада се примењују врло једноставни системи провере као што је комплемент јединице, а то је најчешће случај код бајта заглавља података. Наравно тада можемо једино бити сигурни да ће пакет бити испоручен правом одредишту. Грешке могу настати у самим подацима фрагментираног пакета, и оне не могу бити откривене на овај начин. Али када сви од фрагментираних пакета стигну, пакет се поново успоставља и онда он може бити проверен јер сваки пакет још увек има свој сопствени **CRC**.

Отклањање грешке

Када је нека грешка откривена потребно је имати одговарајућу технику за опоравак података, тј. отклањање грешке. У најпростијем случају када се шаље само један пакет (и нема других пакета) поступак је следећи: Прималац шаље **NAK**, пошиљалац поново шаље податке и прималац потврђује исправан пријем са **ACK**.

Овај поступак је крајње неефикасан и зато се уграђују друге технике у трансмисиони медијум. Знатно боље од чекања на одговор јесте да се осигура да одредиште коректно идентификује долазак пакета, тако што сваки пакет има свој ID или редни број у низу. Један ACK може да потврди пријем више пакета. Ако је пак откривено да један од пакета има грешку, онда се отпочиње процес за отклањање грешке. Нека је рецимо други у низу од три пакета некоректно испоручен, током времена биће откривена грешка али трећи пакет ће већ бити послат.

NAK који ће бити враћен пошиљаоцу указиваће да у пакету 2 постоји нека грешка и да га треба поново послати. Протокол ће сада да тече једном од две могућности. Да поново пошаље пакет два и да настави да шаље пакет 3, 4 и тако даље, или да пошаље пакет 2, а онда да настави да шаље пакет 4 итд. Први поступак је познат под именом **Go-Back-N** и изискује слање великог броја података кроз мрежу, али се чешће користи јер је врло једноставан за примену. Наиме, прималац не мора да чува пакете са већим ID–ом нити је потребно правити прорезе у поновном преносу података и у њиховом коректном пријему на одредишту. Ова друга техника позната је под именом селективна ретрансмисија. Употреба идентификатора пакета носи у себи неке опасности. Наиме, ако се за ID или редни број користе три бита у заглављу пакета онда се могу послати највише 8 пакета са заједничком потврдом. Њихови редни бројеви су од 0 до 7. Нумерација пакета се обавља помоћу кружног бројача који после редног броја 7 отпочиње бројање од 0. То се означава као величина прозора. Ако не добије ACK пошиљалац не сме да настави пренос нове групе пакета јер више не би било могуће извршити правилну ретрансмисију погрешно примљеног пакета. Ово је корисно и као техника за управљање протоком порука у случају када је одредишна станица спора а добија податке од брзог пошиљаоца.

Б - 3.4. LAN МРЕЖЕ

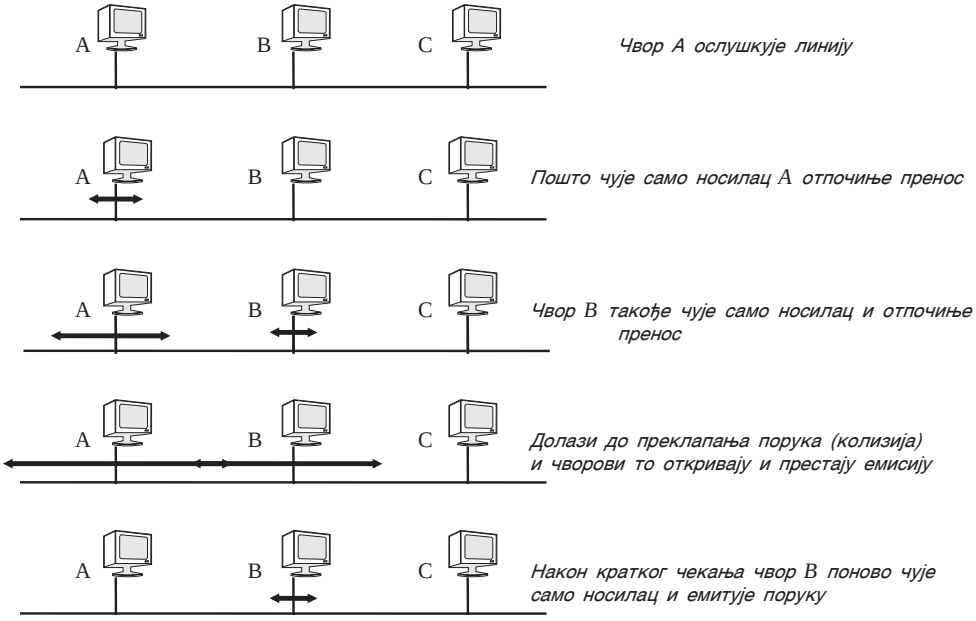
Мреже локалног подручја су, како само име каже, приватне, мале мреже које омогућавају велику брзину повезивања између неколико процесора и периферијских уређаја који су смештени на ограниченом географском простору. Најчешће коришћене **LAN** топологије су: магистрала типа **Ethernet** и прстен са жетоном типа **token ring**. Обе ове мреже имају исти проблем: само један чвор може да врши комуникацију, тј. пренос података у току једног интервала времена. Процедуре које су пројектоване да регулишу коришћење комуникационог медијума називају се протоколи за приступ (**access protocols**). Најчешће коришћени **LAN** про–

токоли су: **Carrier Sense Multiple Access with Collision Detection** (**CSMA/CD**) и прослеђивање жетона (**token passing**).

Б - 3.4.1. ETHERNET

Ethernet је добио име по речи етар (**ether**) која се често користи као синоним за комуникациони медијум. Као и многе друге новине у рачунарској техници, и **ethernet** мреже су развијене у **Xerox Palo Alto Research Center**–у. **Ethernet** користи топологију магистрале, а комуникационим медијумом се управља помоћу **CSMA/CD** протокола. Комуникационе линије имају један специјални сигнал који се зове носилац (**carrier**) који је присутан чак и када нема преноса података. **CSMA/CD** протокол захтева да сваки чвор, који хоће да пошаље поруку, прво ослушкује носилац (**carrier sense**). Ако је медијум (етар) тих (слободан), чвор може да изврши пренос. Ако је медијум заузет, чвор мора прво мало да сачека и онда да поново покуша, тј. ослушне етар.

Може се десити да два или више чворова детектују да је медијум слободан и онда ће сви отпочети пренос. То значи да ће на магистрали истовремено постојати две или више порука (сетова података). Оваква ситуација се назива **сударили** колизија (**collision**), слика Б–15. Чим је колизија детектована, сви чворови морају да обуставе пренос. Сваки од чворова сада мора да сачека неки случајно изабрани интервал времена пре него што поново ослушне етар да би поново покушао да изврши пренос.



Слика Б–15. Судар–колизија: детекција и отклањање

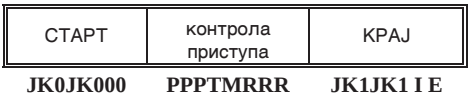
Ethernet систем шаље податке од једног чвора до другог у облику пакета. Овај пакет се састоји од одређених делова, слика Б–16. Преамбула је 64–битно поље за синхронизацију. Одредишна адреса задаје чвор или чворове којима је порука послата а изворишна адреса показује (указује) на чвор који шаље пакет. **Type** поље показује како треба интерпретирати поље података, односно који је тип рама за податке. Поље података (**data field**) садржи од 46 до 1500 **byte**–ова

Слика Б–16. Пакет Ethernet–а

На крају Ethernet пакета се налази поље ***додатне, редундантне цикличне провере*** **CRC** (**cyclic redundancy check**). Сваки чвор који је повезан на магис–тралу испитује сваку поруку посматрајући да ли је она адресирана на њега. Када чвор открије поруку која је адресирана на њега, он најпре проверава, затим потврђује **CRC** и тек онда прихвата податке.

Б - 3.4.2.. TOKEN RINGS

Један другачији приступ управљања комуникацијом користи специјалне поруке које се зову **жетон, белег** или **token**, слика Б–17, а служе да укажу да ли је систем расположив, тј. да ли је слободан, или је у току неки пренос података. Дакле, постоје две врсте **token**–а: заузети и слободни.



Слика Б–17. Жетон – Token

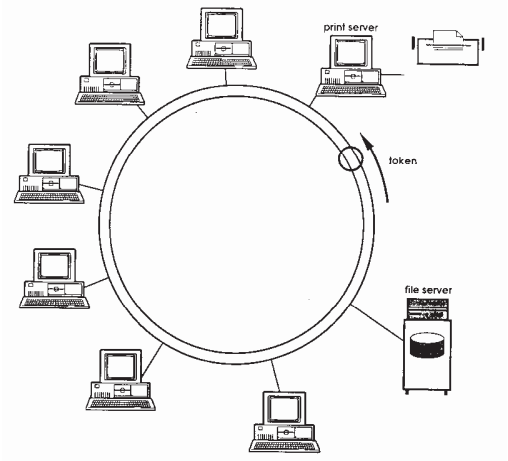
Сваки чвор у **token ring**–у чита и реемитује све поруке. Било који чвор који жели да пошаље поруку мора да сачека док **token** постане слободан (тј. **T** бит је нула). **Token** се тада преузима и овај чвор уписује у **T** бит јединицу (**T=1**). Битови за приоритет (**P**) и резервацију (**R**) помажу да се одреди који чвор може да преузме жетон и који може да га користи, тј. да пошаље поруку. Формат поруке дат је на слици Б–18. Монитор бит (**M**) се користи да се спречи непре–кидна циркулација заузетих **token**–а. Постављање бита **E** означава да је детек–тована грешка. Оквир са подацима садржи сличне информације као и **Etnernet** пакети. Битна разлика је то што има још два поља: *поље провере оквира* (**frame check**) са осам битова: **FF** (означава врсту оквира) и **ZZZZZZ** управљачки би–тови, а на крају оквира се налази *поље стања* (**status field**) са осам битова: **ACRR ACRR**. Када је оквир (**frame**) први пут постављен на прстен, његови би–тови стања **A** и **C** постављају се на нулу. Одредишни чвор мора да покаже

пошиљаоцу да је оквир примљен, а он то чини тако што мења бит **A** са нуле на јединицу. Ако су подаци копирани, одредишни чвор онда поставља и бит **C** на јединицу.

СТАРТ оквира		секвенца за проверу и пакет података					КРАЈ оквира	
СТАРТ	контрола приступа	провера оквира	адреса одредишта	адреса извора	П О Р У К А	провера оквира	КРАЈ	Статус оквира

Слика Б–18. Формат поруке у мрежи типа **token ring**

Одредишни чвор не избацује рам (оквир) са прстена. Он једноставно само копира податке, поставља одговарајуће битове стања и све што је примио поново прослеђује даље на прстен. Рам са подацима остаје на прстену све док поново не дође назад у изворни чвор. Чвор који је послао податке (изворишни чвор) повлачи поруку са прстена, испитује битове стања, ако су јединице он шаље **слободан жетон** на прстен. На слици Б–19. је дат пример једног прстена са жетоном. У несрећном случају неке грешке на изворној станици након генерисања података, а пре њиховог повлачења, систем за надгледање ће идентификовати проблем, очистиће прстен и поново поставити жетон. У ову сврху може послужити и само један бит (**M**) који поставља станица пошиљалац и који може ресетовати само она. Ако је дошло до квара на овој станици структура података одлази у систем за надгледање (**monitor**), и он препознаје ову ситуацију тако што је стање бита за надгледање супротно. Станица која се понаша као монитор онда сређује, чисти прстен и поставља нови жетон. Извор може задржати жетон током времена променљивог трајања, па могу бити послати и пакети података променљиве дужине. Пре него је од стране пошиљаоца слободан жетон постављен на мрежу, морају бити остварена два услова:



Слика Б–19. Прстен са жетоном

1) Подаци су послати и
2) Почетак података се мора вратити пошиљаоцу (од овог правила се одустаје у неким изведбама).

Када су услови задовољени на мрежу се шаље *слободан жетон* и друге станице су слободне да га преузму. Овај механизам обезбеђује да у једном тренутку у прстену постоји само један жетон. Ово је тачно за 4 Mbps изведбе, али мреже са прстеном за веће брзине података уклањају овај услов.

Да би се обезбедила потврда да су подаци примљени исправно, додато је једно једноставно поље за потврду на крају преноса података. Пријемна станица може једноставно да стави у ово поље да је примила податке, или постави да је станица заузета (или нема никаквог одговора). Када се подаци врате пошиљаоцу ово поље се испитује да би се видело да ли је потребно поновно слање (**re-transmission**). Ова потврда се обично може остварити помоћу два бита. Један (**A**) који казује да ли је адреса препозната од стране жељене станице и други (**C**) који показује да ли су подаци копирани у одредишну станицу или су игнорисани јер је одредишна станица заузета (**R**).

Могуће је и увести одређени приоритет међу станицама што се лако постиже употребом *поља за контролу приступа*. Ово ће омогућити неким станицама да имају већа права у погледу приступа прстену у односу на мања права неких других станица. Док заузети жетон пролази кроз мрежу поље за резервацију може бити постављено од стране станице која жели да приступи прстену. У ово поље се уписује ниво приоритета станице која жели да обави пренос. Станица са вишим приоритетом може да заузме ово поље и у њега упише свој ниво приоритета, наравно ако она жели да приступи прстену. Када се жетон врати станици која управо обавља пренос она га препознаје и она шаље слободан жетон који, наравно, садржи поље за резервацију у које је уписан одређени ниво приоритета. Овај слободни жетон са "високим" нивоом приоритета пролази несметано кроз мрежу све до станице која има тај ниво приоритета и жели да обави пренос. Ова станица га онда преузима и заузима прстен. Након што обави пренос, станица која је поставила поље за приоритет мора и да га спусти на претходни ниво и то у тренутку када ослобађа жетон. У пракси, ова могућност се користи ретко. Једна од предности прстена са жетоном је да су особине система стабилне и предвидиве и у условима највећег оптерећења.

Б - 3.5. *WAN МРЕЖЕ*

Локалне рачунарске мреже, без обзира на све предности, ипак су изоловане целине које имају ограничен потенцијал и корисност. То је последица разних ограничавајућих фактора: број рачунара, њихових могућности, расположивих ресурса, база података и слично. **LAN** мреже како им само име каже имају и ограничен географски домет. Давно је постало јасно да се знатно већи ефекти могу постићи ако корисници деле ресурсе, не само неке своје мреже, већ и ако имају приступ и ресурсима других мрежа (чији нису власници). Самим тим по-

јавила се потреба да се мрежа простире на знатно ширем географском подручју, и да се међу собом повезује више разних мрежа. Да би **LAN** и **WAN** мреже могле да се повезују и да функционишу на ширем географском подручју, морају постојати и одређени уређаји који омогућавају повезивање мрежа. Ови уређаји подржавају рад мрежа на различитим слојевима **OSI** референтног модела:

- ***понављачи, репетитори*** (**repeaters**): уређаји који на физичком нивоу повезују два дела једне исте мреже, односно два сегмента која су компатибилна на нивоу карактеристика сигнала који се преносе преко неког физичког медијума. Омогућава повећање броја сегмената мреже (и броја станица и удаљености међу њима). Обично уноси неколико битова кашњења потрбних да се сигнал обнови (ради на нивоу бита).
- ***мостови*** (**bridges**): уређаји који повезују мреже на нивоу **data link**–а, па се могу користити за повезивање делова и мрежа са различитом организацијом. Преко њих се могу повезати две мреже од којих је једна **ethernet** а друга **token ring**. Раде на принципу **прихвати и проследи** па самим тим могу да прилагоде свој излаз према другој мрежи која не мора бити истог типа. Сем тога ови уређаји обављају и одређено филтрирање пакета на бази табеле адреса, чиме се смањује непотребни саобраћај на појединим сегментима мреже. Мост тачно зна на којој страни се налазе поједини уређаји (адресе) и на ту страну усмерава само поруке које имају одговарајућу одредишну адресу.
- ***усмеривачи, рутери*** (**routers**): уређаји који повезују мреже на мрежном нивоу (**network layer**). Интелигентнији су од мостова, и могу на бази одредишне адресе изабрати један, најповољнији, пут од више могућих путева. Користе се за повезивање две административно одвојене мреже које раде независно једна од друге. Усмеривачи раде са мрежним адресама (на пример интернет адресама) које нису исте као хардверске адресе са којима раде мостови. Уочимо да интернет адреса остаје иста без обзира што се на мрежу пријављујемо са разних рачунара (са разним физичким адресама). Наравно ако променимо мрежу, мења се и интернет адреса.
- ***капије, пролази*** (**gateways**): уређаји који покривају све слојеве **OSI** модела и раде на мрежном и вишим слојевима. Они омогућавају повезивање различитих рачунарских мрежа (пример **DECNET** и **SNA**) на апликативном нивоу, и на тај начин обезбеђују потпуно коришћење рачунарских ресурса и пуну комуникацију међу процесорима.

Уређаји који обављају ове функције могу бити реализовани као засебне јединице или се пак поверавају рачунарима опште намене. Када се реализују као засебни уређаји (рачунари посебне намене), онда раде само тај посао и имају врло високе перформансе и ниску цену. Кад се повере рачунарима опште намене онда тај рачунар сем услуга које пружа корисницима обавља и неку од наведених функција. Када су комуникације врло интензивне може се десити да тај рачунар не стиже да обави друге послове, па је боље решење са посебним, специјализованим уређајима.

Б - 3.5.1. INTERNET

Internet је **WAN** мрежа рачунара који користе **TCP/IP** протокол (мада од скоро и друге мреже као **BITNET** или **DECnet** које нису **IP** преко **gateway**–а нуде своје сервисе). Језгро мреже чине рачунари који су распрострањени широм земаљске кугле и перманентно су повезани везама велике брзине. Сваки рачунар прикључен на неки од ових рачунара може успоставити везу са било којим другим рачунаром у мрежи било да је у истој просторији или на другом континенту. **Internet** се често назива "**мрежа свих мрежа**".

Бити прикључен на **Internet** на било који начин значи имати на располагању огроман број информација, могућност слања електронске поште, пренос докумената али и друге разне услуге као што су куповина, поруџбине, посета галеријама, читање књига, играње, читање вести, добијање разног **software**–а и слично. **Internet** постаје интегрални део свакодневног живота као и **fax** или телефон.

Прародитељ **Internet**–а је **ARPANET** (**Advanced Research Project Agency**) коју је 1969. године основала америчка влада у циљу развоја комуникација. 1980. године *Национална научна фондација* (**Natinal Scientific Fondation**, **NSF**) основала је **Internet** мрежу која се касније спајала са другим већ постојећим мрежама, да би коначно 1990. године дошло до повезивања са **EARN**–ом (**IBM**–ова мрежа која је постојала у многим земљама пре свега Европе), **JANET**–а (у Британији) **NORDUnet** у Скандинавским земљама, **FUNET**–а у Финској и тако је настао данашњи **INTERNET**. Данас у састав **Internet**–а улази преко шеснаест милиона рачунара домаћина (**host**), а колико је динамичан развој **Internet**–а најбоље показује табела на слици Б–20.

датум	број рачунара домаћина	подручја–области (domains)	класа мреже *велике*	*средње*	*мале*
јан 96	9 472 000	240 000	92	5 655	87 924
јул 95	6 642 000	120 000	91	5 390	56 057
јан 95	4 852 000	71 000	91	4 979	34 340
окт 94	3 864 000	56 000	93	4 831	32 098
јул 94	3 212 000	46 000	89	4 493	20 628
јан 94	2 217 000	30 000	74	4 043	16 422
окт 93	2 056 000	28 000	69	3 849	12 615
јул 93	1 776 000	26 000	67	3 728	9 972
апр 93	1 486 000	22 000	58	3 409	6 255
јан 93	1 313 000	21 000	54	3 206	4 998

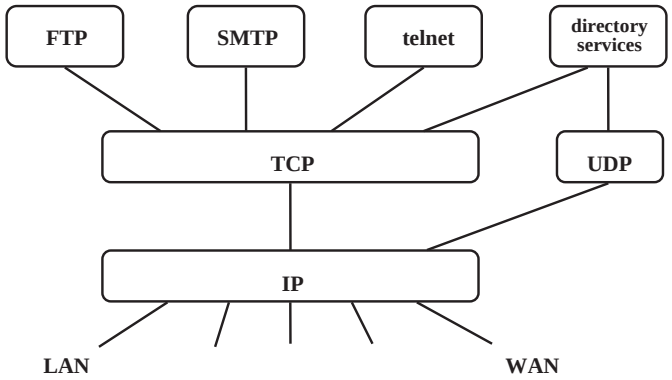
Извор: Network Wizards

Слика Б–20. Пораст броја рачунара домаћина у **Internet** мрежи

Да бисмо се повезали на **Internet** и да бисмо могли да користимо све његове могућности, довољно је имати рачунар **PC** 486 и бољи, са **Windows** графичким корисничким интерфејсом и што више **RAM** меморије. Да би постигли пуну брзину, треба имати **modem** са 56 **Kbps**, а за мултимедијалне презентације и **CCD** камеру, микрофон, колор монитор, звучну картицу и звучнике. Све ово кошта око 2000$, а могућности које пружа **Internet** су практично безграничне.

IP- Internet Protocol

Internet protocol (**IP**), је основни блок на којем се гради **Internet**, и он представља међумрежни протокол на слоју 3 **OSI** референтног модела (**network layer**), слика Б–21. Задатак му је да обезбеди комуникацију између мрежа и рутирање порука на бази приступа **datagram** услуга. Истовремено он подржава и читав низ услуга које обезбеђују мало кашњење, широк пропусни опсег и високу поузданост.



Слика Б–21. **TCP/IP** услуге

Поруке (подаци) које се крећу мрежом издељене су на пакете. Информације унутар пакета су обично од једног до 1500 бајтова. Ово онемогућава било ког корисника мреже да монополизује мрежу. Али, исто тако, ако је мрежа оптрећена, њено понашање постаје горе за све кориснике подједнако. Различити делови **Internet**–а су повезани рачунарима који се називају рутери и који спајају мреже. Ове мреже могу бити и **Ethernet** и **Token Ring**, а најчешће се спајају посредством телефонских линија. Рутери одлучују како (којим путем) се шаљу пакети од којих су састављене поједине поруке. Они имају информације о томе које су везе на располагању и праве оптималне путеве за пакете.

Сваки рачунар на **Internet**–у има јединствену адресу. **Internet** адреса се састоји од четири броја, сваки мањи од 256. Када се напишу ове адресе изгледају, на пример, овако: **192.112.36.5** или слично. Пошто је **Internet** мрежа свих мрежа, адреса се састоји из дела који говори о мрежи којој припада рачунар који шаље поруку (извор) и дела који идентификује мрежу **host**–а, тј. рачунара домаћина који прима поруку (одредиште). Ако је порука дужа од максималне величине пакета (1500 карактера), порука се онда састоји из више пакета. У таквим случајевима, пакети који иду различитим рутама могу стићи до одредишта у неправилном редоследу, или се могу изгубити у потпуности. Зато са на **IP** надовезује следећи слој мреже а то је **TCP**.

TCP -Transmission Control Protocol

TCP је протокол на слоју 4 (**transport**) и директно је преузет из **Arpanet**–а. Он обезбеђује такозвану "с краја на крај" комуникацију (**end–to–end**). **TCP** узима информацију која се шаље и разбија је на делове који су нумерисани тако да се њихов пријем може верификовати (**TCP** на пријемној страни може тражити ре–трансмисију–поновно слање одређеног пакета) и може се обавити реконструкција поруке. **TCP** на тај начин додаје нов део на **IP** адресу. Због могућности измене појединих битова информације у току преноса (због лоших веза и сл.) у **TCP** је уграђен и механизам за проверу да ли је информација приспела на одредиште иста она која је послата са извора. То је **checksum** и он омогућава **TCP**–у на одредишту да види промену и тражи ретрансмисију ако је пакет измењен.

Дакле, **TCP** прави илузију да онај који шаље поруку има додељену линију са одредиштем, док у ствари многи корисници користе исте линије. Ако су поруке кратке, **TCP** значи беспотребно успоравање и зато за апликације са кратким порукама постоји **UDP** протокол (**User Datagram Protocol**), који је далеко јед–ноставнији од **TCP**–а, јер не води рачуна о секвенцама пакета, нити о ис–прављању грешака. Ретрансмисија пакета се обавља ако порука не стигне у одређеном року. Све остале случајеве обрађују саме апликације које користе ко–муникацију.

Протоколи на нивоу апликација

Апликације су још један ниво мреже који се додаје на **TCP**. Оне омогућавају корисницима да целој комуникацији дају смисао. Постоје три стандардна прото–кола за **Internet** апликације и то су: **telnet** повезивање на даљину, **SMTP** (**Sim–ple Mail Transfer Protocol**)–електронска пошта и **ftp** (**file transfer**)–пренос да–тотека. **Telnet** омогућава емулацију терминала и на тај начин обезбеђује корис–нику повезивање и преузимање контроле над апликацијом која је на удаљеној машини (**remote login**). **SMTP** омогућава слање текстуалних порука између било која два уређаја. **FTP** омогућава преношење датотека између било која два **host** рачунара на **Internet**–у.

Осим ових, постоје и друге алатке за претраживање и лакше проналажење ин–формација на **Internet**–у које су базиране на менијима или индексном пре–траживању. Најновији од њих је свакако **WWW** или **World Wide Web** који зах–ваљујући **hypertext**–у дозвољава повезивање (**links**) и **cross reference**. Такође дозвољава пренос слике, видео и аудио порука.

DNS - Domain Name System

Internet је логички подељен на области (**domains**). Област подразумева две ди–мензије: делатност организације којој припада мрежа и земљу у којој се налази.

Типови домена су обично следећи:

- ***.ac*** (академска мрежа),

- ***.com*** (комерцијална организација),
- ***.net*** (**provider** за приступ мрежи),
- ***.edu*** (образовна институција),
- ***.gov*** (владине агенције),
- ***.mil*** (војне),
- ***.org*** (непрофитне организације) итд.

Свака земља има свој код па тако:

- ***.au*** (Аустралија),
- ***.ca*** (Канада),
- ***.de*** (Немачка),
- ***.fr*** (Француска),
- ***.yu*** (Југославија),
- ***.uk*** (**United Kingdom**) итд.

Internet користи **Domain Name System** паралелно са **IP** адресирањем. На овај начин се уместо бројева користе слова и речи. Ако анализирамо **e-mail** адресу **slobo@vets.edu.yu** видимо да је адреса формата: **user@organization.domain,** где је **организација** у овом случају **vets** и овде је **vets** поддомен домена .**edu** (а он је добављач услуге–провајдер), док је кôд земље .**yu** што означава Југославију.

Услуге које нуди Internet

Број апликација и услуга на **Internet**–у расте свакодневно и практично је лимитиран маштом и технолошким могућностима. Поменимо само неке:

E-mail: Електронска пошта представља најстарију, али и најпопуларнију услугу коју ће до краја века користити преко 100 милиона људи. Тренутно сваког месеца кроз **Internet** прође преко 1 трилион бајтова који чине више од милијарду писама. Оно што је карактеристично за електронску пошту је да су све поруке у **ASCII** коду, да није неопходна стална веза са мрежом, тј. примаоцем и да се текст, за разлику од факса, може даље обрађивати. Могуће је и слање слика, цртежа формула и других врста датотека, али се оне помоћу одређених програма такође конвертују у **ASCII**. То није проблем, јер се на месту пријема, уз помоћ програма за обраду може поново добити слика, односно оно што је стварно послато.

File transfer: Има преко 8000 **FTP** услуга које су тренутно на располагању. Датотеке веће од 50 **KB** најпре се компримују применом неке од расположивих техника: **.Z** или **.tar** за **UNIX** системе, **.zip** за **MS–DOS** или **.hqx** за **Macintosh**.

WWW (**World Wide Web**): **WWW** је мрежа докумената који међусобно указују једни на друге, и на тај начин омогућавају брзо и лако шетање по мрежи, такозвано сурфовање. Ова услуга подржава мултимедијалну комуникацију и заснована је на хипер текст систему. Користи графички интерфејс што га чини врло једноставним за употребу. Наиме, на означени део текста кликнемо мишем и аутоматски прелазимо на нови текст који постаје изабрани документ. Пратећи те везе (**links**), ми се уствари крећемо од сервера до сервера, а да при томе не примећујемо никакву разлику у погледу њиховог оперативног система или хард–

вера. **WWW** је организован по принципу клијент/сервер, где клијент и сервер комуницирају посредством протокола **HTTP** (**hyper text transfer protocol**). Сем ових на располагању су и многе друге услуге засноване на менијима и индексном претраживању попут: **BBS** (**Bulletin Board Systems**), **Library Catalogs**, **Online Banking**, **Video Broadcasting**, **Radio Broadcasting**, **Internet Telephony** итд..

Б - 3.5.2. Intranet

Intranet је неочекивани корак супротан интернету. У суштини то су **mini–internet**–ови које су развиле поједине организације или групе организација. Они обезбеђују приступ информационим ресурсима унутар једне компаније, универзитета или институције. Ове мреже користе исте протоколе као и **Internet**. Иако су се појавили тек 1995 **Intranet**–ови се брзо развијају и показало се да је тржиште **intranet** софрвера имало промет већи од милијарду долара већ крајем 1997. године.

Б - 4. ЗАКЉУЧАК

Класична подела рачунара на **mainframe**, **mini** и **micro** рачунаре данас је практично превазиђена. Данас се чешће класификација рачунара врши на бази броја централних процесора, тако да разликујемо: **simplex** (са једним централним процесором), **duplex** (са два централна процесора) и **мултипроцесорске системе** (са више од два централна процесора). Већи број процесора омогућава већу брзину обраде, али и поузданост рачунарског система (имуност на квар и грешку). У једном рачунарском систему, по правилу, постоји више разних врста процесора. Једну групу чине процесори опште намене (централни процесори), а другу групу чине специјализовани процесори. Ови специјални процесори обично помажу централном процесору у обављању неких посебних функција, па се зато зову и помоћни процесори. Посебну врсту рачунарских система чине дистрибуирани системи или рачунарске мреже. Рачунарску мрежу могу сачињавати рачунари који су лоцирани на малом географском подручју (на пример, у једној соби, на једном спрату или у једној или више мало удаљених зграда). Такве приватне мреже које користи релативно мали број корисника зову се мреже локалног подручја – LAN, и омогућавају велику брзину размене и безбедности порука. Другу врсту мрежа чине мреже које користе комуникационе медијуме под контролом државе, и које су лоциране на ширем географском подручју (у једном граду или на разним континентима). То су такозване WAN мреже. Ове мреже пружају значајне погодности великом броју броју корисника, а у том погледу посебно треба поменути "**мрежу свих мрежа**"–Internet. На њој се може наћи огроман број разнородних информација, а корисницима је на располагању велики број услуга (електронска пошта, WWW итд.).

Употреба рачунарских мрежа пружа посебне погодности у погледу брзине и поузданости, али намеће и низ захтева који се тичу пре свега безбедности података. У ту сврху постоје бројни стандарди који се морају поштовати при изради мрежних апликација, посебно када се ради о јавним мрежама. Користе се разне технике шифрирања, провере аутентичности (**passwords**, **intelligent tokens**), ауторизације корисника итд.. Но без обзира на све, рачунарске мреже представљају садашњост и будућност рачунарских система.

Б - 5. КЉУЧНЕ РЕЧИ

- минирачунар (**minicomputer**)
- микрорачунар (**microcomputer**)
- **mainframe**
- рачунар са једним **CPU** (**Simplex**)
- мултипроцесорски системи (**multiprocessor system**)
- круто, чврсто повезани системи (**tightly coupled systems**)
- лабаво спрегнути системи (**loosely coupled systems**)
- имуни на грешку (**fault tolerant**)
- системи са два независна процесора (**tandem**)
- принцип већинског гласања (**voting**)
- процесори за подршку (**support processors**, **attached support processors**)
- дистрибуирана обрада (**distributed processing**)
- упредене парице (**twisted pair**)
- телефонска линија (**phone line**)
- коаксијални кабл (**coaxial cable**)
- оптички кабл (**optical fiber**)
- микроталасне везе (**microwave**)
- радио везе (**radio**)
- сателитске везе (**satellite**)
- модем (**modulator–demodulator**)
- амплитудна модулација (**amplitude shift keying**, **ASK**)
- фреквентна модулација (**frequency shift keying**, **FSK**)
- фазна модулација (**phase shift keying**, **PSK**)
- линијски појачавач (**line driver**)
- фреквенцијски мултиплекс (**frequency division multiplexing**, **FDM**)
- временски мултиплекс (**time division multiplexing**, **TDM**)
- топологија (**topology**)
- мрежа локалног подручја (**local area network**, **LAN**)
- мрежа пространог подручја (**wide area network**, **WAN**)
- од тачке до тачке (**point to point**)
- звезда (**star**)
- прстен (**ring**)
- магистрала (**bus**)
- **Ethernet**
- прстен са жетоном (**token ring**)
- протоколи за приступ (**access protocols**)
- **CSMA/CD** (**carrier sense multiple access with collision detection**)
- Прослеђивање жетона (**token passing**)
- етар (**ether**)
- носилац (**carrier**)
- судар, колизија (**collision**)
- редундантна циклична провера (**cyclic redundancy check**, **CRC**)
- жетон, белег (**token**)
- оквир, рам (**frame**)
- референтни модел за отворену међусобну комуникацију (**open systems interconnection reference model**, **OSI**)
- слој апликације (**application layer**)
- слој представљања (**presentation layer**)
- слој сесије (**session layer**)
- транспортни слој (**transport layer**)
- мрежни слој (**network layer**)
- слој повезивања (**data link layer**)
- физички слој (**physical layer**)
- **SNA** (**systems network architecture**)
- **datagram service**
- пакетска комутација (**packet switching**)
- рутирање (**routing**)
- репетитори (**repeaters**)
- мостови (**bridges**)
- рутери (**routers**)
- капије (**gateways**)
- интернет протоколи (**Internet protocols**, **IP**)
- протокол за управљањем преносом (**transmission control protocol**, **TCP**)
- протокол за пренос датотека (**file transfer**, **FTP**)
- протокол за повезивање на даљину (**telnet**)
- електронска пошта (**E-mail**)
- област (**domain**)
- **domain name system**, **DNS**
- **world wide web** (**WWW**)
- **intranet**

ПРИЛОГ - Ц
ОПИС ПОЈМОВА

А

Адресирање помоћу сегмент регистара (**segment register addressing**): Начин адресирања у којем се садржај сегмент регистра додаје на неку другу адресу и на тај начин се добија ефективна адреса.

Адресна магистрала (**address bus**): Магистрала дуж које се преноси адреса података из једне компоненте рачунара у другу, најчешће из процесора у меморију.

Адресни простор (**address space**): Све меморијске адресе које су на располагању корисничком програму.

Акумулатори (**accumulators, AC**): Регистри у **CPU** који се користе за чување резултата (и операнада) многих аритметичких и логичких операција.

Анализатор прекида, програм за обраду прекида (**interrupt handler**): Програм оперативног система који се активира ради обраде прекида. Његов задатак је да обради прекид и, најчешће, да врати контролу у прекинути програм.

Аналогни сигнал (**analog signal**): Континуални сигнал који се у току времена непрекидно мења по амплитуди. То је често нека измерена физичка величина: напон, температура и слично.

Апсолутни програм (**absolute program**): Програм на машинском језику са фиксним (нерелокативним) адресама, спреман да буде напуњен у меморију и извршен.

Апсолутно адресирање (**absolute addressing**): Начин адресирања којим се одређује редни број једне непроменљиве (нерелокативне) локације у меморији.

Аргумент (**argument**): Променљива вредност или адреса која се користи за пренос информација између два програма.

Архитектура рачунара (**computer architecture**): Компоненте рачунарског система, начин њиховог међусобног повезивања, скуп свих инструкција и начина адресирања.

Архитектура са фиксним блоковима (**Fixed-Block Architecture**): Један од начина организовања диска код којег се свака стаза дели на блокове или секторе фиксне величине, који се по потреби могу произвољно додељивати датотекама.

Аритметички померај (**arithmetic shift**): Померање битова удесно или улево, при чему се чува информација о знаку броја.

Аритметичко-логичка јединица (**Arithmetic And Logic Unit, ALU**): Део рачунара који извршава све аритметичке, логичке и операције померања.

ASCII (**American Standard Code For Information Interchange**): То је код који се користи за представљање слова, цифара и симбола помоћу низова од седам битова (или 8 битова у многим микрорачунарима).

Асемблер (**assembler**): Системски програм који преводи програм са симболичког машинског језика (асемблерски програм) у програм на машинском језику или релокативни код.

Асемблерски језик (**assembly language**): Види симболички машински језик.

Асоцијативна меморија (**associative memory**): Меморија код које се меморијска локација у

којој се налази податак проналази, не на основу адресе, већ на бази њеног садржаја, тј. на основу податка или дела податка који је у њој уписан.

Assert: Активирање или искључивање дигиталног сигнала.

Ажурирање (**update**): Мењање садржаја постојећих записа и података у датотеци.

Б

База (**base**): Основа позиционог бројног система.

Базно адресирање (**base register addressing**): Начин адресирања у коме се ефективна адреса израчунава сабирањем садржаја базног регистра и неке друге адресе.

Бајт (**byte**): Група од, најчешће, осам битова, којима се приступа као једној целини. Један бајт је најчешће најмања количина података која се може прочитати или уписати у меморију (најмања адресибина количина података).

Бајт мултиплексерски канали (**byte multiplexed channel**): Канал, (периферијски процесор) који контролише неколико спорих уређаја као што су принтери, плотери и слично.

Backspace: Враћање назад, на неки претходни запис у датотеци, уместо уобичајеног померања унапред.

Белег (**token**): Види жетон.

Безусловни скок (**unconditional jump**): Види безусловно гранање.

Безусловно гранање (**unconditional branch**): Инструкција која мења уобичајени секвенцијални (узастопни) редослед извршавања програма, и изазива скок на неку другу локацију. Скок се догађа сваки пут када се ова инструкција изврши, тј. нема тестирања услова.

Бинарне операције (**dyadic**): Аритметичке и логичке операције које се врше над два операнда.

Бинарни систем (**binary**): Бројни систем чија је основа два.

Бит (**bit**): Једна бинарна цифра.

Битовна мапа (**bit map**): Види мапа битова.

Бит највеће тежине, важности (**Most Significant Bit, MSB**): Крајње леви бит у бинарном броју.

Бит вертикалне парности (**vertical parity bit**): Бит за контролу парности једног карактера или бајта.

Блок (**block**): Термин који означава страницу или сегмент, а који се користи у систему за управљање меморијом.

Блок мултиплексорски канал (**block multiplexer channel**): Канал који надгледа више брзих уређаја са којих се подаци преносе у блоковима (од по неколико килобајта и већи).

Блок за контролу процеса (**Process Control Block, PCB**): Табела коју прави оперативни систем за сваки процес у рачунару. **PCB** обично садржи информације о стању процеса, његовој величини, приоритету итд..

Блокирано стање (**blocked state**): Види стање чекања.

Блокирање (**blocking**): Поступак груписања више логичких записа у један физички запис.

Bootstrap пунилац (**bootstrap loader**): Мали програм који се користи за пуњење у оперативну меморију.

Браћа (**siblings**): Чворови у структури података који имају истог родитеља (претходника).

Брисање (**clear**): Постављање бита или садржаја регистра на нулу.

Бројач инструкција: Види програмски бројач.

Бројач локација (**location counter**): Бројач који се користи у преводиоцима и асемблерима, а указује на количину потребне меморије при превођењу инструкција и података. Његов садржај увек показује на следећу локацију која је на располагању.

Бројач наредби: Види програмски бројач.

Бројеви са фиксном запетом (**fixed point number**): Број који има децималну тачку (запету) која се увек налази на истом месту.

Бројеви са непокретном запетом (**fixed point number**): Види бројеви у фиксном зарезу.

Бројеви са покретном запетом (**floating point numbers**): Рачунарски еквиваленнт за запис бројева где се одвојено памте експонент и значајне цифре броја (са знаком).

Бројни систем (**number system**): Скуп цифара и правила за дефинисање записа бројева.

Брзина преноса (**transfer rate**): Брзина преноса података **у** и **из** рачунара.

BCD код (**Binary Coded Decimal**): Систем кодирања за представљање било које декадне цифре као скупа четворобитних бинарних бројева.

В

Вектор (**vector**): Једнодимензионално поље података.

Векторски прекид (**vectored interrupt**): Поступак обраде прекида у којем свакој врсти прекида одговара посебна локација у меморији. На свакој од ових адреса налази се инструкција скока или адреса рутине за обраду те врсте прекида.

Векторски процесор (**array processor**): Помоћни процесор способан да изврши операције над читавим матрицама одједном, при чему то ради знатно брже од уобичајених рачунара.

Виртуелна меморија (**virtual storage system**): Управљање програмима који су сувише велики да стану у главну меморију. Базира се на подели таквих програма на блокове (странице или сегменте), при чему се у главној меморији држе само они блокови који се управо користе. Блокови који се тренутно не користе налазе се на секундарној или резервној меморији.

Вишепроцесорски рачнуар (**multiprocessor**): Рачунар који има два или више централних процесора.

Време додељивања: Види време придруживања.

Време придруживања-повезивања (**binding time**): Време када се одређују коначне адресе физичких локација у меморији.

Време тражења (**seek time**): Време потребно да се глава у диску помери са једне стазе на другу.

Временски мултиплекс (**Time Division Multiplexing, TMD**): Комуникациони протокол којим се време преноса дели на мале интервале (**slot**), и сваки од њих се додељују другом уређају који жели да врши пренос података.

Врста по врста (**row major**): Смештање поља података тако да се најпре сместе подаци из прве врсте, а за њима следе подаци из друге врсте и тако редом.

Г

Глава листе (**list head**): Специјална локација у меморији која садржи показивач на прву структуру у повезаној листи.

Главна меморија (**main memory, primary storage, operating memory**): Меморија у којој се налазе програми и подаци који се управо користе (обрађују).

Главни директоријум (**Master File Directory, MFD**): Директоријум који садржи информације о корисничким директоријумима у систему са више нивоа директоријума.

Гранање (**branch**): Измена секвенцијалног редоследа извршавања инструкције у програму.

Гранични регистар (**fence register**): Регистар који се користи за дељење меморије на две партиције: једну за корисника, а другу за оперативни систем.

Границе (**bounds**): Горња и доња граница важећих индекса који се могу користити у неком низу података.

Грешка парности (**parity error**): Ситуација која настаје када се при провери парности констатује погрешан број јединичних битова. Указује на грешку при смештању, читању или преносу података.

Групно кодирање (**Group-Coded Recording**): Метод кодирања података који користи групе битова за представљање друге групе битова. Често се користи пет битова за представљање четворобитних бројева.

Густина (**density**): Број карактера или бајтова који могу бити запамћени на јединици дужине меморијског простора као, на пример, број бајтова по инчу код магнетне траке.

Д

Database machine: Помоћни процесор који подржава специјалне операције за обраду база података и обавља их уместо централ–ног процесора.

Datagram service: Једноставан метод предаје порука на мрежу, али који не гарантује ни квалитет, ни редослед пријема, ни путању.

Датотека (**file**): Група записа који су логички повезани и који су заједно меморисани и обрађују се као целина.

Двоструко повезане листе (**doubly linked list**): Врста повезане листе у којој сваки члан (елемент) има показиваче и на следећи и на претходни елемент листе.

Декадни систем (**decimal**): Бројни систем базиран на основи десет.

Декрементирање (**decrement**): Смањење вредности неке променљиве.

Демодулација (**demodulation**): Конверзија аналогног сигнала са комуникационе линије поново у дигиталну форму тако да га може користити рачунар.

Дете (**children**): Види следбеник.

Дигитална кола (**digital circuit**): Кола која обављају аритметичке и логичке операције и функцију меморисања.

Дигиталне мреже (**digital circuit**): Види дигитална кола.

Дигитални сигнал (**digital signal**): Електронски сигнал који постоји у једном од два дискретна стања (супротно од континуално променљиве природе аналогних сигнала).

Димензија (**dimension**): Величина поља, тј. максимални број врста, колона итд.

Динамичка меморија (**dynamic storage**): Метод у коме се меморија додељује процесу само онда када је потребна за чување дела програма или података, и враћа се оперативном систему када више није неопходна.

Директива (**pseudoinstruction**): Види псеудо–инструкција.

Директан приступ меморији (**Direct Memory Access, DMA**): Техника за директан пренос података између брзе периферије (најчешће секундарне меморије) и главне меморије, без учешћа **CPU**–а.

Директно адресирање (**direct addressing**): Види апсолутно адресирање.

Диспечер (**dispatcher**): Део оперативног система који пребацује процес (задатак, **task**) из стања спремности у стање извођења, односно додељује му **CPU**.

Дисплеј са течним кристалом (**Liquid Crystal Display, LCD**): Течни кристал се налази у сендвичу између две стаклене (или пластичне) плоче. Кристални материјал рефлектује више светла када су молекули оријентисани у једном смеру, него када су оријентисани у неком другом смеру.

Дистрибуирани системи за обраду (**distributed processing systems**): Слабо спрегнути процесори и периферијске јединице који најчешће деле неке скупе ресурсе.

Додавање (**append**): Додавање у већ постојећи скуп. На пример, додавање записа у једну већ постојећу датотеку.

Други комплемент (**radix complement, two's complement**): Начин представљања негативних бројева. Добија се тако што се на први комплемент негативног броја дода јединица.

Дуплекс (**duplex**): Рачунарски систем са два процесора.

Е

EBCDIC код (**Extended Binary Coded Decimal Interchange Code**): Осмобитни код који је развио **IBM**, у којем се сваки знак кодира са осам битова.

Ефективна адреса (**effective address**): Стварна, физичка адреса која се користи за приступ жељеној локацији у меморији, а која се добија након што се изврши израчунавање зависно од начина адресирања (индексирање, индирекција итд.).

Ексклузивно ИЛИ (**exclusive OR**, **XOR**): Бинарна логичка операција која даје резултат тачан само ако је један једини улазни сигнал тачан.

Екстерна фрагментација (**external fragmentation**): Мали блокови примарне меморије који се не могу користити ни у једном сегменту или партицији.

Елемент (**element**): Једна јединка у пољу података.

Енкодер тастатуре (**keyboard encoder**): Логичка мрежа која конвертује сигнал који долази са тастатуре у сигнал који може да разуме рачунар.

Етернет (**ethernet**): Врста локалне мреже (**LAN**) која користи топологију магистрала и **CSMA/CD** протокол.

Ж

Жетон (**token**): Специјална порука у рачунарској мрежи која показује расположивост система.

З

Заштита датотека (**file protection**): Систем који се користи за спречавање недозвољеног приступа датотекама. Обично се остварује употребом шифре (**pasword**) која се мора унети пре него што се одобри приступ. Неки системи траже употребу више различитих шифара зависно од врсте приступа, на пример, једну за читање, а другу за модификацију датотеке.

Замка (**trap**): Слично прекиду, изазива скок на унапред дефинисану локацију када процес покуша да изврши неку недозвољену операцију.

Запис (**record**): Група елемената који су на одређени начин међусобом повезани и који се третирају као једна јединица.

Запис са девет стаза (**nine-track recording**): Подаци снимљени на магнетну траку у групама од девет битова.

Заставица (**flag**): Управљачки сигнал, често представљен само једним битом, а показује да се нека посебна ситуација десила у рачунарском систему (на пример, неки посебан услов: препуњење, негативан број, нула, итд.).

Затварање (**close**): Операција над датотеком која указује рачунарском систему да му та датотека неће више бити потребна.

Завршни чвор (**terminal node**): Чвор у структури података који није спојен ни са једним чвором на нижем нивоу (зове се још и лист).

И

Идентификатор процеса (**Process IDentifier, PID**): Јединствени број који се додељује сваком новом процесу.

ИЛИ операција (**inclusive OR**): Логичка операција која даје као резултат истину (тачан) ако је бар једна од улазних величина истинита. Резултат је нетачан (лаж) само ако су сви улазни подаци нетачни.

Имплицитно адресирање (**implied addressing**): Фиксна, непроменљива адреса податка. Најчешће се односи на један или више регистара (у **CPU**).

Имун на квар (**fault tolerant**): Специјализовани вишепроцесорски системи који имају редундоване компоненте тако да систем може да ради чак и након отказа неке од компонената.

Индекс (**index**): Табела која садржи локације података у датотеци, пољу или некој другој структури података.

Индекс (**subscript**): Вредност која указује на жељени елемент у пољу података.

Индекс регистар (**index register**): Регистри чији се садржај може додати некој адреси да би се добила нова ефективна адреса. Најчешће се користе за контролу програмских петљи.

Индексирано адресирање (**indexed addressing**): Начин адресирања у којем се садржај неког индекс регистра додаје некој адреси да би се добила коначна ефективна адреса.

Индиректно адресирање (**indirect addressing**): Начин адресирања у којем прва ефективна адреса садржи, уствари, поново адресу (адреса адресе података), а не податак.

Инкремент (**increment**): Повећање вредности неког броја или променљиве.

Инструкција (**instruction**): Група битова или бајтова који представљају код неке операције у рачунару.

Интеграција високог нивоа (**large-scale integration**, **LSI**): Интегрисана кола, чипови са 100 до 100 000 логичких кола.

Интеграција врло високог нивоа (**Very Large-Scale Integration, VLSI**): Више од 100 000 логичких кола интегрисаних у једном колу (чипу).

Интегрисано коло (**Integrated Circuit, IC**): Већи број функционално повезаних логичких кола направљених заједно на истом комаду силицијума (у истом чипу).

Interleaving: Хардверска техника која користи две или више одвојених меморијских јединица уређених тако да су узастопне адресе у разним јединицама што допушта истовремени приступ већем броју локација. Познат је принцип парних и непарних адреса које користе микрорачунари на бази **Intel**–ових процесора из серије 80x86.

Интерна фрагментација (**internal fragmentation**): Део меморијског простора који припада једном сегменту или партицији који се не користи, јер се не може доделити другом сегменту или партицији.

Интерпретер (**interpreter**): Језички преводилац који анализира програм по принципу "линија по линија". Свака линија се скенира, разлаже на саставне делове и извршава пре него што се пређе на следећу линију.

Инвертована листа (**inverted list**): Индекс. Може се логички креирати копирањем кључева и показивача из повезане листе у посебну структуру.

Инвертовање (**inversion**): Најпростија логичка операција позната као негација (НЕ, **NOT**). Инверзија мења истину у лаж и обрнуто. Исто што и негација, НЕ операција.

Искључиво ИЛИ: Види ексклузивно ИЛИ.

Избрисиви медијум (**erasable media**): Меморијски медиј који се може користити за чување података, а затим обрисати и поново употребити за складиштење података.

Излаз (**output**): Подаци и информације напуштају рачунарски систем.

Изоловани улаз-излаз (**isolated input/-output**): Улазно–излазне операције захтевају другачије инструкције од оних које се користе за приступ меморијским локацијама.

Изведени симбол (**nonterminal**): Симбол у граматици рачунарских језика који се састоји од једноставнијих, тј. може се делити, није коначан.

Извођење: Види стање извођења.

Изворни код, **програм** (**source code**): Програм на асемблеру или на вишем програмском језику који улази у језички преводилац.

J

Језгро (**kernel**): Део оперативног система који мора увек бити присутан у меморији (резидентан).

Језици ниског нивоа (**low-level languages**): Програмски језици врло блиски машинским језицима, тако да се једна инструкција овог језика по правилу преводи у једну инструкцију машинског језика.

Језици високог нивоа (**high-level languages**): Програмски језици који не зависе од хардвера рачунара ни од његовог машинског језика. Да би се програм написан на овим језицима извршио мора најпре да се преведе на машински језик.

Joy stick: Показивачки уређај који има вертикалну ручицу (палицу) која се може померати у произвољном правцу и смеру, и која може трансформисати то померање у електрични сигнал који се шаље у рачунар.

K

Канали (**channels**): Специјални рачунари намењени за обављање **I/O** операција. Често се називају периферијски процесори. Инструкције за њихов рад припадају такозваном канал–програму (**channel program**).

Карактер, знак (**character data**): Ненумерички податак који може бити слово, специјални знак, стринг.

Катодна цев (**Cathode Ray Tube, CRT**): Електронска цев видео дисплеја која користи загревање катоде да би се генерисали слободни електрони у електронском топу. Електрони (негативно наелектрисани) се крећу ка позитивно наелектрисаној површини екрана цеви. Површина екрана је покривена фосфоресцентним материјалом који светли када дође у додир са електронима са великом енергијом.

Кластер (**cluster**): Јединица диск меморије у личним рачунарима. Један кластер на флопи диску од 1,2 **MB** је 8 сектора тј. 4096 бајтова.

Код операције (**opcode**): део инструкције у машинском језику којим се кодира жељена операција.

Колизија (**collision**): Ситуација када два чвора мреже истовремено приступе комуникационој линији. То је такође ситуација која настаје када се две логичке адресе пресликавају у исту физичку адресу (метод остатка дељења).

Колона по колона (**column major**): Смештање матрица у облику низа колона.

Компајлер (**compiler**): Програм који преводи програм написан на вишем програмском језику у објектни модул на машинском језику.

Компилатор (**compiler**): Види компајлер.

Комплемент (**complement**): Начин приказивања негативних бројева. Комплементи позитивних бројева су исти као и сам тај број.

Коначни симболи (**terminal**): Основне, недељиве јединице које се користе за дефинисање синтаксе рачунарских језика.

Концепт рачунара са меморисаним програмом (**stored program concept**): У исту меморију су записани и инструкције и подаци.

Константа (**constant**): Вредност која се не мења током извршавања програма.

Контролер (**controller**): Рачунарски хардвер који садржи логичке мреже које управљају радом периферијских јединица.

Контролни блок система (**system control block**): Табела коју креира оперативни систем и садржи информације о свим процесима који се тренутно налазе у систему, врло често у облику показивача на одговарајуће редове и табеле.

Копирање (**copy**): Репродуковање података на новој локацији без промене оригинала.

Копроцесор (**coprocessor**): **CPU** пројектован за посебну намену и потпомаже главни **CPU**. На пример, микрорачунари на бази **Intel**–овог 80x86 користе копроцесоре серије 80x87 за извршавање сложених аритметичких операција.

Корен (**root**): Чвор на врху стабла структуре података. Нема претходника.

Кључ (**key**): Елементарни податак који омогућава брзу идентификацију, тј. проналажење жељеног записа или неке друге структуре података.

Л

Лабаво спрегнути системи (**loosely coupled system**): Вишепроцесорски системи у којима сваки од централних процесора има своју сопствену, приватну меморију, али процесори могу директно да комуницирају међу собом.

Лабела податак (**label data**): Стварна адреса неке лабеле у меморији.

Линијски појачавач (**line driver**): Уређај који појачава сигнал пре него што га пошаље на комуникациону линију. Линијски појачавачи омогућавају да компоненте рачунарског система буду знатно више удаљене него што би без њих то било могуће.

Листови (**leaves**): Чворови у структури стабла који немају следбенике, тј. чворове на нижем нивоу.

Логички подаци (**Boolean data**): Подаци који могу имати само вредност тачан (**true**) или нетачан (**false**).

Логички запис (**logical record**): Структура података која у себи садржи једну или више група сродних или различитих података, а који се могу обрађивати засебно или као целина. Више оваквих записа чине један физички запис који се као такав уписује или чита са секундарне меморије.

Логичко коло (**logic gate**): Основне логичке мреже које се користе у дигиталним колима,

а извршавају једну од основних логичких операција.

М

Машински језик (**machine language**): Једини језик који је непосредно разумљив дигиталним мрежама и може се извршити у рачунарском хардверу.

Магацин: Види стек.

Магистрала података (**data bus**): Скуп линија које се користе за пренос података између разних делова рачунарског система.

Магистрала (**bus**): Заједничке комуникационе линије које се користе за повезивање компонената рачунарског система. Информације које су постављене на магистралу доступне су било којој јединици која је повезана на магистралу.

Магнето-оптички диск (**magneto-optic disk**): Врста избрисиве оптичке меморије коју чини диск на који је нанесен материјал који ротира поларизовану светлост. Подаци се уписују под дејством магнетног поља када је диск загрејан ласером. Подаци се могу избрисати ако се диск поново загреје и изложи дејству магнетног поља.

Мантиса (**mantissa**): Значајне цифре броја у покретном зарезу.

Мапа битова, **бит мапа** (**bit map**): Структура података која се користи за управљање доделом меморије у датотечким системима и ретким низовима. **FAT** табела која се користи у **PC-DOS** је једна варијација мапе битова.

Мапа блокова (**block map**): Табела која показује на локације у примарној или секундарној меморији за сваки блок који користи неки програм.

Мапирање (**mapping**): Претварање имена променљивих у меморијске адресе.

Маска (**mask**): Низ битова који се користи за издвајање информација из података у меморији.

Маскирање (**masking**): Поступак спречавања прекида.

Матрица (**matrix**): Поље које има две или више димензија.

Међублоковски размак (**interblock gap**): Види празнина.

Међумеморија (**buffer**): Меморијско поље које се користи за привремено чување података док се они преносе из једне јединице или процеса у другу.

Међусклоп (**interface**): Место (уређај) коме рачунар и спољни свет приступају заједно. На пример, један улазно–излазни међусклоп контролише пренос података између **I/O** јединице и рачунара.

Међузаписни размак (**interrecord gap**): Види празнина.

Меморија са неизменљивим садржајем (**Read Only Memory**, **ROM**): Садржи програме који могу бити само прочитани, али не и модификовани. Садржај **ROM**–а остаје сачуван чак и након искључења напајања.

Меморија са случајним, произвољним приступом (**Random Access Memory, RAM**): Меморија каја садржи програме и податке који могу бити модификовани у било које време. Другим речима, меморија у коју се могу уписати и из које се могу прочитати подаци и програми.

Меморије са директним приступом (**Direct Access Storage Device, DASD**): Меморијски уређај који допушта кориснику да приступи једном запису, промени га и врати га назад на право место, а да при томе не мора да приступа другим записима. То се обично ради на бази кључа или задавањем његове тачне адресе.

Меморијски шемирани И/О простор (**memory mapped I/O**): За приступ I/O уређајима користе се исте инструкције као за приступ главној меморији, јер сваком I/O уређају одговара једна или више меморијских адреса.

Меморијски адресни регистар (**Memory Address Register, MAR**): Регистар који припада главној меморији и увек садржи адресу меморијске локације којој ће се приступити у следећем циклусу (наредном тренутку времена).

Меморијски бафер регистар (**Memory Buffer Register, MBR**): Види меморијски прихватни регистар.

Меморијски прихватни регистар, бафер (**Memory Buffer Register, MBR**): Регистар придружен јединици главне меморије, а који

привремено чува бајт или податак који се уписује у, или чита из меморије. Неки аутори га зову и **data latch**.

Миш (**mouse**): Показивачка улазна јединица која се покреће по равној површини, и има један или више тастера који се могу користити за избор акција и операција. Кретање миша по површини се претвара у кретање неког показивача на екрану.

Микроинструкција (**microinstruction**): Инструкције ниског нивоа које директно контролишу рад логичких мрежа у микропрограмираним рачунарима. Од њих се праве конвенционалне машинске инструкције или макроинструкције. Инструкција којом се кодира једна елементарна микрооперација. Уобичајене машинске инструкције обично представљају низ микроинструкција.

Микрокод (**microcode**): Скуп микроинструкција од којих се праве машинске и макроинструкције.

Микропроцесор (**microprocessor**): **CPU** пројектован и направљен као једно једино интегрисано коло.

Микропрограм (**microprogram**): Низ микроинструкција које координирају активности у логичим колима рачунара. Микропрогами могу да имплементирају функције или макроинструкције.

Микропрограмска меморија (**microprogram control store**): Део **CPU**–а у којем се налазе микроинструкције.

Микрорачунар (**microcomputer**): Рачунари који користе микропроцесор као свој **CPU**.

Минирачунар (**minicomputer**): Рачунари чија је меморијска реч дужине најмање 32 **bit-a**. Ови рачунари имају капацитет примарне меморије од најмање 32 **Mbyte**, а систем подржава од 16 до 256 радних станица

Мнемоник (**mnemonic**): Симболички код инструкција који се користи у симболичком машинском језику. Име кода "звучи као" име операције.

Модем (**modem**): Уређај који се користи за конвертовање дигиталних сигнала у аналогне и обрнуто. То омогућава да подаци буду пренети са једног рачунара на други посредством телефонске линије.

Модул за пуњење (**load module**): Рачунарски програм (врли сличан машинском) који може бити напуњен у меморију и извршен без додатних трансформација (превођења).

Модулација (**modulation**): Процес конверзије дигиталних сигнала у аналогне који се могу преносити преко телефонске линије. Овај процес обавља се помоћу модема.

Монитор (**monitors**): Први оперативни системи који су подржавали рутинске послове као што су, на пример, улазо/излазне функције.

Мониторски режим рада (**monitor mode**): Исто што и режим супервизора.

Мрежа локалног подручја, локална мрежа (**local area network, LAN**): Мала приватна мрежа која омогућава велику брзину повезивања већег броја процесора и периферија.

Мрежа пространог подручја (**wide area network**, **WAN**): Мрежа чији су чворови распрострањени на ширем географском подручју и повезују се употребом јавних или приватних комуникационих система који су под контролом државе.

Мрежа широког подручја (**wide area network**, **WAN**): Види мрежа пространог подручја.

Мултиплексер (**multiplexer, MUX**): Једноставно логичко коло које има два или више улаза и само један излаз. Овај уређај омогућава да се сигнали са два или више комуникационих уређаја пренесу преко само једне линије.

Мултиплексирање (**multiplexing**): Овај процес омогућава да два или више комуникационих уређаја деле (користе) исте комуникационе линије.

Мултипроцесирање (**multiprocessing**): Извршавање две или више инструкција истог програма у исто време.

Мултипрограмирање (**multiprogramming**): Истовремено постојање више од једног програма у меморији (али обрађује се само једна инструкција у једном интервалу времена).

Н

Наредба: Види инструкција.

НЕ (**NOT**): Најједноставније логичко коло. НЕ коло прихвата на улазу вредност нетачан (лаж, 0) и даје на излазу истиниту вредност (1).

Негација (**negation**): Конвертовање истините вредности у лаж или лажи у истину. То је најпростија логичка операција која се зове НЕ (**NOT**).

Немаскирани прекид (**nonmaskable interrupt**): Прекид који не може бити маскиран (онемогућен, спречен).

Непарна парност (**odd parity**): Систем парности где сваки низ битова мора имати непаран број јединица.

Непосредно адресирање (**immediate addressing**): Начин адресирања у којем се операнд налази на следећој локацији у меморији, или је у делу инструкције који следи иза кода операције.

Непомични програм (**nonrelocatable program**): Види нерелокативни програм.

Нерелокативни програм (**nonrelocatable program**): Програм у машинском језику који се увек мора извршавати са једне исте локације у меморији.

Ниске: Види стринг.

Низ података (**array**): Види поље података.

Носилац (**carrier**): Сигнал на комуникационој линији који је присутан чак и када на њој нема података.

Нула (**zero**): Специјална цифра која се користи за резервисање места у тежинским бројним системима.

О

Обележје (**label data**): Види лабела.

Објектни код (**object code**): Код на машинском нивоу али није извршан, а добија се као резултат превођења. Да би био извршан морају се обавити још неке модификације (повезивање, пуњење). Најчешће садржи релокативне адресе које пре извршавања морају бити одређене.

Октални број (**octal**): Бројни систем са основом 8.

Операција И (**AND**): Логичка операција чији је резултат истинит само ако су сви улази истинити.

Операнди (**operands**): Константе и/или адресе које се користе у програмирању на машинском језику. Величине над којима се врше аритметичке и логичке операције.

Оперативна меморија: Види главна меморија.

Оперативни систем (**operating system**): Низ програма који омогућавају коришћење и дељење рачунарских ресурса.

Описне инструкције: Види псеудоинструкције.

Опоравак (**recover**): Настављање програма или обнављање (довођење у претходно стање) програма и података након што је дошло до отказа система или грешке.

OSI (**Open Systems Interconnection reference model**): Модел рачунарског система пројектован да помогне комуникацију између различитих уређаја.

Остатак дељења (**hashing algorithm**): Поступак претварања кључа у адресу.

Отварање датотеке (**open**): Команда којом се казује оперативном систему које датотеке су потребне.

П

Пакетна обрада (**batch**): Група програма или делова процеса који се као једна јединица обрађују у рачунару. Неинтерактивна обрада.

Паралелна обрада (**parallel processing**): Истовремено извршавање две или више операција. Потребан је рачунар са више процесора.

Паралелни пренос (**parallel transmission**): Истовремени пренос свих битова једног бајта.

Парна парност (**even parity**): Систем парности који захтева да сваки податак (бајт или реч) има паран број јединица.

Парност (**parity**): Метод који се користи за квалитетно складиштење података. Додаје се

један нови допунски бит у бајт података када се он меморише, а његов садржај се проверава приликом читања.

Партиције (**partitions**): Део меморије додељен неком програму. Први рачунарски системи користили су партиције фиксне дужине, а код модерних рачунара величина партиције се динамички мења.

Периферијска јединица (**peripheral device**): То је јединица за улаз–излаз или секундарна меморија.

Периферијски процесор (**peripheral processor**): Специјални процесор намењен да извршава улазне–излазне операције. Види канал.

Планирање: Види управљање задацима.

Плазма екран (**plasma screen**): Сендвич јонизованог гаса између две стаклене плоче које садрже мрежу хоризонталних и вертикалних проводника. Кроз ове проводнике се пропушта мала струја, и на оним местима где струја протиче и кроз хоризонтални и кроз вертикални проводник, гас почиње да светли.

Подаци за контролу тока програма (**program control data**): Подаци које обично користи оперативни систем за обављање неких операција над задацима у рачунару (лабеле и случајни подаци).

Податак (**data**): Бројеви, имена, слова итд. који се обрађују помоћу рачунара.

Поддиректоријум (**subdirectory**): Датотека која се користи као директоријум за друге датотеке, а сама припада неком директоријуму на вишем нивоу.

Подканал (**subchannel**): Подјединица у бајт–мултиплексерском каналу. Користи се за пренос података са релативно спорих уређаја као што су терминали.

Показивач стека (**stack pointer, SP**): Регистар који сарджи адресу врха стека у меморији.

Показивачи (**pointer data**): Подаци који представљају локацију у меморији, најчешће адресу структуре података.

Полубајт (**nibble**): Појам који се користи у микрорачунарима и означава лева или десна четири бита у једном бајту.

Полусабирач (**half adder**): Мрежа за сабирање две бинарне цифре (два бита).

Померач (**shifter**): Логичка мрежа која реализује операцију померања.

Померање (**shift**): Померање битова у регистру улево или удесно. Битови који напуштају регистар су или изгубљени, или се привремено смештају у заставицу за пренос.

Помоћни процесор (**support processor**): Исто што и придружени процесор.

Pop: Враћање података са стека.

Поље лабеле (**label field**): Поље у инструкцији програмског језика које показује на локацију неке симболичке адресе.

Поље података (**array**): Структура података која представља скуп података истог типа (сви су цели бројеви, сви су записи и слично).

Post order: Метод кретања кроз стабло података у којем се почиње од корена, а затим се иде кроз крајње лево подстабло све до листова.

Потомак (**children**): Види следбеник.

Потврда лонгитудиналне парности (**longitudinal parity check**): Види потврда подужне парности.

Потврда подужне парности (**longitudinal parity check**): Потврда парности већег броја битова исте тежине при запису више бајтова на магнетној траци. У комбинацији са битом вертикалне парности може се користити за корекцију грешке.

Повезана листа (**linked list**): Структура података код које је најмање један саставни део показивач на другу структуру.

Повезивач (**linker**): Функција (програм) оперативног система која конвертује објектни код у модул за пуњење, тј. модул који може бити напуњен у меморију и извршен. Најчешће повезује програм са потпрограмима који ће му бити потребни у току извођења.

Позадински процесор (**back-end processor**): Помоћни процесор који извршава програме (задатке, **task**) који не захтевају директан контакт са корисником.

Позив потпрограма (**subroutine call**): Врста инструкције која мења ток извођења програма и чува садржај бројача наредби (адресу следеће наредбе) пре него што у њега упише почетну адресу новог програма–потпрограма. Програм се може вратити на следећу инструкцију (одмах иза позива потпрограма) извршавањем једне инструкције која поново у програмски бројач уписује сачувану адресу.

Позив супервизора (**Supervisor Call, SVC**): Захтев оперативном систему да пружи своје услуге.

Последњи ушао први изашао (**Lirst-In First-Out, LIFO**): Види магацин, стек.

Права приступа (**access rights**): Информација у систему директоријума која показује ко може да приступи датотеци или директоријуму, и шта може да ради са њима.

Празнина, размак (**gap**): Простор на меморијском медијуму који се не може користити за запис података. Они су неопходни на магнетним тракама да би се омогућило покретање и заустављање траке. На дисковима се користе да би рачунар могао да обради једну групу података пре него друга група стигне.

Предметни код (**object code**): Види објектни код, објектни програм.

Прекид (**interrupt**): Сигнал који показује да се десио неки изненадни догађај због којег се мора прекинути извршавање текућег програма и позива се неки специјални програм. Овај сигнал се јавља случајно, у тренутку времена и на месту у програму који нису унапред познати.

Преклапање (**overlay**): Разни сегменти истог програма са секундарне меморије копирају се на исте локације у примарној меморији. На овај начин могу се извршавати и програми који су већи од примарне меморије (која им је додељена).

Пренос (**carry**): Бит који настаје када је збир две цифре једнак или већи од основе бројног система. Такође се односи на бит који излази из регистра приликом извођења операција померања или ротације.

Препуњење (**overflow**): Када је резултат аритметичке операције сувише велики да би се сместио у један регистар.

Претходник: Види родитељ.

Претпражњење (**preemption**): Процедура управљања процесором која омогућава да процесор буде одузет чак и процесима који нису завршили са радом или нису блокирани.

Придружени, процесор за подршку (**attached support processor**): Независни централни процесор који помаже главном **CPU** у рачунарском систему да брже изврши неке специфичне операције. Он обично подржава комуникације, обраду база података или низова података.

Примарна меморија (**primary storage**): Исто што и главна меморија.

Привилеговано стање, режим рада (**privileged mode**): Исто што и режим супервизора (**supervisor mode**).

Привремени програм (**transient program**): Део оперативног система који не мора бити стално присутан у меморији. Програми се пребацују у меморију када су потребни и након што обаве обраду избацују се из ње.

Проширена меморија (**extended memory**): Врста спорије примарне меморије која се користи за прављење резервних копија програма и података.

Проширење имена датотеке (**file name extension**): Карактери који се додају на име датотеке и у општем случају се третирају као део имена. Они се најчешће користе као средство за опис типа датотеке. Тако, рецимо, текстуалне датотеке имају екстензију **TXT** или **DOC**.

Проблемски подаци (**problem data**): Променљиве и константе које користи програмер у свом, корисничком програму: цели бројеви, карктери итд..

Процедурално оријентисани језици (**procedure oriented languages**): Језици високог нивоа који допуштају програмеру да прави процедуре и алгоритме који решавају његов проблем.

Процес (**process**): Програм који је унет у рачунар и може да конкурише за ресурсе.

Програм (**program**): Логички самоконтролисани скуп рачунарских инструкција.

Програмски бројач (**Program Counter, PC**): Регистар који садржи адресу следеће машинске инструкције.

Програмски језик (**programming language**): Језик који се користи за писање инструкција које сачињавају рачунарски програм.

Програмски подаци: Види проблемски подаци.

Програмски управљани улаз/излаз (**program controlled I/O**): Потпуно раздвајање меморијских и улазно–излазних операција. Изоловани улаз–излаз.

Прослеђивање белега: Види пролазак жетона.

Прослеђивање жетона (**token passing**): Поступак којим чворови у мрежи преузимају и испитују жетон–белег (token,), извршавају потребну акцију и шаљу token следећем чвору.

Променљива (**variable**): Симболичко име додељено меморијској локацији. Користи се у свим језицима осим у машинском.

Пропусни опсег (**band width**): Опсег учестаности које са малим слабљењем пролазе кроз комуникациони медијум и могу се користити за пренос информација. Обично се користи као мера капацитета комуникационих система и у дигиталним системима се изражава у битовима у секунди.

Прорез (**slot**): Интервал вренена у временском мултиплексу придружен једном уређају који тада преноси своје податке.

Простор имена (**name space**): Скуп логичких имена променљивих које одређује корисник.

Простор кључева (**key space**): Скуп свих могућих кључева.

Простор локација (**location space**): Физичке адресе које се користе у програму.

Протоколи за приступ (**access protocols**): Процедуре пројектоване да регулишу коришћење комуникационих медија.

Прстен с белегом (**token ring**): Врста локалне мреже која користи топологију прстена и шаље токен од чвора до чвора, види пролазак жетона, жетон.

Први комплемент (**one's complement**): Начин представљања негативних бројева у бинарном бројном систему. Добија се тако што се свака бинарна цифра негативног броја (изузев знака) замењује њеном допуном до јединице, односно нуле се замењују јединицама а јединице нулама.

Први ушао први изашао (**First-In First-Out, FIFO**): Види ред.

Псеудоинструкција (**pseudoinstruction**): Инструкција у асемблеру која не одговара ни једној хардверској операцији (не извршава се), већ представља директиву (помаже) асемблеру–преводиоцу да он обави превођење, резервише меморијски простор и слично.

Пунилац (**loader**): Програм оперативног система који смешта (пуни) објектни модул или модул за пуњење у меморију (и тек онда програми постају извршни). Он уствари додељује стварне меморијске адресе инструкцијама и подацима.

Push: Смештање података на стек.

Пут (**path**): низ директоријума које треба прећи да би се дохватила датотека.

Путања: Види пут.

Пуњење (**load**): Пренос података из меморије у регистар.

Пуњење оперативног система (**booting**): Пуњење оперативног система у оперативну меморију.

Пуњење програма (**loading**): Пренос програма на тачно одређене локације у оперативној меморији.

Р

Рачунари са редукованим сетом инструкција (**Reduced Instruction Set Computer**, **RISC**): Наредбе су пројектоване тако да се извршавају великом брзином и имплементиране су директно у хардвер рачунара. За сваку инструкцију постоји логичка мрежа која је извршава. Овакви рачунари обично имају знатно мањи сет инструкција од микропрограмираних.

Распоређивање: Види управљање задацима.

Размак (**gap**): Види празнина.

Редундантна циклична провера (**Cyclic Redundancy Check, CRC**): Нумеричко поље које се користи за детекцију грешака у меморисаним подацима. Формира се при упису података на диск. Током операције читања, рачунар генерише нови **CRC** и упоређује га са старим (који је меморисан).

Режим супервизора (**supervisor mode**): Режим оперативног система или привилеговани режим рада који допушта приступ свим меморијским локацијама и употребу свих инструкција.

Реч (**word**): Група битова којима се приступа као једној целини. Обично се састоји од неколико бајтова.

Ред (**queue**): Група елемената или структура података организованих или обрађиваних у поретку **први-ушао-први-изашао**, **FIFO**.

Регистар (**register**): Меморијски уређај са великом брзином рада (кратко време приступа).

Регистар инструкција: Види регистар наредби.

Регистар наредби (**instruction register**): Регистар у којем се налази инструкција која управо треба да се изврши.

Регистар стања (**Status Register**, **SR**): Регистар чији поједини битови представљају заставице које показују да су задовољени неки услови. Слично као регистар стања процесора.

Регистар стања процесора (**processor status register, PSR**): Регистар који садржи информације о условима који могу утицати на текућу или будућу операцију (пренос, препуњење, негативан, итд.).

Регистар стања програма (**Program Status Word, PSW**): Види регистар стања процесора.

Регистри опште намене (**general purpose registers**): Регистри који се могу користити било као акумулатори, било као индекс регистри, базни или на други начин.

Резервисана реч (**reserved word**): Речи у програмском језику чије је значење унапред одређено и које се не могу користити у другом смислу.

Резервна копија (**backup**): Копија датотеке на неком другом медијуму за чување ради сигурности.

Резервна меморија (**backing store**): Меморијски уређај (најчешће скривени диск или проширена меморија) који се користи за држање програма и података који су привремено избачени из главне меморије.

Рекурзивне дефиниције (**recursive definition**): Дефиниција која се позива на саму себе.

Релативно адресирање (**relative addressing**): Адреса меморијске локације се задаје у односу на текућу вредност програмског бројача.

Релокација (**relocation**): Поступак померања података и инструкција у меморији. Инструкције које приступају меморији морају бити кориговане тако да показују на нове, тачне адресе.

Ресет (**reset**): Брисање садржаја бита, заставице, бајта или регистра.

Ретки низови (**sparse array**): Метод доделе меморијског простора само оним елементима поља који стварно садрже податке.

Rewind: Премотавање траке на почетак, или одлазак на почетак датотеке на траци или диску.

Rewrite: Уписивање измењеног, модификованог записа назад на диск или траку.

Родитељ (**parent node, predecessor**): Чвор у структури типа стабла који има једног или више следбеника (наследника, деце).

Ротација (**rotate**): Померање битова у регистру улево или удесно, тако да се битови који напуштају регистар поново уписују у исти регистар само на супротном крају.

Ротационо кашњење (**rotational delay**): Време потребно да се диск окрене до положаја у коме се подаци налазе испод уписно/читајуће главе.

Рутирање (**routing**): Поступак одређивања пута којим треба послати податке преко комуникационе линије.

С

Сабирница (**bus**): Види магистрала.

Садржајем адресибилна меморија (**content addressable memory**): Меморија којој се приступа задавањем целог или дела њеног садржаја уместо адресе. Види асоцијативна меморија.

Светлосна оловка (**light pen**): Показивачки уређај који када се ослони на површину терминала детектује електрично поље екрана и тако обезбеђује рачунару информацију која се може користити за одређивање места на екрану које је додирнуто.

Сегмент регистар (**segment register**): Регистар који садржи почетну адресу меморијског сегмента у рачунарима са **Intel** процесорима серије 8086.

Сегментација (**segmentation**): Систем са виртуелном меморијом који користи блокове променљиве дужине.

Сектор (**sector**): Део стазе на диску који садржи податке, а коме се може директно приступити. Сваки сектор на диску садржи исти број бајтова података.

Секундарне меморије (**secondary storage**): Дуготрајне меморије, великог капацитета и мале брзине, као што су оптички или магнетни диск или трака.

Селекторски канал (**selector channel**): Улазно–излазни канал који у једном интервалу времена преноси блок података само са једне, врло брзе, периферијске јединице.

Серијски пренос (**serial transmission**): Пренос података бит по бит.

Сесија (**session**): Један од слојева у моделу комуникационог система.

Сет (**set**): Постављање, упис јединице у бит или бајт. Супротно од ресета.

Сихрона обрада (**synchronous processing**): Обрада у којој **CPU** шаље једну **I/O** операцију на извршење у неки канал, а он сам чека да операција буде извршена. Када се **I/O** операција заврши **CPU** наставља са обрадом.

Симболички машински језик (**assembly language**): То је програмски језик који користи симболичке кодове (мнемонике) инструкција уместо бинарних кодова. Такође користи симболичка имена регистара и адреса меморијских локација. То је симболички језик ниског нивоа који је врло близак машинском језику. Једна инструкција или реченица асемблерског програма преводи се у једну инструкцију машинског језика.

Симплекс (**simplex**): Рачунар са само једним **CPU**.

Систем директоријума (**directory system**): Део оперативног система која организује и управља датотекама на секундарној меморији.

Ситакса (**syntax**): Правила која су дефинисана у неком језику.

Скалар (**scalar**): Најпростије структуре података. Један број или карактер.

Скривена меморија (**cache memory**): Брза међумеморија између регистра (у **CPU**) и главне меморије.

Скривени диск (**cache disk**): Брзи диск који се користи за привремено смештање програма и података када се они пребацују између главне и секундарне меморије.

Слабо спрегнути системи: Види лабаво спрегнути системи.

Следбеник (**succesor**): Део структуре стабла. Сви чворови повезани на исти чвор на вишем нивоу представљају његове следбенике, односно потомке.

Случајни подаци (**event data**): Врста података за контролу програма којима се бележи догађање неке посебне ситуације, на пример крај датотеке (**End-Of-File, EOF**).

Средње време приступа (**average seek time**): Просечно време потребно да се уписно–/читајућа глава на диску позиционира на жељену стазу (тј. цилиндар).

Стабло (**tree**): Структура података у којој сваки чвор може имати највише једног претходника, а може имати више следбеника. Корен стабла нема претходника.

Статичка додела меморије (**static storage allocation**): Додела меморије се врши само једном и не може се мењати.

Стање чекања (**wait state**): Када процесу треба ресурс који је недоступан, процес не

може да користи ни **CPU** и онда се каже да је он у блокираном стању.

Стање излаза (**output state**): Стање у којем процесу више није потребан **CPU**, али још увек чека да се одштампају или меморишу резултати.

Стање извођења: Види стање извршавања.

Стање извршавања (**run state**): Стање процеса у којем он управља процесором и када се извршавају његове инструкције.

Стање спремности (**ready state**): Стање у којем процес има све што му је потребно да би могао да се извршава у централном процесору, тј. да добије **CPU**.

Стање суспензије (**suspend**): Избацивање процеса из конкуренције за добијање ресурса. Избацивање из реда спремности.

Стање улаза (**input state**): Стање процеса у коме је он препознат од стране оперативног система, али још увек му нису додељени ресурси и не може да се такмичи за добијање централног процесора.

Стаза (**track**): Логички дефинисано кружно поље на диску. Део површине диска на истом полупречнику.

Стек (**stack**): Структура података која садржи податке који се морају користити (обрађивати) у редоследу **последњи ушао први изашао**, **LIFO**. Подаци се увек додају и узимају са исте стране.

Страничење (**paging**): Систем са виртуелном меморијом у којем су инструкције и подаци подељени у више блокова једнаке дужине.

Стринг (**string**): Групе или поља ненумеричких података или карактера.

Структура података (**data structure**): Скуп логички повезаних података. То су сложени подаци састављени од компоненти које нису истог типа и не морају бити скаларне.

Структурирани подаци (**structure data**): Груписање различитих типова података у једну логичку целину, као, на пример, у повезаним листама.

Scan code: Код који показује који тастер је притиснут. Значење скан кода одређује се помоћу програма.

Т

Табела аргумената (**argument table**): То је, на пример, структура која садржи адресе сваког аргумента који се из програма преноси у потпрограм и обрнуто.

Табела истинитости (**truth table**): Табела која се користи за приказивање резултата неке логичке операције.

Табела сегмената (**segment table**): Табела блокова у меморији којом се мапирају адресе сегмената у стварној меморији.

Табела симбола (**symbol table**): Табела која се креира за време превођења, а садржи симболичке адресе (имена) и њихове локације у преведеном програму.

Такт сигнал (**clock signal**): Сигнал којим се синхронизује (усаглашава) рад свих дигиталних мрежа у рачунару.

Таск (**task**): Програм у фази извршавања. Исто што и процес и задатак.

Текућа трака (**pipeline**): Техника којом се повећава брзина рада рачунара дељењем сложеног рачунског процеса у низ подпроцеса.

Текући директоријум (**current directory**): Директоријум који ће бити коришћен ако се другачије експлицитно не нареди.

Топологија (**topology**): Конфигурација комуникационе мреже. Начин повезивања чворова.

У

Улаз (**input**): Било који унос података у рачунарски систем.

Улазна тачка (**entry point**): Симболичка адреса дефинисана у програму која показује на адресу на коју се мора предати управљање. То је најчешће прва адреса неког програма или потпрограма.

Унарна (**monadic**): Аритметичка или логичка операција која захтева само један операнд.

Упис (**write**): Додавање новог записа.

Управљачка јединица (**Control Unit**, **CU**): Део **CPU** који генерише управљачке сигнале

који су неопходни за координацију и времен–ско вођење свих операција у рачунару.

Управљачкаа магистрала (**control bus**): Маг–истрала која се користи за пренос сигнала којима централни процесор управља радом свих делова рачунарског система.

Управљачки сигнал (**control signal**): Сигнал који се користи за издвајање само оних дигиталних мрежа и склопова у рачунару који учествују у извршењу неке операције.

Управљање задацима (**scheduler**): Део оп–ративног система који је одговоран за довођење програма из стања улаза у стања спремности.

Условни скок (**conditional jump**): Види ус–ловно гранање.

Условно гранање (**conditional branch**): Инс–трукција која у одређеним условима, може изменити уобичајени редослед узастопног извршавања инструкција програма, гранањем (скоком) на неку локацију након обављеног теста.

Успутно адресирање (**implied addressing**): Види имплицитно адресирање.

Узастопна меморија (**consecutive storage**): Начин смештања низа података у главну или секундарну меморију на узастопне локације.

Ф

Фактор блокирања (**blocking factor**): Број логичких записа који су груписани заједно (садржани у једном физичком запису).

Фаза извршавања (**execution cycle**): Вре–менски период у коме се декодира инс–трукција и извршава се назначена операција. Ова фаза следи фазу прибављања.

Фаза прибављања (**fetch cycle**): Поступак одређивања локације инструкције у меморији и њеног копирања у централни процесор ради извршавања.

File Allocation Table, FAT: Табела у **PC–DOS** оперативном систему која се користи за доделу простора на диску датотекама. Ова **FAT** табела има по један бајт за сваки блок (сектор) на диску.

Фирмвер (**firmware**): Програм записан у **ROM** меморији.

Физичка адреса (**physical address**): Види ефективна адреса.

Физички запис (**physical record**): То је стварни запис података на јединици се–кундарне меморије.

Flip-flop: Логичко коло које се користи за памћење једног бита информације.

Folding: Извршавање неке операције за време превођења програма (а не за време извршавања). На пример, инструкција **x=2+3** може се за време превођења заменити ин–струкцијом **x=5** јер се константе не мењају за време извршавања.

Фонт (**font**): Скуп карактера који имају исти стил или формат.

Форматирање (**formatting**): Поступак уписа специјалних контролних информација на диск да би га припремили за коришћење под датим оперативним системом.

Фреквенцијски мултиплекс (**Frequency Divi-sion Multiplexing, FDM**): Комуникациони ме–тод у коме се расположиво фреквентно по–дручје дели на фреквенцијске опсеге, и сваки опсег се додељује другом уређају. На тај начин се постиже једновремени пренос по–датака између више уређаја дуж једне кому–никационе линије.

Х

Хексадецимални број (**hexadecimal**): Бројни систем са основом 16.

Ц

CD-ROM (**Compact Disk Read-Only Mem-ory**): Оптички меморијски медиј великог ка–пацитета базиран на истој технологији која је примењивана за прављење музичких **CD.**

Цели бројеви (**integers**): Сви бројеви који немају децималну тачку или децимални (разломљени) део.

Централна јединица за обраду (**CPU**): Види централни процесор.

Централни процесор (**Central Processing Unit, CPU**): Дигитална кола која обављају аритметичке, логичке и управљачке операције у рачунарском систему.

Цифре (**digits**): Низ јединствених симбола који се користе у неком бројном систему. Број цифара у тежинском бројном систему једнак је основи система.

Циклична листа (**circular list**): Врста повезане листе код које последњи елемент листе има показивач на први елемент листе.

Циклична провера (**Cyclic Check, CC**): Види редундантна циклична провера

Циклични ред (**circular queue**): Врста структуре података типа **ред** у којем је први елемент реда логички следбеник последњег елемента

Циклични пренос (**end-around carry**): При одузимању бројева у првом комплементу, ако се појави бит преноса, онда се он додаје на бит најмање тежине.

Цилиндар (**cylinder**): Скуп стаза на диску којима се може приступити без померања главе.

CSMA/CD (**Carrier Sense Multiple Access with Collision Detection**): Протокол који се користи у неким **LAN** мрежама за контролу употребе комуникационе линије, и да онемогући да више од једног чвора користи линију у исто време.

Ч

Чвор (**node**): Потенцијална тачка гранања у структури података типа стабла.

Чвор (**node**): Уређај који припада некој мрежи.

Чврсто спрегнути систем (**tightly coupled system**): Систем у коме неколико централних процесора дели заједничку меморију.

Челни процесор (**front-end processor**): Помоћни процесор логички смештен између корисника и главног **CPU** (на пример, канали, комуникациони процесори).

Читање (**read**): Поступак или наредба која се користи за премештање података из секундарне у примарну меморију.

Чишћење меморије (**garbage collection**): Избацивање непотребних објеката из меморије.