

МИКРОРАЧУНАРИ

ВЕРИЦА ВАСИЉЕВИЋ
БОРИСЛАВ ХАЏИБАБИЋ
БРАНИСЛАВ ПАВИЋ
ВЛАДИМИР ТАДИЋ

Висока школа електротехнике и рачунарства
струковних студија
Београд, 2009.

Верица Васиљевић
Борислав Хацибабић
Бранислав Павић
Владимир Тадић

Микрорачунари

Висока школа електротехнике и рачунарства
струковних студија
Београд, 2009.

др Верица Васиљевић, мр Борислав Хаџибабић
Бранислав Павић, Владимир Тадић

МИКРОРАЧУНАРИ
друго измењено издање

Рецензенти
др Миленко Цветиновић, др Александар Жорић

Издавач
Висока школа електротехнике и рачунарства
струковних студија у Београду

За издавача
др Драгољуб Мартиновић

Лектор
Анђелка Ковачевић

Техничка обрада
др Верица Васиљевић, Мирко Ступар

Дизајн
Мирко Ступар

Штампа
Грид, Београд

Тираж
120

ISBN
978-86-7982-043-3

Наставно веће Високе школе електротехнике и рачунарства
струковних студија у Београду, на седници одржаној 30.6.2009. године,
одобрило је издавање и примену овог уџбеника у настави.

Предговор

Ова књига замишљена је као уџбеник који треба да пружи практично применљива знања из области микрорачунара студентима Високе школе електротехнике и рачунарства у Београду, али она може да буде коришћена и као уџбеник за заинтересоване за микрорачунаре уопште а посебно за потенцијалне кориснике широко распрострањене фамилије коју је развила Motorola под ознаком MC68HC11. Књига је намењена читаоцима са оскудним предзнањем из дигиталне електронике и поступно их уводи преко елементарних приказа од основних логичких кола, меморијских компонената до микропроцесора и микроконтролера, од њиховог повезивања са спољашњим светом који треба да надгледају, или њиме управљају, до програмирања којим се задата функција реализује.

У првом поглављу стога се дефинишу логичка кола која реализују основне функције Булове алгебре, најчешће коришћени меморијски елементи (флип-флопови), бројачи и регистри. Такође су приказана и логичка кола средњег степена интеграције (декодери, мултиплексери, аритметичка кола) која представљају како неопходне елементе, који се додају ради повезивања на сензорске, приказивачке или актуаторске елементе система или као проширење функционалности компонената високог степена интеграције (микропроцесора, микроконтролера), тако и као функционалне целине које омогућавају разумевање рада кола високог степена интеграције.

Основни концепт рачунара је да се управљачки алгоритам (програм) и потребни параметри запамте у одговарајућим меморијским елементима и затим у предвиђено време изврше над актуелним подацима. Стога су у другом поглављу приказане разне врсте меморијских модула и различити начини њиховог комбиновања ради постизања жељеног капацитета.

Појам рачунара, дефинисање основних појмова везаних за рад, основни модел микрорачунара, елементи структуре, начин извршавања наредби предмет су нешто дужег трећег поглавља. Ту се на упрошћеном

моделу приказује извршавање наредби у бинарном облику запамћених у меморији на упрошћеном моделу архитектуре процесора из фамилије Motorola MC68HC11. Дат је бинарни приказ наредби и начин њиховог извршавања на нивоу обраде у аритметичко-логичкој јединици и трансферу садржаја између основних регистара. Дефинисан је бројач наредби као најчешће коришћени механизам за секвенцирање наредби које треба извршити. Уведени су и појмови машинског, асемблерског и виших програмских језика. Ово поглавље представља елементарно упознавање читаоца са извршавањем програма од стране процесорске јединице

Поглавље четири детаљније представља микрорачунар, неопходне функционалности за рад и начин за повезивање са спољним светом на дигиталном нивоу: са меморијским модулима, улазним и излазним елементима и њихово временско вођење. Објашњено је повезивање меморијских компонената и улазних или излазних модула потребних да задовоље потребе захтеване примене.

Пето поглавље посвећено је процесорској јединици MC68HC11, њеним архитектурним елементима и њиховом функционисању током извршавања неких битних наредби. Уведен је појам позива потпрограма, најчешћи механизам чувања адресе повратка преко магацинске меморије (стека) као и механизам прекида програма (interrupt) којим се захтева опслуживање хардверским позивањем одговарајућег потпрограма.

После принципског упознавања са основним елементима процесора у шестом поглављу објашњено је програмирање MC68HC11: начини адресирања, све неопходне наредбе за управљање радом и манипулације подацима. Кроз велики број једноставних примера илустровано је извршавање свих типова наредби и дате су типичне кратке секвенце у којима се користе.

У следећем поглављу елабориран је начин рада са улазно излазним уређајима од директног, програмски контролисаног, преко прекида захтеваних од периферије или придруженог часовника до директног приступа меморији. Такође је приказана и спрега са аналогним системима преко одговарајућих конвертора.

Практична питања везивања периферијских уређаја предмет су осмог поглавља: од галванског раздвајања, преко синхроног и асинхроног серијског интерфејса и паралелног интерфејса уопштено до

објашњења специјализованих периферијских компонената и њиховог програмирања.

У претходним поглављима коришћен је упрошћени модел MC68HC11 са циљем да се читаоцу не замагли основна слика претераним детаљима. Стога девето поглавље представља практично референтни опис MC68HC11 са свим потребним детаљима за коришћење уграђених периферијских модула.

Поглавље десет представља сублимацију претходно стечених знања кроз више комплетних примера којима се илуструје решавање типичних проблема који се решавају микроконтролерима. Док се у претходним поглављима објашњавају концепти и основни елементи микроконтролера и уређаја базираних на њима у овом се дају комплетни примери чије разумевање новом стручњаку даје сигурност да може сам да приступи решавању практичних проблема коришћењем микроконтролера.

Поглавље једанаест представља комплетну референцу за инструкцијски скуп MC68HC11 погодну за практично коришћење.

Читава књига обилује примерима који значајно доприносе разумевању материје онеме ко се са тим први пут сусреће. Смањењем њиховог броја обим књиге могао би бити мањи, уз ризик да разумљивост за почетника у овој области буде смањена. Већи број питања на крају сваког поглавља доприноси да читалац, одговарајући на њих, провери своје разумевање и у случају потребе обрати додатну пажњу на поједине делове. Вредност слика у разумевању процеса је општепозната и стога велики број илустрација у овој књизи томе значајно доприноси.

Београд, септембар 2009. године.

Рецензент
др Миленко Цветиновић

Садржај

1. ДИГИТАЛНА КОЛА	1
Паралелни и серијски пренос	2
Логичка кола	3
Логички еквиваленти	5
Фамилије интегрисаних логичких кола	6
Логика три стања	7
Флип-флоп	9
Базично флип - флоп коло	10
Такт сигнали	11
Окидачки флип-флопови	13
Синхрони и асинхрони улази флип-флопова	16
Бинарни бројач	17
Флип-флоп регистри	20
Регистри	23
Повезивање на заједничкој магистрали	36
Декодери	31
Кодери	34
Мултиплексер (селектор података)	34
Демултиплексери (дистрибутери података)	36
Аритметичка кола	36
Питања и задаци	38
 2. МЕМОРИЈСКИ МОДУЛИ	 39
Терминологија која се користи код меморија	41
Генерални начин рада са меморијом	44
Технологија полупроводничких меморија	46
ROM меморије	48
Архитектура ROM меморије	50
Временски дијаграм читавања ROM меморије	53
Врсте интегрисаних ROM меморија	53
Полупроводничка RAM меморија	55
Проширивање дужине речи и капацитета	59
Питања и задаци	62

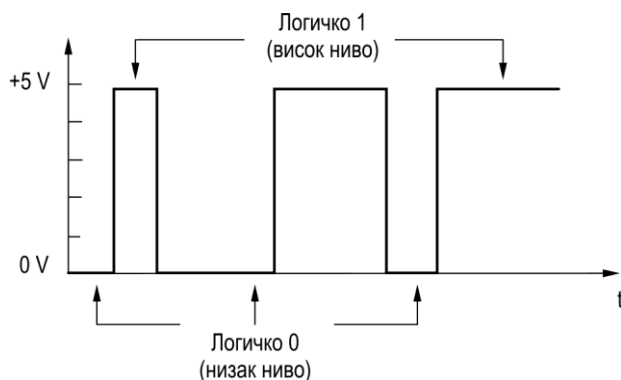
3. УВОД У РАЧУНАРЕ	63
Шта рачунар може да ради?	63
Како рачунари размишљају?	64
Колико типова рачунара постоји?	66
Основна структура рачунара	67
Микропроцесор, микроконтролер и процесор	71
Рачунарска реч	72
Бинарна реч података	73
Кодирани подаци	74
Инструкцијска реч	76
Процесор MC68HC11 - поједностављена верзија	83
Регистри процесора MC68HC11	85
Извршавање програма	87
Инструкције гранања и скока	94
Хардвер, софтвер и фирмвер	97
Програмски језици - машински језик	97
Асемблерски језик	100
Питања и задаци	105
 4. СТРУКТУРА И НАЧИН РАДА МИКРОРАЧУНАРА	 107
Елементи микрорачунара	107
Типична примена микрорачунара	111
Организација микрорачунара	112
Временски дијаграм при уписивању и читавању	120
Активности за време извршавања инструкција	128
Адресни простор микрорачунара	134
Меморијски модули	139
Декодовање адресе	142
Комплетан пример декодовања у микрорачунарима	144
Повезивање магистрала процесора	158
Меморијски мапирани и изоловани улазно/излазни модули	160
Питања и задаци	162

5. ПРОЦЕСОР	165
Процесор MC68HC11	166
Управљачко/временска логика	166
Регистри	172
Аритметичко/логичка јединица	186
Питања и задаци	193
 6. ПРОГРАМИРАЊЕ ПРОЦЕСОРА MC68HC11	 195
Програмски модел процесора MC68HC11	197
Начин адресирања процесора MC68HC11	198
Инструкцијски скуп	199
Начин представљања програма - листинг	212
Дијаграми тока и структурно програмирање	212
Инструкције акумулатор/меморија	215
Инструкције за рад са регистром услова	252
Инструкције које управљају извршавањем програма	272
Примена индексног адресирања	273
Програмски контролисани временски интервали	277
Програми за кашњење	283
Процес развијања софтвера	287
Питања и задаци	291
 7. УЛАЗНО/ИЗЛАЗНИ НАЧИН РАДА	 293
Основни појмови	293
Примери улазно/излазног начина рада	293
Различите врсте улазно/излазног преноса	296
Безусловни улазно/излазни пренос	297
Условни улазно/излазни пренос	302
Пренос који иницира улазно/излазни уређај - прекид	310
Појава прекида за време извршавања потпрограма за обраду прекида	320
Рад са више улазно/излазних уређаја	322
Пренос који иницира периферија	326
Програмабилни часовник	334
Питања и задаци	344

8. ВЕЗИВАЊЕ УЛАЗНО/ИЗЛАЗНИХ УРЕЂАЈА	347
Практична разматрања везана за интерфејсе	347
Синхронизација у преносу података	354
Паралелно/серијски интерфејс	361
Синхрона серијска комуникација	381
Стандард за серијску комуникацију EIA-232-F	384
Паралелно/паралелни интерфејс	390
Питања и задаци	402
9. МИКРОКОНТРОЛЕР MC68HC11	403
Улазно/излазни портови микроконтролера MC68HC11	405
Паралелни улазно/излазни порт	407
Серијски комуникациони интерфејс	409
Синхрони периферијски интерфејс	415
Аналогно/дигитални конвертори	418
Систем часовника микроконтролера MC68HC11	421
Питања и задаци	424
10. ПРИМЕНА МИКРОКОНТРОЛЕРА СЕРИЈЕ MC68HC11	425
Повезивање екрана са три седмосегментна показивача и микроконтролера	425
Управљање једносмерним мотором	429
Полупроводнички ветроказ	432
Једна реализација управљања трофазним асинхроним мотором	447
11. КОМПЛЕТАН СКУП ИНСТРУКЦИЈА MC68HC11	467

1. Дигитална кола

Кола у дигиталним системима пројектована су за рад са сигнаlima који су дигиталне природе, што значи да ови сигнали могу да имају само две могуће вредности у датом тренутку. Слика 1.1 показује типичне дигиталне облике који се најчешће крећу између 0 V (волти) и +5V. Овим напонским нивоима додељене су бинарне вредности 0 и 1, респективно. Ове 0 и 1 репрезентације називају се логички нивои. Иако су логички нивои на слици 1.1 приказани тачно као 0V и +5V, у реалним системима сваки логички ниво ће представљати опсег напона. На пример, логичка 0 (често названа низак ниво) може да буде било који напон између 0V и +0,8V, а логичко 1 (названо висок ниво) може да варира од +2V до +5V. Ово је опсег логичких нивоа за скуп (фамилију) интегрисаних кола са транзистор- транзистор логиком (TTL¹ кола).



Слика 1.1 Типичан дигитални сигнал

У дигиталним рачунарима дигитални напонски нивои могу да представљају различите ствари. Један дигитални напонски сигнал може да

¹ Transistor Transistor Logic

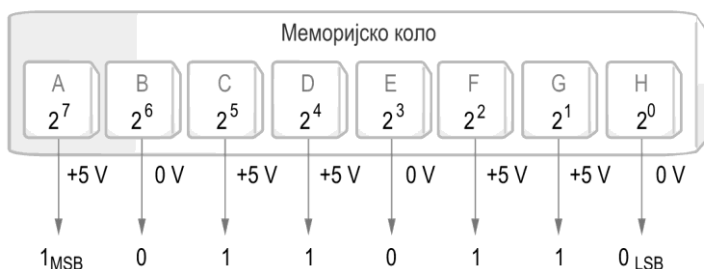
1. Дигитална кола

репрезентује један бит бинарног броја, један бит неког бинарног кода (нпр. BCD¹, ASCII²), логички или управљачки ниво сигнала, статус неких догађаја, или неколико других могућности. Независно од тога шта дигитални или логички сигнал репрезентује, кола која раде са овим сигналимa називају се дигитална логичка кола. У овом поглављу биће укратко објашњена сва логичка кола потребна за разумевање начина рада рачунара.

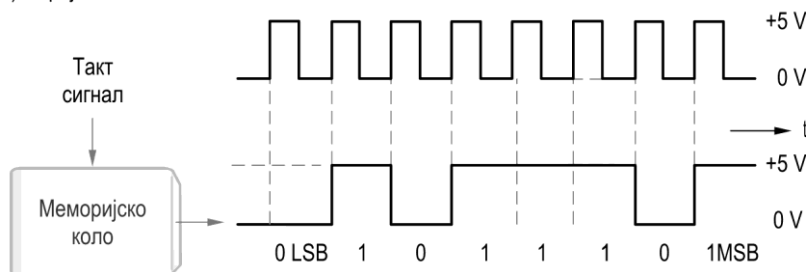
1.1 Паралелни и серијски пренос

Генерално је потребно слати сигнале који представљају потпуне бинарне бројеве, који се састоје од одговарајућег броја битава, из једног дела система у други. Постоје у основи два начина како се то може урадити: паралелни пренос и серијски пренос.

а) Паралелан излаз



б) Серијски излаз



Слика 1.2 а) паралелни и б) серијски пренос бројева³

¹ Binary Coded Decimal

² American Standard Code for Information Interchange

³ Бит најмање тежине LSB (Least Significant Bit); бит највеће тежине MSB (Most Significant Bit).

У паралелној репрезентацији или преносу бинарних бројева (или кодова), сваки бит је изведен из посебног излаза кола и послат преко посебне линије. Слика 1.2а илуструје представљање 8-битног броја.

Сваки излаз кола представља једну бинарну цифру (бит) и може да буде или 0V (бинарно 0) или 5V (бинарно 1). Осам излаза кола појављује се истовремено, тако да је у сваком тренутку комплетан бинарни број доступан на излазима. На слици 1.2а је представљен бинарни број 10110110_2 .

У серијском преносу, који је илустрован на слици 1.2б, за пренос бинарног броја користи се само једна сигнална излазна линија. На излазну линију шаљу се различите битови један по један. Лако је уочити да је потребна нека врста тачно дефинисаних тренутака времена¹ да би се могла направити разлика између различитих битова. На слици се користи такт сигнал да би обезбедио секвенционирање. Такт сигнал је важан део већине дигиталних система и користи се да би се дефинисали тачни тренуци времена за извршавање операција. У серијском преносу, сваки пут када се такт импулс појави излазни сигнал се мења у следећи бит бинарног броја. На слици 1.2б, излазна секвенца су битови 10111010_2 (уочимо да време расте слева надесно). Стварни бинарни број који се преноси је 01011101_2 , ако се бит најмање тежине шаље први.

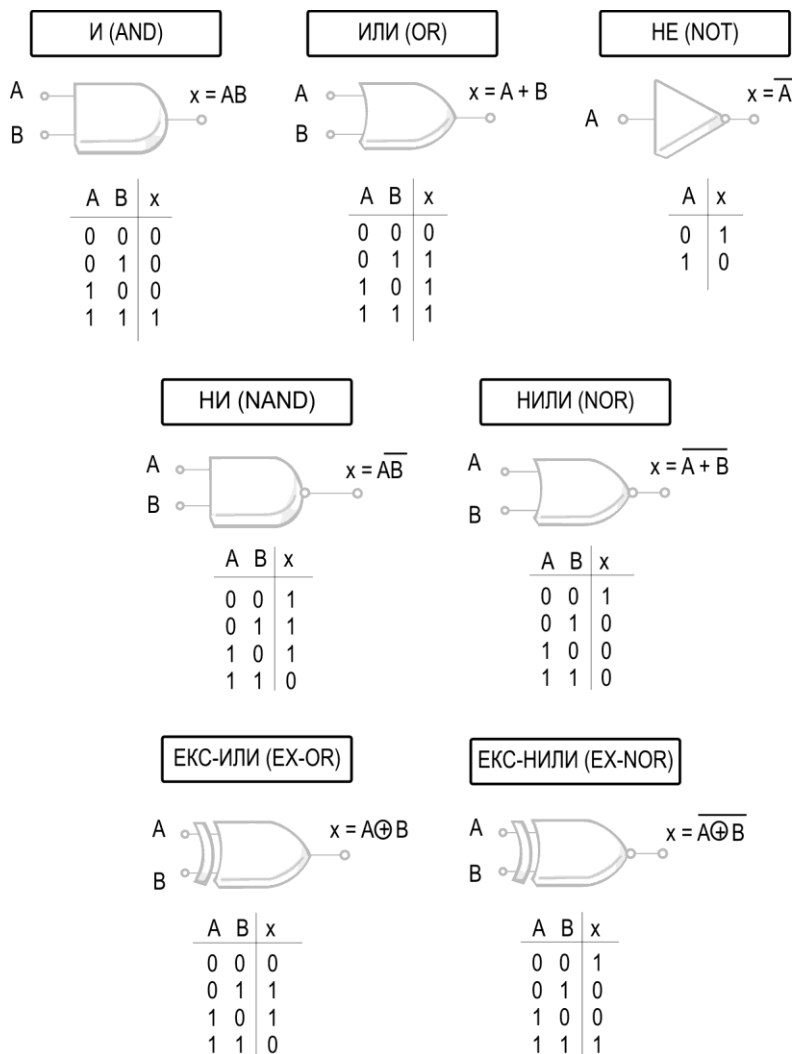
1.2 ЛОГИЧКА КОЛА

Логичка кола су дигитална кола која имају један или више логичких улаза и један или више логичких излаза. Логичка стања излаза одређена су логичким стањима улаза и функцијом логичког кола. Слика 1.3 показује свако од основних логичких кола, њихову математичку репрезентацију и излазни ниво у зависности од различитих комбинација улазног нивоа.

ИЛИ кола дају на излазу логичко 1 када је било који улаз на нивоу 1. Његова математичка репрезентација је $G=A+B$, где знак „+” представља ИЛИ операцију, а не обично сабирање. За троулазно ИЛИ коло, било би $G=A+B+C$ и тако даље, за више улаза. И кола раде на начин да ће излаз бити на логичком 1 само када су сви улази на 1. Математичка репрезентација за двоулазна И кола је $G=AB$, исто као обично множење. За три улаза биће $G=ABC$ и тако даље, за више улаза.

¹ Timing

1. Дигитална кола



Слика 1.3 Символи и функционалне табеле за основна логичка кола

НЕ кола имају само један улаз. Често се називају инвертори и дају излаз чији је логички ниво увек супротан од улазног логичког нивоа.

Математички израз за ову операцију је $G = \bar{A}$. Надвучена ознака, у овој књизи, увек указује на операцију инверзије (комплементирање) (НЕ).

НИ коло комбинује И и НЕ операције, тако да ће излаз бити 0 само када су сви улази 1. Његова математичка представа је $G=\overline{AB}$, и она указује да су улази А и В прво у међусобној И операцији, а затим да је резултат инвертован. Тако НИ кола увек дају излаз који је инверзан (супротан) од И кола.

НИЛИ коло комбинује ИЛИ и НЕ операције тако да ће излаз бити 0 када је било који улаз 1. Његова математичка репрезентација је $G=\overline{A+B}$, што указује да се прво изврши ИЛИ операција над улазима А и В, а затим резултат инвертује. НИЛИ коло увек даје излазни логички ниво који је инверзан од ИЛИ кола.

ЕКСКЛУЗИВНО ИЛИ коло даје 1 на излазу само када су улази на различитим логичким нивоима. ЕКС ИЛИ коло увек има два улаза, и математичка репрезентација његовог излаза је $G=A \oplus B$

ЕКСКЛУЗИВНО НИЛИ коло је инверзно од ЕКС ИЛИ кола. Оно даје 1 на излазу само када су улази на истом логичком нивоу. Математичка репрезентација ЕКС НИЛИ излаза је $G=A \oplus B$.

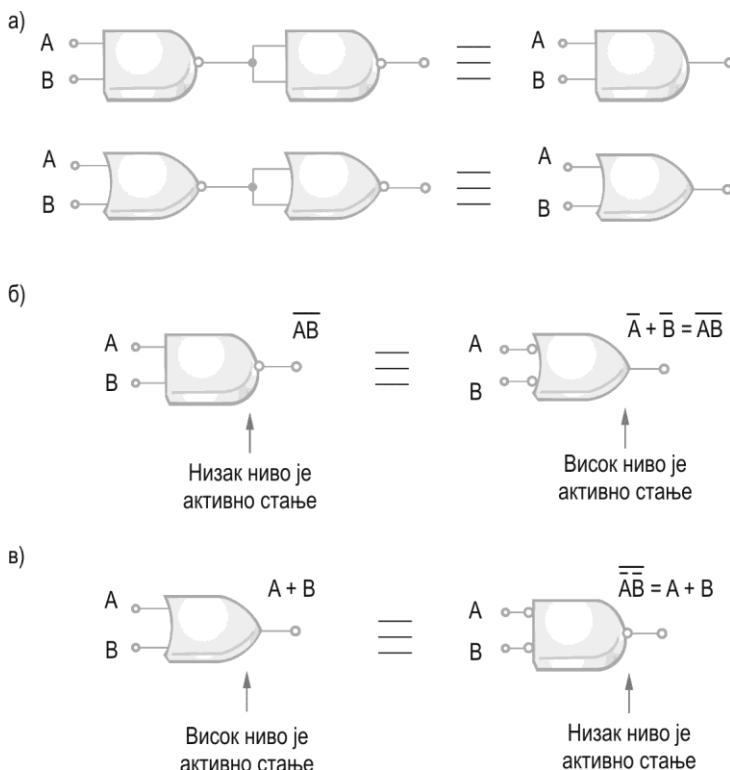
1.3 ЛОГИЧКИ ЕКВИВАЛЕНТИ

Исте логичке операције могу да се реализују на више начина. Уобичајен пример је приказан на слици 1.4а, где је И операција реализована са два НИ кола а ИЛИ операција са два НИЛИ кола. У сваком од ових случајева друго коло се користи као НЕ коло.

Два мање уочљива еквивалента приказана су на сликама 1.4а и б. На слици 1.4б, види се да су НИ кола еквивалентна са ИЛИ колима која имају инверторе на сваком улазу. Мали круг на ИЛИ колу представља инверзију операције и понаша се као НЕ коло. Ове две репрезентације, иако еквивалентне, могу се интерпретирати различито као што је приказано на дијаграму. Одлука која репрезентација ће бити употребљена у шеми логичких кола зависи од тога која су излазна стања нормална, а која су у активном стању.

На пример, уколико је излаз нормално висок за време рада кола, и пада на низак ниво када је побуђен - активан, само у одређеним тренуцима времена, користи се начин презентовања на десној страни. Другим речима, када је излазни сигнал нормално низак, а постаје висок

1. Дигитална кола



Слика 1.4 Еквиваленти логичких кола

када је активан (само у одређеним тренуцима времена), користи се репрезентација са леве стране.

Иста идеја се користи и за друге логичке операције, као што је ИЛИ операција приказана на слици 1.4а. Поново, обе репрезентације су еквивалентне за ИЛИ репрезентацију. Репрезентација а) се користи када је стање високог напонског нивоа активно излазно стање, а репрезентација б) се користи када је ниско стање активно излазно стање.

1.4 Фамилије интегрисаних логичких кола

Три важне карактеристике интегрисаних кола јесу брзина рада, потрошња и густина паковања (број елемената у интегрисаном колу).

Брзина рада

Брзина рада одрђује максималну радну учестаност логичких кола.

Најбржа су логичка кола из фамилија биполарних логичких кола. При томе су ECL¹ кола најбржа, спорија су Шоткијева TTL и I²L² кола, а најспорија конвенционална TTL кола. MOS³ фамилија је спорија. Најбржа су CMOS⁴ кола, а иза њих су PMOS⁵ и NMOS⁶ кола. У новије време CMOS кола достижу брзине биполарних (HC⁷).

Потрошња

CMOS су најбоља од напред поменутих кола, а потом I²L, TTL са малом потрошњом, PMOS и NMOS. ECL и Шоткијева TTL фамилија троше (расипају) највише енергије.

Густина паковања

NMOS је водећа технологија у овој области, за њим су CMOS па TTL и ECL.

1.5 Логика три стања

Развој рачунара са заједничком магистралом довео је до развоја логичких кола која имају три различита стања на излазу. Ова кола, која се називају кола са три логичка стања (TSL⁸), имају треће излазно стање као додаток логици високог и ниског напонског нивоа. Треће стање назива се стање високе импедансе, или високо Z стање.

Слика 1.5 показује симболе склопа са три логичка стања и рад еквивалентан колу 74HC125 (1.6). Улаз за селектовање ($\overline{EN}$ ⁹), који омогућава да ово коло ради, одређује излазну операцију тако да се излаз понаша или као логички излаз ($\overline{EN}=0$) или као коло високе

¹ Emitter Coupled Logic

² Integrated Injection Logic - означавају се као I²L.

³ Metal Oxide Semiconductor

⁴ Complementary Metal Oxide Semiconductor

⁵ PMOS - MOS са P каналом

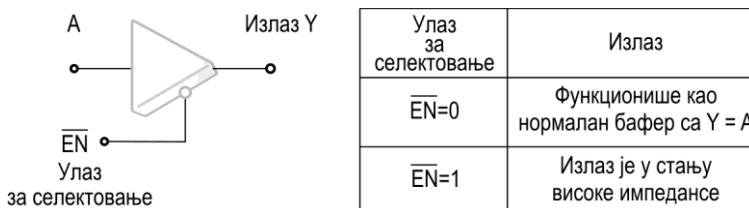
⁶ NMOS - MOS са N каналом

⁷ High Speed CMOS

⁸ Tri State Logic, Three State Logic, 3-state

⁹ Output Enable

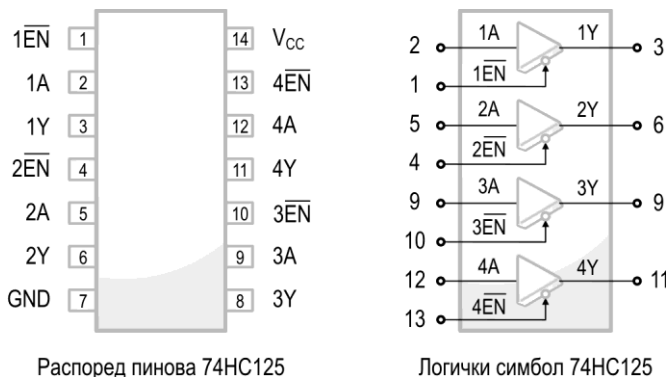
1. Дигитална кола



Слика 1.5 Бафери са три стања

импедансе ($\overline{EN}=1$). У стању када је селектовано коло се понаша као обично логичко коло, производећи излазни напонски ниво еквивалентан улазном логичком нивоу. У стању када није селектовано¹ коло се налази у стању високе импедансе. Излаз кола понаша се као да је откачен од логичког кола; другим речима, понаша се као да је отворено коло.

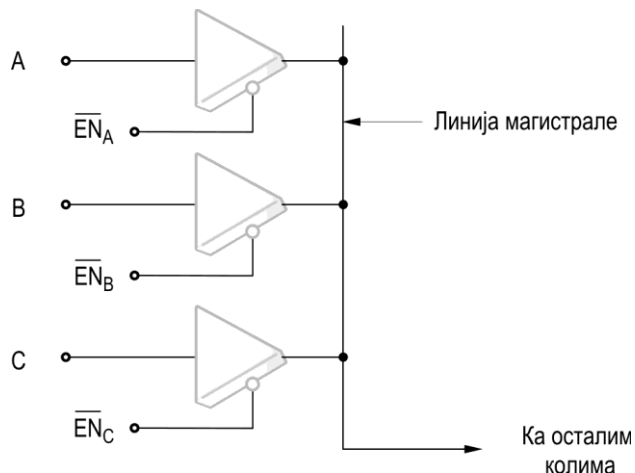
Улаз за селектовање на слици 1.5 активан је када је на ниском напонском нивоу², као што је назначено малим кругом на симболу логичког кола. Ово значи да је излаз селектован када је на улазу активан ниво. За логичка кола са три стања као што је 74HC125 (слика 1.6), низак напонски ниво на улазу за селектовање представља активан ниво, тако да су њихови излази селектовани када је на ниском напонском нивоу.



Слика 1.6 Интегрисано коло 74HC125

¹ Disable

² Термини имају исто значење: низак напонски ниво, напонски ниво логичке нуле, низак. Такође следећи термини се користе у истом контексту: висок напонски ниво, напонски ниво логичке јединице, висок.



Слика 1.7 Организација на магистрали са баферима са три стања

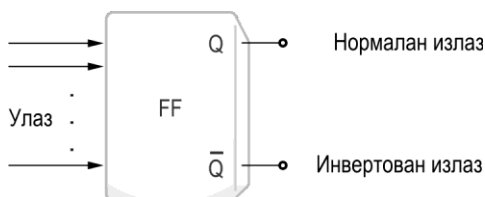
Логичка кола са три стања обично се примењују у системима у којима неколико логичких сигнала треба да се повеже на заједничку линију која се назива заједничка магистрала¹. Основна идеја приказана је графички на слици 1.7.

Оваква организација дозвољава да се било који од три сигнала (А, В, С) шаље преко магистрале ка другим колима селектовањем (активирањем) бафера који одговара жељеном сигналу. Уколико је у датом тренутку више од једног бафера селектовано, више излаза бафера биће активно, што производи сигнал на магистрали као комбинацију више сигнала и неодређено стање. Оваква ситуација није пожељна и назива се „сукоб” на магистрали. То може такође произвести оштећење излазних кола бафера, што је последица додатне струје када један излаз покушава да остане на ниском напонском нивоу, док други покушава да буде на високом напонском нивоу.

1.6 ФЛИП-ФЛОП

Логичка кола производе излазе који зависе само од тренутног логичког стања на његовим улазима. Дигитални системи такође захтевају

¹ Bus



Слика 1.8 Општи симбол флип-флопа

логичка кола која могу да произведу излазе који зависе од претходног стања, другим речима са памћењем (меморисањем). Широко коришћено меморијско коло које има два излаза (која су инверзија један другом) је флип-флоп (FF¹).

Слика 1.8 приказује ове излазе као Q и $\bar{Q}$ (може се користити било које слово, али је Q уобичајено). Q излаз се назива нормални излаз флип-флопа, а $\bar{Q}$ је инвертован излаз флип-флопа. Када се каже да је флип-флоп у стању високог нивоа (1) или ниског нивоа (0), онда се мисли на стање Q излаза. Наравно, у регуларном раду, $\bar{Q}$ је увек инверзија Q .

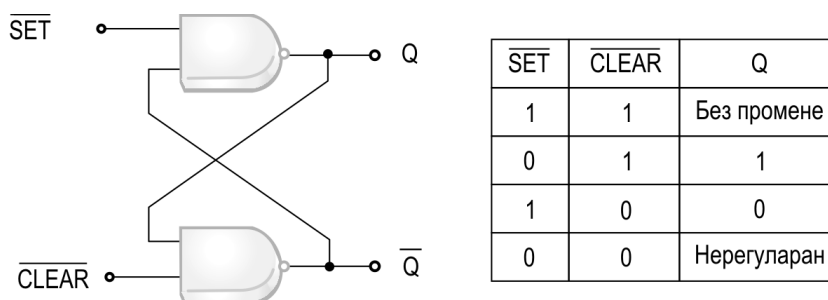
Постоје два могућа радна стања за флип-флоп: (1) $Q=0$, $\bar{Q}=1$; и (2) $Q=1$, $\bar{Q}=0$. Флип-флоп има један или више улаза, који се користе да проузрокују да флип-флоп прелази између два стања. Једном када улазни сигнал проузрокује да флип-флоп оде у неко стање, флип-флоп ће остати у том стању, чак и када се тај улазни сигнал оконча. Ово је карактеристика памћења. Флип-флоп је познат и под другим називима као што су бистабилни мултивибратор и лач. У овом уџбенику користићемо назив флип-флоп.

1.7 Базично флип - флоп коло

Слика 1.9 показује како су укрштена НИ кола повезана тако да формирају базично постави/обриши флип-флоп коло (SC FF²). Коло има два улаза, $\overline{SET}$ и $\overline{CLEAR}$. Ови улази су нормално увек 1. Уколико улаз дође на 0, Q излаз иде у стање 1 (и $\bar{Q}=0$). Уколико се улаз враћа на 1 Q излаз ће остати у стању 1, захваљујући интерној повратној спрези.

¹ Flip - Flop

² SET/CLEAR Flip - Flop



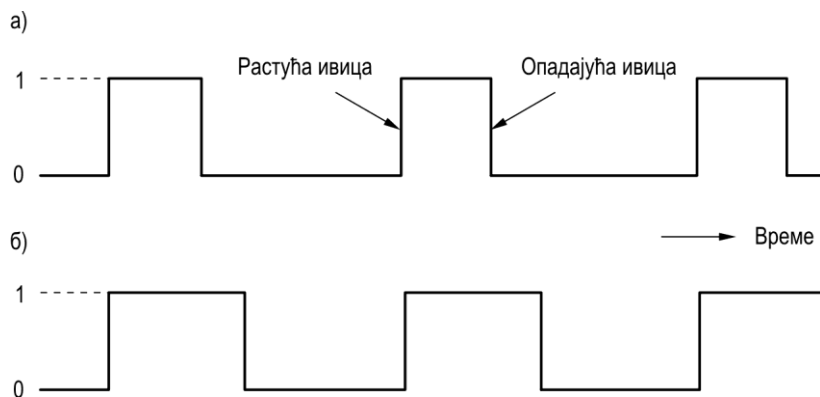
Слика 1.9 Основни постави/обриши флип-флоп (SC FF) направљен од НИ кола

Ово се назива постављање флип-флопа. Такође, када је $\overline{\text{CLEAR}}$ улаз 0, Q излаз ће прећи у стање 0 и остати у том стању. Ово се назива брисање флип-флопа. Табела на слици 1.9 приказује ову операцију. Уочимо да $\overline{\text{SET}}$ и $\overline{\text{CLEAR}}$ улазе не треба истовремено поставити на низак ниво, јер се тада појављује нерегуларно излазно стање. Улаз $\overline{\text{CLEAR}}$ често се назива и RESET улаз. Операција „ресетовања” флип-флопа истоветна је операцији брисања. Флип-флоп са слике 1.9 сам за себе има ограничену примену, али је базични градивни блок многих врста интегрисаних кола. Интегрисани флип-флопови користе такт сигнал да би синхронисали промене из једног стања у друго.

1.8 ТАКТ СИГНАЛИ

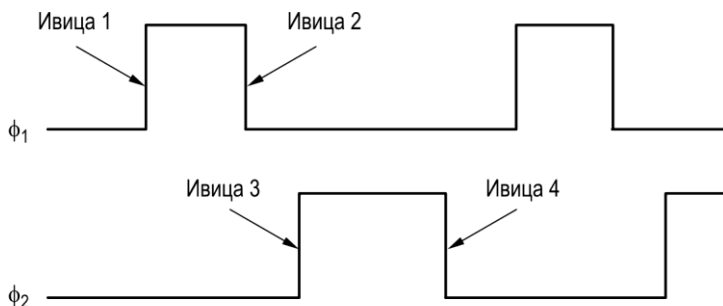
Многи дигитални системи раде као синхрони секвенцијални системи. Ово значи да је редослед операција које се извршавају синхронисан преко главног такт сигнала, који генерише периодичне импулсе који су дистрибуирани у све делове система. Једна врста такт сигнала приказана је на слици 1.10. Обично је то правоугли облик као што је приказано на сликама 1.10а и 1.10б. Такт сигнал обезбеђује да се промене стања система дешавају у тачно дефинисаним тренуцима и интервалима. Операције у систему извршавају се када такт сигнал прелази са 0 на 1, или са 1 на 0. Ова времена преласка означена су на слици 1.10. Прелазак (транзиција) са 0 на 1 назива се растућа (узлазна) ивица такт сигнала; прелазак са 1 на 0 назива се опадајућа (силазна) ивица такт сигнала.

1. Дигитална кола



Слика 1.10 Такт сигнал

Синхронизациона активност такт сигнала резултат је коришћења окидачких флип-флопова. Они су тако пројектовани да мењају стање на свом излазу на растућој или опадајућој (али не на обе) ивици такт сигнала. Другим речима, окидачки флип-флоп мењаће стање у одговарајућим променама (транзицијама) такт сигнала и неће реаговати нинашта између узастопних такт импулса. Максимална учестаност такт импулса генерално је одређена временом које је потребно флип-флоповима и улазима кола да реагују на промене нивоа које иницира такт импулс (тј. пропагационо



Стање 1: $\phi_1 = 0, \phi_2 = 0$

Стање 2: $\phi_1 = 1, \phi_2 = 0$

Стање 3: $\phi_1 = 0, \phi_2 = 1$

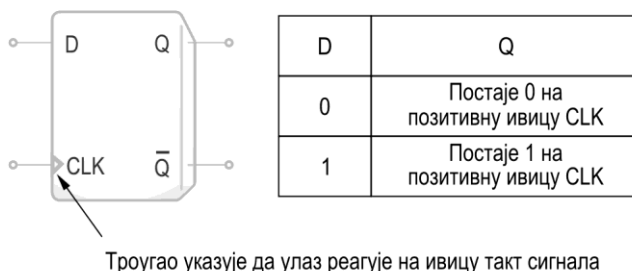
Слика 1.11 Двофазни такт сигнал

кашњење различитих логичких кола). Најчешће рачунарима, укључујући ту и микрорачунаре, управљају два или више повезаних такт сигнала. Једна комбинација такт сигнала приказана је на слици 1.11. Користе се два такт сигнала који су обележени са ϕ_1 и ϕ_2 (фаза 1 и фаза 2). Оваква организација обезбеђује четири различите ивице и три различита стања такт сигнала (у поређењу са два стања по периоди за један такт сигнал). У даљој анализи рада микрорачунара користићемо такт сигнале ϕ_1 и ϕ_2 .

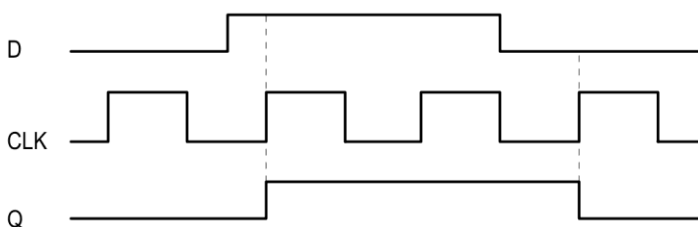
1.9 ОКИДАЧКИ ФЛИП-ФЛОПОВИ

Сви окидачки флип-флопови имају најмање два типа улаза: за такт сигнал (такт улаз - скраћено CLK) и један или више управљачких улаза. Управљачки улази се користе да одреде у које стање ће флип-флоп излаз да се постави када је сигнал доведен на такт улаз. Сигнал на CLK улазу је активан окидачки сигнал, који проузрокује да флип-флоп одговори (реагује) на управљачке улазе.

а)



б)



Слика 1.12 D флип-флоп који се окида улазном ивицом

1.9.1 Ивицом окидан D флип-флоп

Слика 1.12а показује симболе и табелу стања за ивицом-окидан D флип-флоп еквивалентан са колом 74НС74. Он се тако назива пошто има један управљачки сигнал, D. Ови флип-флопови раде тако да се логички нивои који су присутни на D улазу преносе на Q излаз само на узлазној ивици такт сигнала. У било ком другом тренутку D улаз нема никаквог ефекта на Q излаз, као што је илустровано примером временског дијаграма на слици 1.12б.

Уочимо да силазна ивица такт сигнала нема никаквог ефекта на излаз флип-флопа. Постоје D флип-флопови који се окидају силазном ивицом такт сигнала. Начин рада је исти, али је графичком приказу улаза такт сигнала додат мали круг који указује да се флип-флоп окида силазном ивицом.

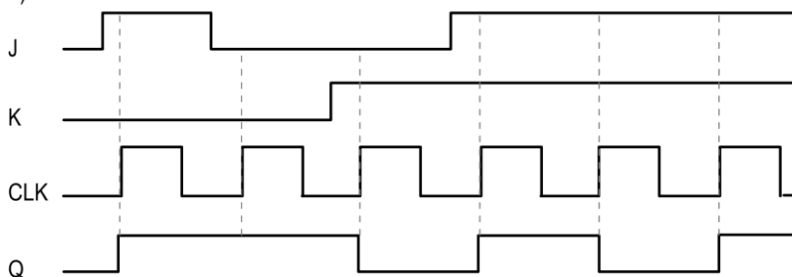
1.9.2 Ивицом окидан JK флип-флоп

Ивицом окидан JK флип-флоп је једна од многих врста флип-флопова која је у широкој употреби. Као што је приказано на слици 1.13, он користи

а)



б)



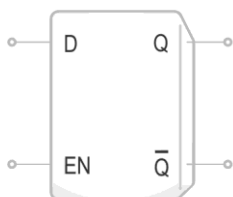
Слика 1.13 Ивицом окидани JK флип-флоп

два управљачка улаза, J и K. Ови улази одређују шта ће се догодити на Q излазу када се појављује узлазна ивица такт сигнала (а према одговарајућој табели стања). Уколико су (слика 1.136):

- J=0 и K=0 нема промене стања на излазу флип-флопа,
- J=1 и K=0 излаз флип-флопа биће Q=1 без обзира на претходно стање,
- J=0 и K=1 излаз флип-флопа биће Q=0 без обзира на претходно стање,
- J=1 и K=1 излаз флип-флопа Q прећи ће у супротно стање, тј. ако је био 1 прелази на 0 и обрнуто.

Поново учимо да нема промена на Q излазу осим када се појави узлазна ивица такт сигнала. Постоје JK флип-флопови који се окидају силазном ивицом као на пример 7476, чији је начин рада у суштини исти. Мали круг на улазу такт сигнала симболизује окидање негативном, силазном, ивицом.

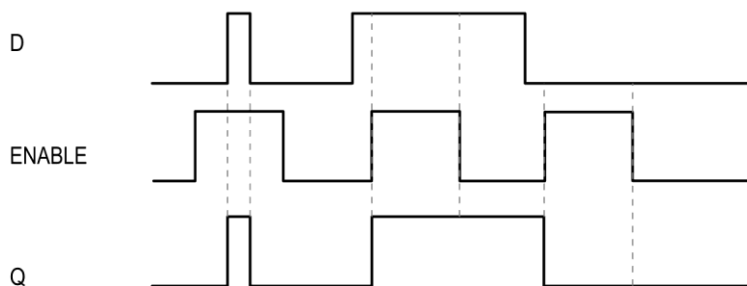
а)



D	EN	Q
X*	0	Чува претходну вредност
0	1	0
1	1	1

* X може бити 1 или 0

б)



Слика 1.14 Флип-флоп D типа (D FF)

1.9.3 Флип-флоп D типа

Ивицом окидан флип-флоп може да мења стања само када се појави одговарајућа ивица такт сигнала. Флип-флоп D типа или лач¹ сличан је ивицом окиданом D флип-флопу, с том разликом што он може да мења стања на свом излазу док је сигнал селектовања на његовом улазу на активном напонском нивоу.

На слици 1.14 приказан је графички симбол и начин рада D флип-флопа 7475. Докле год је сигнал за селектовање висок излаз Q ће пратити промене на D улазу. Када сигнал за селектовање пређе на низак ниво излаз Q ће запамтити (држати, сачувати) последњу вредност. Од тог тренутка надаље промене на D улазу немају више никаквог утицаја на Q излаз.

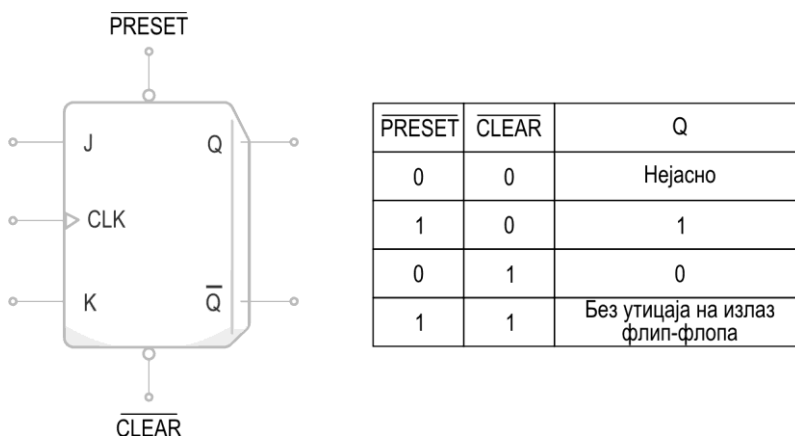
1.10 СИНХРОНИ И АСИНХРОНИ УЛАЗИ ФЛИП-ФЛОПОВА

Код флип-флопова које смо анализирали JK и D улази означавају се као управљачки (контролни) улази. Често се називају синхрони улази пошто је њихов ефекат на излазе флип-флопова синхронисан са улазом такт сигнала. Већина ових (окидачких) флип-флопова има један или више асинхронних улаза, који раде независно од синхронних улаза и улаза такт сигнала. Асинхрон улаз може, у било ком тренутку, независно од стања на другим улазима поставити излаз флип-флопа (логичко 1) или обрисати излаз флип-флопа (логичко 0). На други начин речено, асинхрони улази су надвладавајући² улази, који се могу користити да надвладају све друге улазе и поставе излаз флип-флопа у једно од стања.

Слика 1.15 приказује JK флип-флоп са такт сигналом, $\overline{\text{PRESET}}$ и $\overline{\text{CLEAR}}$ улазима. Асинхрони улази активирају се нултим логичким нивоом, што је означено малим кругом на симболу флип-флопа. Одговарајућа табела истинитости показује какав утицај имају ови улази на излаз флип-флопа. Низак ниво на $\overline{\text{PRESET}}$ улазу поставља Q излаз ($Q=1$). Низак ниво на $\overline{\text{CLEAR}}$ улазу тренутно брише Q излаз ($Q=0$).

¹ Latch

² Override



Слика 1.15 JK флип-флоп са асинхроним улазима

Истовремено ниски нивои на улазима $\overline{\text{PRESET}}$ и $\overline{\text{CLEAR}}$ доводе до нерегуларног стања (најчешће су оба излаза на 1). Када ниједан од ових улаза није низак флип-флоп је у могућности да одговори на J, K и CLK улазе. Важно је уочити да асинхрони улази реагују на ниво сигнала. Ово значи да уколико се константа 0 држи на $\overline{\text{PRESET}}$ улазу излаз флип-флопа ће бити у стању $Q=1$, независно од тога шта се догађа на другим улазима. Слично, константно низак ниво на $\overline{\text{CLEAR}}$ улазу држи флип-флоп у стању $Q=0$. Тако се асинхрони улази могу употребити за држање флип-флопа у одређеном стању у било ком жељеном интервалу.

Треба напоменути да промена стања на излазима касни у односу на активну ивицу такт сигнала. Ово кашњење зависи од технологије у којој су кола израђена и може се видети у произвођачкој документацији.

1.11 БИНАРНИ БРОЈАЧ

Флип-флопови повезани на различит начин могу да раде као бинарни бројачи који броје импULSE улазног такт сигнала. Велики број бројача може се наћи у виду стандардних интегрисаних кола¹ и ретко када је потребно конструисати бројач од посебних флип-флопова.

Због тога нећемо задирати у интерни рад бројача. Уместо тога,

¹ IC (Integrated Circuit)

а)



б)

D	C	B	A	
0	0	0	0	После првог импулса на улазу
0	0	0	1	После другог импулса на улазу
0	0	1	0	После трећег импулса на улазу
0	0	1	1	После четвртог импулса на улазу
.	.	.	.	...
1	1	0	0	После дванаестог импулса на улазу
1	1	0	1	После тринаестог импулса на улазу
1	1	1	0	После четрнаестог импулса на улазу
1	1	1	1	После петнаестог импулса на улазу
0	0	0	0	После шеснаестог импулса на улазу
0	0	0	0	После седамнаестог импулса на улазу

Слика 1.16 Четворобитни бројач

посматраћемо спољне радне карактеристике доступних, интегрисаних бројача.

1.11.1 Основни рад бројача

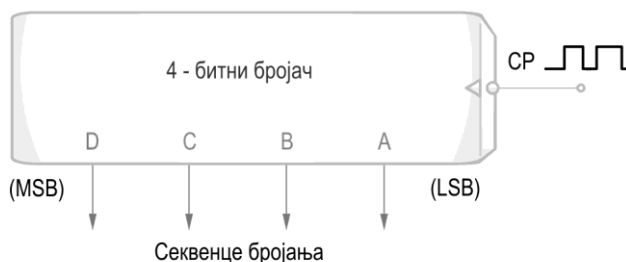
Слика 1.16а представља шематски приказ 4-битног бројача. Овај бројач садржи 4 флип-флопа (сваки за по један бит), са излазом означеним са А, В, С и D. Приказана су два улаза: улазни такт импулс (CP) и улаз за поништавање (RESET). Стање на излазима четири флип-флопа бројача представља бинарни број који је једнак броју импулса који су доведени

на CP улаз. У табели (слика 1.166) представљене су секвенце сигнала на излазима флип-флопова када се на улаз бројача доводи такт сигнал. Излаз A представља бит најмање тежине (LSB), а излаз D бит највеће тежине (MSB) бинарног бројача. На пример, после петог импулса улазног такт сигнала излаз је DCBA=0101, односно бинарни еквивалент броја пет. Уочимо ознаке на CP улазу: мали круг и троугао који указују да флип-флоп у бројачу мења стање (окида се) на силазној ивици такт сигнала. Бројачи који се окидају узлазном ивицом такт сигнала могу се такође наћи на тржишту (и немају ознаку круг на CP улазу).

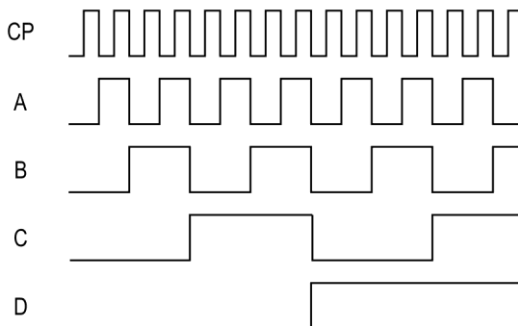
1.11.2 Секвенце бројања

Генерално, бројач са N флип-флопова може да се нађе у 2^N различитих стања. Укупан број различитих стања назива се MOD-број бројача. На слици 1.17 је MOD-16 бројач. Бројач са N флип-флопова је MOD- 2^N бројач.

а)



б)



Слика 1.17 Сигнали на излазу бројача

Неки интегрисани бројачи пројектовани су тако да корисник може да мења секвенцу бројања преко одговарајућих спољашњих логичких веза. Они се најчешће називају променљиви MOD бројачи или програмабилни бројачи.

1.11.3 Дељење учестаности

Поред бројања импулса бројачи могу да обављају и дељење учестаности. Ово је илустровано сликом 1.17 за 4-битни, MOD-16 бројач. Види се да се стање излаза А мења брзином тачно двоструко мањом ($1/2$) од улаза CP. За остале излазе:

- учестаност излаза В је $1/2$ учестаности излаза А и $1/4$ CP улаза,
- учестаност излаза С је $1/2$ учестаности излаза В и $1/8$ улазне учестаности,
- учестаност излаза D је $1/2$ учестаности излаза С и $1/16$ улазне учестаности.

Генерално, учестаност сигнала на излазу бројача одговараће учестаности улазног сигнала подељеној са MOD бројем. На пример, D излаз декадног бројача (MOD-10) имаће учестаност која је $1/10$ учестаности улазног такт сигнала.

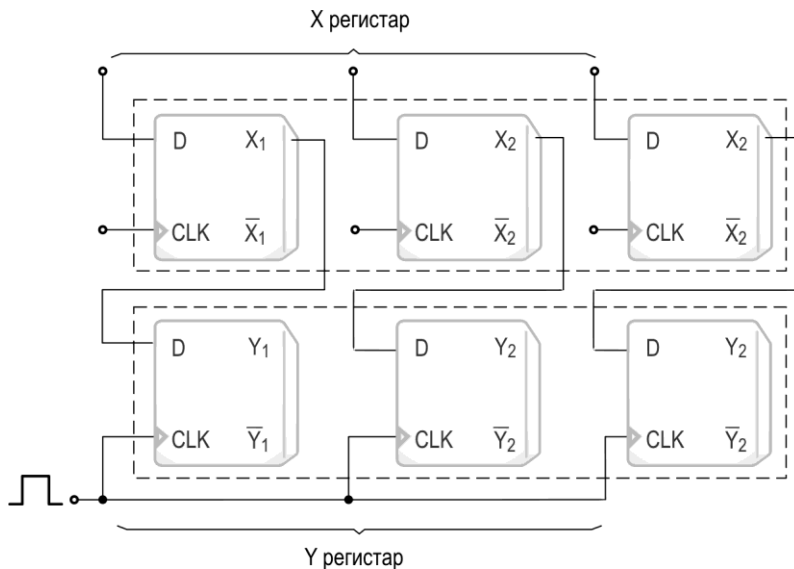
1.11.4 Бројач навише и наниже

Бројачи које смо до сада спомињали могу да броје од 0 до неке максималне вредности и затим да се поново враћају на нулу (ресетују). Постоје и интегрисани бројачи који могу да броје у оба правца и називају се бројачи навише/наниже.

1.12 ФЛИП-ФЛОП РЕГИСТРИ

Регистар је група меморијских кола која се користи да смести бинарни податак. Много комплекснији регистри могу и да (на одређени начин) модификују смештене информације. Најчешће коришћени меморијски елемент је флип-флоп, а бројач је један од примера флип-флоп регистра. Два друга типа флип-флоп регистра су регистар за смештање (бафер) и померачки регистри.

Као што ћемо видети у поглављима која следе, регистри играју значајну улогу у дигиталним рачунарским системима. У ствари, у свом најједноставнијем облику, рачунар је скуп регистара у коме се бинарни



Слика 1.18 Паралелни пренос из X у Y регистар

подаци преносе из једног у други регистар, где се над њима изврши нека операција, и затим поново преносе у неки од регистра. За сада, задржаћемо се на операцији преношења информације из једног регистра у други.

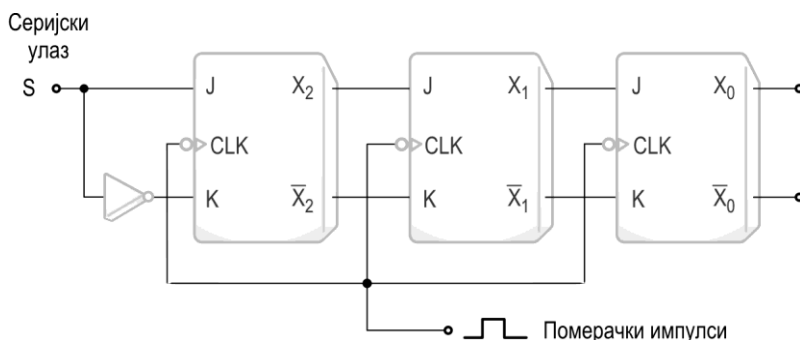
1.12.1 Паралелни пренос

На слици 1.18 представљена је паралелна веза два 3-битна регистра. Регистар X садржи флип-флопове X_1 , X_2 и X_3 ; регистар Y састоји се од флип-флопова Y_1 , Y_2 и Y_3 . Сваки флип-флоп је ивицом окидани D флип-флоп. Импулс за пренос (трансфер) који се појављује на CLK улазу Y регистра доводи до тога да се вредност X_1 преноси у Y_1 , X_2 у Y_2 и X_3 у Y_3 . Овај пренос садржаја X регистра у Y регистар је паралелни пренос, пошто се сва три бита преносе истовремено. Када посматрамо пренос из регистра у регистар користимо поједностављени начин означавања који показује такав пренос.

$$[X] \rightarrow Y$$

Симбол $[X]$ значи садржај X регистра. Стрелица означава правац преноса.

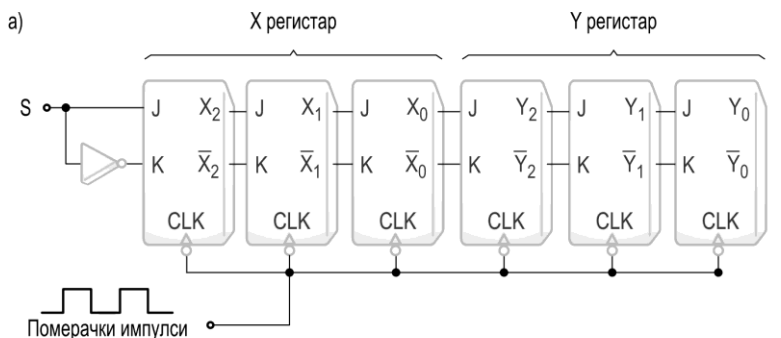
1. Дигитална кола



Слика 1.19 Померачки регистар

1.12.2 Серијски пренос

Серијски пренос из једног регистра у други извршава се тако што се преноси 1 бит у јединици времена. Пре него што објаснимо серијски пренос анализираћемо начин рада померачких регистра. Слика 1.19



б)

S	X ₂	X ₁	X ₀	Y ₂	Y ₁	Y ₀	
0	1	0	1	0	1	1	(Пре померачких импулса)
0	0	1	0	1	0	1	(После једног импулса)
0	0	0	1	0	1	0	(После два импулса)
0	0	0	0	1	0	1	(После три импулса)

Слика 1.20 Серијски пренос из X регистра у Y регистар

показује 3-битни померачки регистар који користи три JK флип-флопа.

Флип-флопови су повезани тако да се када се појави силазна ивица померачког импулса (SHIFT) текући садржај сваког флип-флопа преноси у флип-флоп на његовој десној страни. Ниво присутан на серијском улазу S преноси се у X_2 флип-флоп.

Да бисмо ово илустровали претпоставимо да је садржај X регистра $X_2=1$, $X_1=0$, $X_0=1$. Користећи скраћенице можемо да напишемо: $[X]=101_2$. Такође, претпоставимо да је $S=0$. Када се померачки импулс појави суседни флип-флоп преузима вредност логичког нивоа претходног флип-флопа. Тако, после појављивања силазне ивице померачких импулса садржај регистра је $[X]=010_2$. То значи: ниво са улаза S помери се у X_2 флип-флоп, претходни ниво X_2 флип-флоп помера се у X_1 флип-флоп, и претходни ниво X_1 флип-флоп помера се у X_0 флип-флоп. Претходни ниво (вредност) X_0 флип-флопа је изгубљен.

Слика 1.20 показује два 3-битна померачка регистра повезана тако да се садржај једног регистра (X) преноси серијски у други регистар (Y). Потребна су три померачка импулса, по један за сваки бит. Садржаји X и Y регистра у току померања представљени су на слици 1.20б, када је вредност на улазу $S=0$, а садржаји регистра $[X]=101_2$ и $[Y]=011_2$. За серијски пренос потребно је више времена него за паралелни пренос, пошто се преноси само 1 бит у јединици времена. Када се два регистра повезују серијски захтева се мањи број линија него када би се они паралелно повезивали.

1.13 РЕГИСТРИ

Ретко је потребно правити регистре од посебних флип-флопова пошто је доступан велики број интегрисаних кола. У овом уџбенику није нам намера да покажемо бројне варијанте, већ ћемо анализирати само основне врсте које се користе у микрорачунарским применама.

1.13.1 Регистри за држање података

Регистар за држање података (леч¹) јесте регистар који користи D врсту флип-флопа. Слика 1.21 показује шематски приказ 4-битног леч

¹ Data Latching Registers



Слика 1.21 Четворобитни регистри за држање података

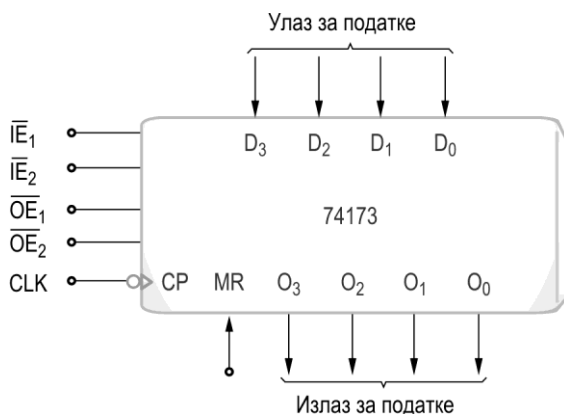
регистра. Улаз за селектовање заједнички је за сваки леч и омогућава да излази података Q_3 , Q_2 , Q_1 и Q_0 реагују на улазе података D_3 , D_2 , D_1 и D_0 и то на следећи начин:

- Када је улаз за селектовање на високом нивоу сваки Q излаз следи логичке нивое присутне на одговарајућим D улазима (тј. Q_3 прати D_3).
- Када улаз за селектовање пређе на низак ниво (падне), сваки Q излаз задржава последњу вредност са D улаза, и не мења се чак и ако се D промени.

Када се на улаз за брисање ($\overline{CLR}$) доведе низак ниво истовремено



Слика 1.22 Четворобитни ивицом окидани регистри

Слика 1.23 Логички симбол за 74173¹

сви излази за податке постављају се на напонски ниво који одговара логичкој нули².

1.13.2 Регистар окидан ивицом

Тип регистра који се окида ивицом такт сигнала (слика 1.22) користи ивицом окидани D флип-флоп и ради као регистар на слици 1.21, сем што улаз података утиче на излаз података у тренутку када такт сигнал (CLK) прелази са ниског на висок ниво. У том тренутку, нивои присутни на D улазу преносе се на Q излаз. Промене нивоа на D улазу неће изазвати промене на Q излазу, али за исправан пренос података неопходно је да је D улаз стабилан када наступа активна транзиција такт сигнала.

1.13.3 Регистар са три стања

У рачунарима пренос података извршава се преко заједничког скупа линија повезивања које се називају магистрала података³. Многи различити делови могу да имају своје излазе и улазе везане на заједничке линије магистрале података. Због тога делови који су везани на магистралу података често имају излазе са три стања, или су везани за магистралу података преко бафера са три стања као што је представљено на слици 1.7. Неки од ових модула који су обично везани за магистралу података

¹ IE (Input Enable), OE (Output Enable), MR (Master Reset)

² У овом уџбенику користићемо и израз „поставити на 0” или „обрисати”.

³ Data Bus

јесу: микроконтролери, полупроводничка меморијска интегрисана кола, дигитално-аналогни (Д/А) и аналогно-дигитални конвертори (А/Д), универзални асинхрони примопредајник (UART¹) и периферијски интерфејсни адаптер (PIA²).

Модул који се везује на магистралу података садржи и лич регистре. Излази ових регистара повезују се на магистралу преко бафера са три стања. Постоје многи интегрисани регистри који се могу наћи на тржишту, а који садрже бафере са три стања на истој компоненти са регистрима. Један од њих је интегрисано TTL коло 74173³ чији симбол је приказан на слици 1.23.

Интегрисано коло 74173 је 4-битни регистар са паралелним улазом/излазом. Постоје четири флип-флопа, чији су излази интерно повезани са баферима са три стања. Када је $\overline{OE}_1 = \overline{OE}_2 = 0$, селектовани су бафери са три стања чиме се обезбеђује активирање O_0 - O_3 излаза.

Уколико су OE_1 , или OE_2 или оба једнака 1, излази чипа O_0 - O_3 су у стању високе импедансе. Подаци који се доводе на D_0 - D_3 улаз интегрисаног кола повезани су интерним везама на D улазе флип-флопова регистра. Ова логичка кола допуштају два начина рада :

1. Упис података: када је $IE_1 = IE_2 = 0$ и када се појави узлазна ивица такт импулса (CP) подаци са улаза D_0 - D_3 преносе се у флип-флопове;
2. Држање података: када су $\overline{IE}_1$ или $\overline{IE}_2$ или оба на 1, а појави се узлазна ивица такт импулса (CP) подаци смештени у флип-флоповима регистра не мењају своју вредност.

Операција уписивања може се извршити независно од тога да ли су (или нису) излази у стању високе импедансе. Када је улаз $MR=1$ садржаји флип-флопова регистра бришу се (асинхроно), без обзира на логичка стања на улазима $\overline{IE}_1$, $\overline{IE}_2$, $\overline{OE}_1$ и $\overline{OE}_2$.

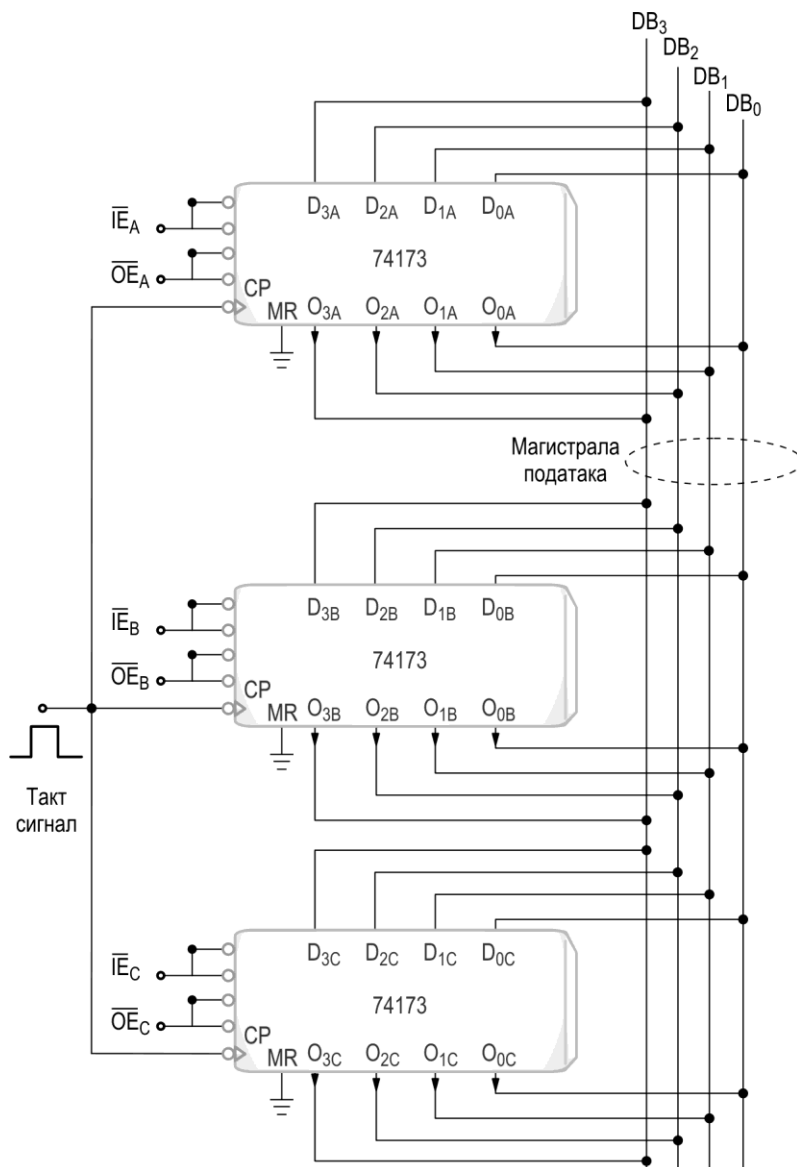
1.14 ПОВЕЗИВАЊЕ НА ЗАЈЕДНИЧКОЈ МАГИСТРАЛИ

Заједничка магистрала је важна компонента микрорачунарских система. У овом одељку илустроваћемо како се магистрала података

¹ Universal Asynchronous Receiver Transmitter

² Peripheral Interface Adapter

³ CMOS верзија 74HC173.



Слика 1.24 Регистри са три стања повезани на магистралу података

користи за пренос података из регистра у регистар. На слици 1.24 приказан је систем организован са магистралама и интегрисаним колима 74173. Уочимо да сваки регистар има свој пар $\overline{OE}$ улаза међусобно повезаних, тако да чине један заједнички $\overline{OE}$ улаз (исто је и са $\overline{IE}$ улазима). Такође, уочимо да су регистри означени са А, В и С од дна ка врху. Ово се види у ознакама сваког улаза и излаза.

У оваквој организацији магистрала података састоји се од четири линије које су означене са DB_0 - DB_3 . Одговарајући излази сваког од регистра повезани су на исту линију магистрале података (нпр. OZ_A , OZ_B и OZ_C повезани су са DB_3). Пошто три регистра имају своје излазе међусобно повезане неопходно је да само један регистар има селектован свој излаз, док су друга два излаза регистра у стању високе импедансе. У супротном, постојао би „сукоб” на магистрали (два или више излаза би тражили приступ магистрали) и могуће оштећење компонената.

Одговарајући улази регистра такође су повезани на исту линију магистрале (нпр. DZ_A , DZ_B , DZ_C су повезани са DB_3). Према томе, нивои на магистрали увек ће бити спремни за пренос у један или више регистра, зависно од улаза.

1.14.1 Пренос података

Садржај било ког од три регистра може се преко магистрале података (користећи паралелни пренос) пребацити из једног регистра у други, постављањем одговарајућих логичких нивоа на улазе за селектовање регистра. Уобичајено, управљачка јединица рачунара (нпр. CPU¹) генерише сигнале који селектују регистар који поставља своје податке на магистралу података, и регистар који узима (учитава) те податке. Пример 1.1 илуструје овакав начин рада.

Пример 1.1 *Описати које се вредности улазних сигнала захтевају када треба пренети садржај регистра А у регистар С: $[A] \rightarrow C$.*

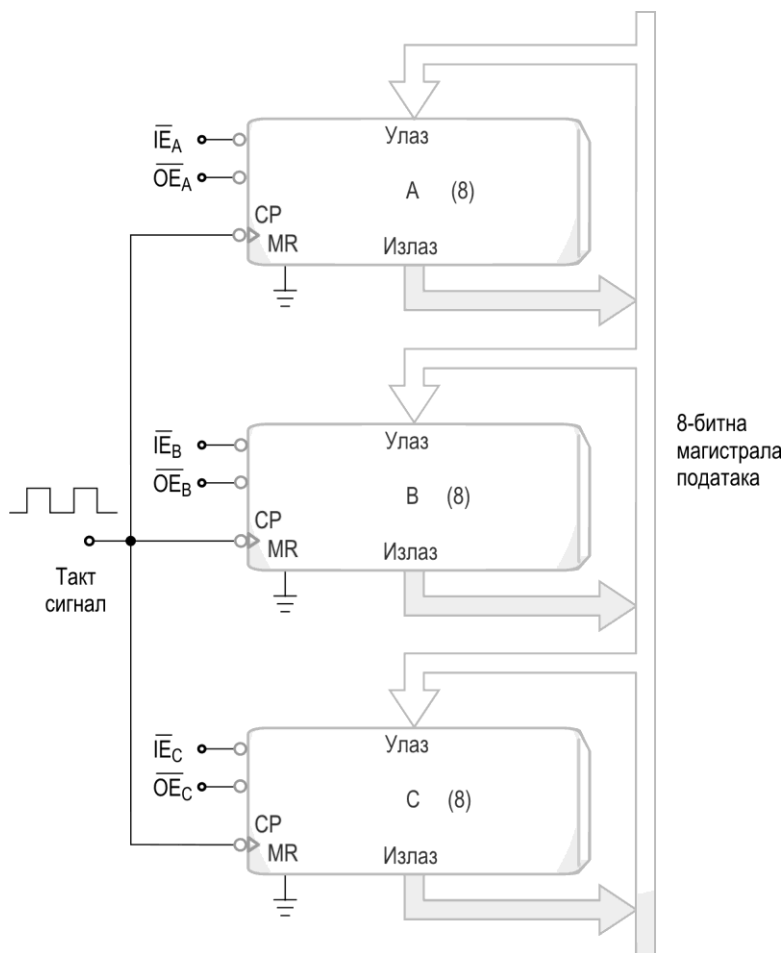
Решење: Потребно је селектовати само излазе регистра А (не В и С). То значи да:

¹ Central Processing Unit

$$\overline{OE}_A=0, \overline{OE}_B=\overline{OE}_C=1$$

Са овим вредностима сигнала $\overline{OE}_A$, $\overline{OE}_B$ и $\overline{OE}_C$ само ће садржај регистра А бити постављен на магистралу података. Потребно је селектовати улазе регистра С што значи:

$$\overline{IE}_C=0, \overline{IE}_A=\overline{IE}_B=1$$



Слика 1.25 Поједностављени дијаграм организације на магистрали података

Ове вредности сигнала обезбеђују да само регистар С може да, када се појави узлазна ивица сигнала, прихвати податке са магистрале података. На крају, још један такт импулс је потребан да се подаци са магистрале пренесу (упишу) у регистар С.

На магистралу података може се повезати (додати) већи број регистара. Сваки од додатих регистара захтева улазе за селектовање којима треба управљати. Додатни регистри имају улазе за податке које треба да побуђују¹ излази оних регистара који постављају податке на магистралу података.

У системима где је велики број регистара (или других склопова) везан на магистралу, излази регистара могу да буду везани (баферовани) преко кола која се називају драјвери² магистрале. Чак и када су сви модули MOS и CMOS, укупно капацитивно оптерећење, везано на сваку линију магистрале, погоршава време преласка са једног логичког нивоа на други на свакој од линија магистрале.

Драјвер за магистралу података треба да има веома малу излазну импедансу у сваком логичком стању, и треба да буде у стању да пуни и празни капацитивност магистрале много брже него стандардно интегрисано коло.

1.14.2 Поједностављена репрезентација магистрале

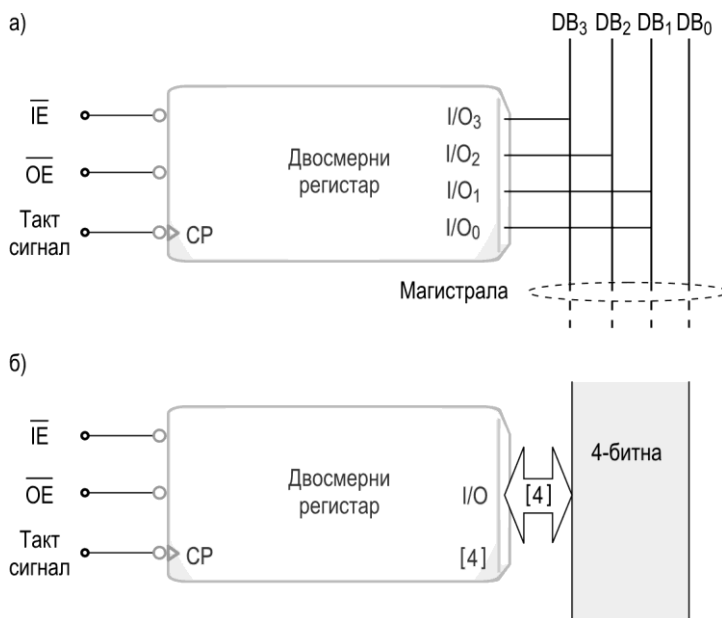
Постоји много модула који се везују за исту магистралу. На графичком приказу међусобних веза (шеми кола) ово може да изазове забуну око линија и спојева. Када нису неопходне детаљне шеме кола користи се једноставнији приказ магистрала. Поједностављен изглед осмолинијске магистрале података приказан је на слици 1.25. Веза са и на магистралу података представљена је широком стрелицом.

1.14.3 Двосмерно повезивање на заједничку магистралу

Сваки регистар на слици 1.24 има и улаз и излаз повезан на магистралу података, тако да су одговарајући улази и излази међусобно кратко спојени. На пример, сваки регистар има излаз O_2 повезан на улаз D_2 због њихове заједничке везе са DB_2 .

¹ Driven

² Bus drivers



Слика 1.26 а) Двосмерни регистри;
б) поједностављен приказ магистрале података

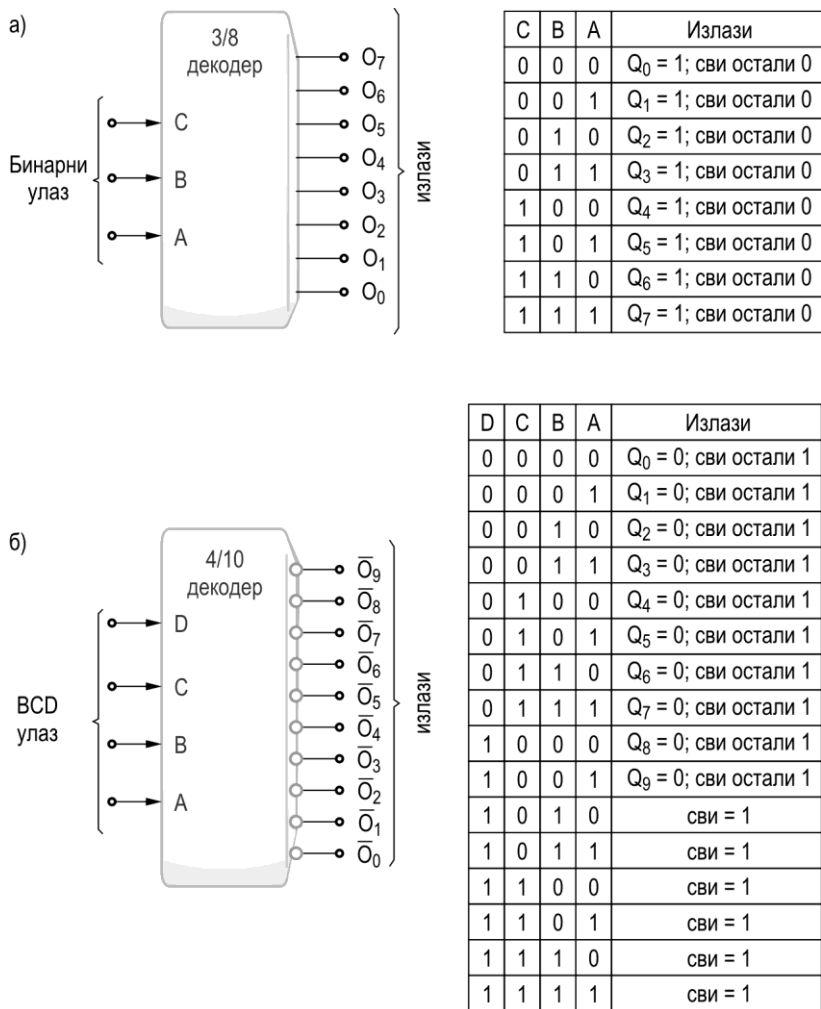
У системима са заједничком магистралом излази и улази обично су међусобно повезани. Због тога су произвођачи, да би смањили број контакта (пинова), развили интегрисана кола у којима су излази и улази интерно повезани. На слици 1.26 приказан је један овакав 4-битни регистар. Посебна улазна линија за податке (D_0-D_3) и улазна линија (O_0-O_3) замењена је улазно/излазним линијама ($I/O_0 - I/O_3$).

Свака улазно/излазна линија функционисаће као улазна или као излазна зависно од стања улаза за селектовање; зато се и називају двосмерне (бидирекционе) линије података. Најчешће меморије и микроконтролери користе магистрале за двосмерни пренос података.

1.15 ДЕКОДЕРИ

У дигиталним рачунарима бинарни код се користи да репрезентује различите типове информација као што су инструкције, нумерички подаци,

1. Дигитална кола



Слика 1.27 Декодери

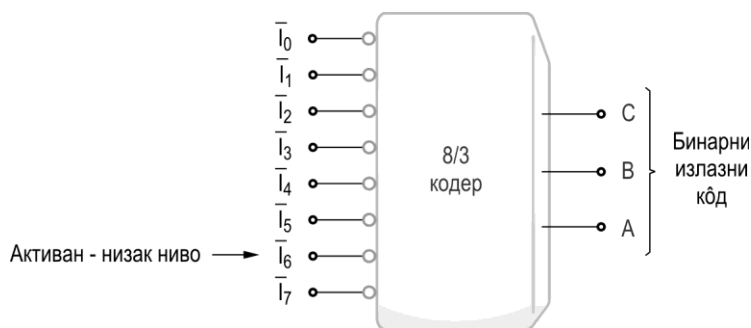
меморијске адресе и управљачке команде. Кодна група која садржи N битова може да има 2^N различитих комбинација, и свака од њих може да репрезентује различите делове информације. Према томе, јавља се потреба за логичким колама која могу да приме N -битни код као логички

улаз а затим генеришу одговарајући излазни сигнал да би означили која је од 2^n комбинација присутна. Такво коло назива се декодер.

Већина интегрисаних декодера може да декодује 2-, 3- или 4-битне улазне кодове. Неколико примера приказано је на слици 1.27 да бисмо илустровали карактеристике декодерских кола. Дијаграм на врху је 3-линијски на 8-линијски декодер. Он има 3-битни улазни код (C, B, A) и због тога $2^3=8$ могућих комбинација. Осам излазних линија 0 до 7 показују која је комбинација присутна. Ово је приказано одговарајућом табелом истинитости.

На пример, ако је улазни код CBA=100 (бинарни еквивалент од 4_{10}), излаз 4 биће 1 док су сви остали излази 0. Тако, један и само један излаз биће активан (висок) зависно од тога који улазни код је присутан. Ова врста декодера среће се и под називом бинарно-октални декодер, пошто од бинарног улазног кода производи осам излаза.

Декодер на слици 1.27б је 4-линије у 10-линија декодер. Иако он има 4-битни улаз користи се само 10 BCD улазних кодова (000 - 1001). Према томе, постоји само 10 излазних ножица (пинова). Свака излазна ножица је активна-ниска (што значи да је нормално високог нивоа) а пада на низак ниво када је одговарајући улазни код присутан. Уочимо да круг и знак за инвертовање¹ на излазу (нпр. $\overline{O}_3$) указују да је активан низак ниво. Овај декодер се уобичајено назива BCD-децимални декодер, пошто конвертује BCD-кодирани улаз у децимални излаз.



Слика 1.28 Кодек

¹ Знак за инвертовање обично се користи да би указао да је активан низак ниво на улазу или излазу.

1.16 КОДЕРИ

Декодер узима улазни кôд и активира један одговарајући излаз. Кодер обавља супротну операцију; генерише бинарни улазни кôд који одговара оној улазної линији која је активирана. Уобичајено коришћени интегрисани кодер представљен је на слици 1.28. Он има осам улаза који се активирају ниским нивоом, и који се нормално држе на високом нивоу. Када се један од ових улаза доведе на 0 генерише се бинарни излазни кôд који одговара том улазу. На пример, када је улаз $\bar{I}_3=0$, излаз би био CBA=011, што је бинарни еквивалент за децимално 3. Када је $\bar{I}_6=0$, излаз ће бити CBA=110.

1.16.1 Кодер са приоритетом

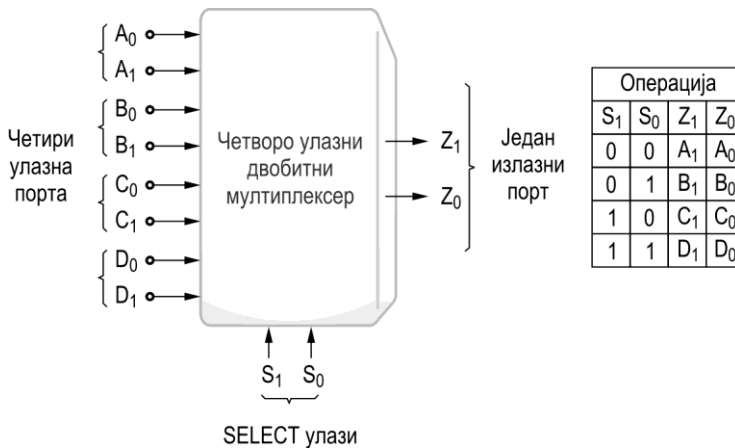
Шта се дешава ако је више од једног кодерског улаза постављено на низак ниво? За неке кодере излаз је неупотребљив. За кодере са приоритетом излази ће бити бинарни кôд за улаз са највећим бројем који је активиран. На пример, претпоставимо да је кодер на слици 1.28 кодер са приоритетом и да су улази $\bar{I}_4$ и $\bar{I}_7$ истовремено постављени на низак ниво. Излазни кôд биће CBA = 111, што одговара излазу 7. Без обзира колико улаза је активно, кôд за највиши улаз појавиће се на излазу.

1.17 МУЛТИПЛЕКСЕР (СЕЛЕКТОР ПОДАТАКА)

Мултиплексер или селектор података је логичко коло које може да има више података на свом улазу, али дозвољава само једном од њих, у



Слика 1.29 Општи дијаграм мултиплексера



Слика 1.30 Четвороулазни, 2-битни мултиплексер

датом тренутку, да се проследи на излаз. Прослеђивање жељеног податка са улаза ка излазу контролише се са улазом за селектовање који се среће и под називом адресни улаз. На слици 1.29 приказан је мултиплексер чији су улази и излази представљени великим стрелицама што указује да постоји више од једне линије за податке.

Постоје многи интегрисани мултиплексери са различитим бројем улаза за податке и за селектовање. Основни начин рада исти је за све њих а карактеристичан пример приказан је на слици 1.30. То је четвороулазни, 2-битни мултиплексер. Сваки улаз састоји се од 2 бита; постоје четири улаза који се срећу и под називом улазни портови. На пример, линије A_1 и A_0 представљају један улазни порт, и линије B_1 и B_0 други... Постоји само један излазни порт који чине линије Z_1 и Z_0 . Логички нивои на излазном порту, у било ком тренутку, биће идентични логичким нивоима једног од улазних портова одређених вредностима на линијама S_0 и S_1 односно улазом SELECT.

Када је:

- $S_1=S_0=0$, селектован је улазни порт A тако да ће бити: $Z_1=A_1$ и $Z_0=A_0$.
- $S_1=1$ и $S_0=0$, селектован је улазни порт C тако да је излазни порт $Z_1=C_1$ и $Z_0=C_0$.

Само селектовани улази имаће утицаја на излаз. Јасно, мултиплексер



Слика 1.31 Општи модел демултиплексера

се понаша као вишеполни, вишепозициони прекидач чија се позиција контролише преко логичких нивоа на S_0 и S_1 .

1.18 ДЕМУЛТИПЛЕКСЕРИ (ДИСТРИБУТЕРИ ПОДАТАКА)

Демултиплексер има један улазни порт и више излазних портова (слика 1.31). Подаци се са улазног порта прослеђују ка једном од излазних портова што је одређено SELECT улазом.

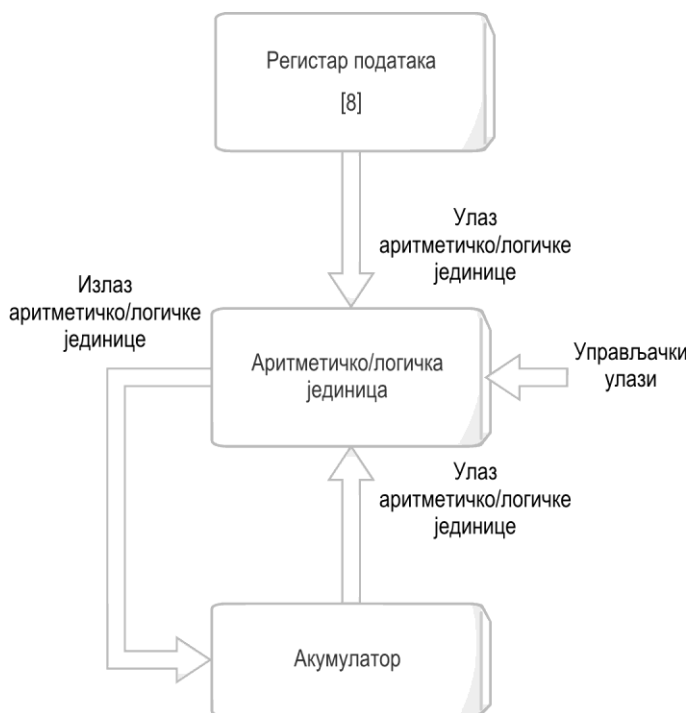
1.19 АРИТМЕТИЧКА КОЛА

Рачунар је у стању да извршава аритметичке, логичке и манипулативне операције над бинарним бројевима. Кола која обављају ове операције су интерна за рачунар и нису доступна корисницима. У оквиру овог уџбеника нећемо анализирати како се повезују логичка кола да би обављала функцију сабирања, одузимања, итд. већ ћемо их посматрати као једну целину која се назива аритметичко/логичка јединица (ALU¹).

На слици 1.32 приказан је дијаграм аритметичко/логичке јединице заједно са одговарајућим регистрима. Она се састоји од логичких кола која извршавају операције сабирања, одузимања, множења, дељења, манипулације са подацима (нпр. померање), и логичке операције (И, ИЛИ, итд.) над два броја у бинарном запису садржана у два улазна регистра.

Два регистра се пуне са два броја над којима ће аритметичко/логичка

¹ Arithmetic/Logic Unit



Слика 1.32 Аритметичко/логичка јединица

јединица да извршава операције. Ови бројеви називају се операнди. Аритметичко/логичка јединица извршава операције у зависности од управљачких улаза. На пример, један операнд је смештен у регистру за податке¹, а други операнд је смештен у регистру који се назива акумулатор². Управљачки улази одређују које ће операције аритметичко/логичка јединица извршавати над овим операндима. Резултат операције појављује се на излазу аритметичко/логичке јединице и смешта у акумулатор.

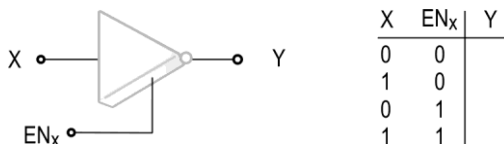
Детаљнија анализа рада аритметичко/логичке јединице биће дата у другим поглављима овог уџбеника.

¹ *Data Register*

² *Accumulator* – регистар који је директно везан за аритметичко/логичку јединицу и у који она уписује резултат аритметичко/логичких операција.

1.20 ПИТАЊА И ЗАДАЦИ

- Колико логичких излаза је потребно да би се представио 8-битни бинарни број:
 - у паралелном преносу
 - у серијском преносу
- Коло на слици 1.33 је инвертор са три стања. Попунити таблицу истинитости:



Слика 1.33 Слика за пример 2

- Објаснити разлику између синхроног и асинхроног улаза флип-флопа.
- Која је разлика између ивицом окиданог D флип-флопа и D лача?
- Из ког разлога регистри имају заједничке улазно/излазне линије?
- Упоредити паралелни и серијски преноса података између регистра.
- Особитни регистар садржи податак 10101010. Који ће садржај регистра бити после 4 померања удесно. Претпоставити да је на серијском улазу 1.
- Који су логички улази потребни да би се обезбедио пренос података из регистра C у регистар B на слици 1.25?
- Каква је разлика између стања високе импедансе магистрале података и пливајућег стања.
- Повезати модуле у првој колони са описом у другој колони табеле 1.1.

Модул	Опис
(а) декодер	1) даје бинарно кодирани излаз који одговара активном улазу
(б) кодер	2) даје активан сигнал на један од излаза који одговара бинарном улазном коду
(в) аритметичко/логичка јединица	3) повезује један од излаза са улазом
(г) мултиплексер	4) скуп аритметичких и логичких кола која се користе за реализацију свих аритметичких и логичких операција у рачунару

Табела 1.1

2. Меморијски модули

Једна од највећих предности дигиталних система над аналогним јесте способност да се једноставно и поуздано чува (смешта¹) велики број дигиталних података у дужем или краћем периоду времена. Ова способност чини дигиталне системе разноликим и прилагодљивим различитим потребама. На пример, у дигиталном рачунару интерна, главна меморија, садржи инструкције које указују рачунару шта да ради под свим могућим околностима, тако да рачунар обавља свој посао са минималном људском интервенцијом.

У овом поглављу биће дат кратак преглед најчешће коришћених врста меморијских компонената и система. У поглављу 1. анализирали смо начин рада флип-флопа као електронског меморијског елемента; како се групе флип-флопова (регистри) користе за смештање информација; како се ове информације могу пренети до других регистара. Флип-флоп регистри су меморијски елементи велике брзине који се користе у интерним операцијама дигиталних рачунара. Напредак у LSI² и VLSI³ технологији омогућио је да се велики број флип-флопова нађе у једном интегрисаном колу. Биполарне и MOS полупроводничке меморије најбрже су меморије, а њихова цена се смањује како LSI технологија напредује.

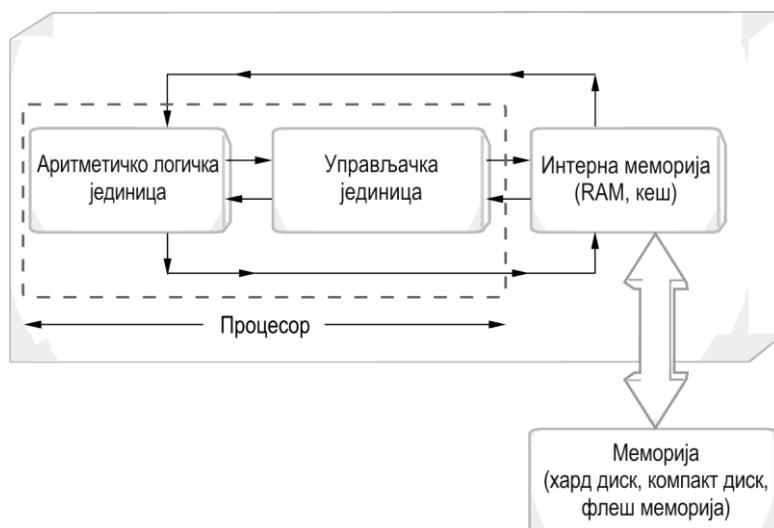
Дигитални подаци могу да се чувају као наелектрисање на кондензатору, у специјалном типу полупроводничких меморија које користе овај принцип да би постигле велику густину паковања. Полупроводничке меморије користе се као интерне меморије рачунара (слика 2.1) у којима је важна брзина рада. За време извршавања програма интерна меморија рачунара стално комуницира са процесор. Среће се и под називом интерна или радна меморија рачунара. Програм и подаци које програм користи обично су смештени у интерној меморији рачунара.

¹ Storage

² Large Scale Integration - висок степен интеграције

³ Very Large Scale Integration - веома висок степен интеграције

2. Меморијски модули



Слика 2.1 Рачунарски систем нормално користи интерну меморију велике брзине и спорију спољну меморију

Екстерна (спољна) меморија односи се на меморију која је ван процесорског модула (слика 2.1) и може да складишти (чува) податке без потребе за електричним напајањем. Спољне меморије које се данас користе су: магнетни диск¹ (масовна меморија), компакт диск и флеш меморија.

Масовна меморија значајно је спорија од интерне меморије и користи се за смештање информације (програма, података итд.) коју процесор тренутно не користи. Ова информација преноси се у интерну меморију када то процесор захтева.

Магнетни дискови и траке били су често коришћени уређаји за масовно смештање података пошто су им цене по јединици капацитета биле много ниже од цена полупроводничких меморија.

Флеш меморија је електрично избрисива програмабилна меморија (EEPROM²). Користи се за уређаје који користе меморијске картице или

¹ Хард диск (*hard disk drive - HDD, hard drive, hard disk, fixed disk drive*) је врста меморијских модула који не губе свој садржај и код којих се дигитални подаци чувају на магнетним површинама.

² *Electrical Erasable PROM - E²PROM* – електрично избрисива програмабилна меморија

преносиву меморију која користи USB¹ порт. Њихов капацитет је у све већем порасту, а цена све нижа.

2.1 ТЕРМИНОЛОГИЈА КОЈА СЕ КОРИСТИ КОД МЕМОРИЈА

Меморијска ћелија је компонента или електрично коло које се користи за чување једног бита (0 или 1). Примери меморијске ћелије су: флип-флоп, једно магнетно језгро и једна тачка на магнетном диску или траци.

Меморијска реч је група битова (ћелија) у меморији, која репрезентује информацију или податак истог типа. На пример, регистар који се састоји од осам флип-флопова може се сматрати да је меморија која чува 8-битну реч. Најчешћа дужина речи је од 4 до 64 бита, у зависности од типа или врсте рачунара.

Бајт је специјални термин који се користи за 8-битну реч.

Капацитет дефинише број битова смештених у одређеној меморијској компоненти (уређају) или комплетном меморијском систему. Да бисмо ово илустровали претпоставимо да имамо меморију у коју може да стане 4096 речи од по 20 битова. Укупан капацитет је 81920 битова. Капацитет можемо да изразимо и на следећи начин: 4096X20. Када капацитет описујемо на овај начин први број (4096) је број речи а други број (20) је број битова у речи (дужина речи). Број речи у меморији обично је мултипл од 1024. Уобичајено је да се користи ознака „1k”, што представља 1024 (2^{10}) када се односи на капацитет меморије. Тако, меморија која је капацитета 4kX20 у ствари је меморија капацитета 4096X20.

Густина је други термин који се везује за капацитет. Један меморијски уређај је веће густине од другог меморијског уређаја уколико може да складишти (чува) већи број битова у истој запремини.

Адреса је број који идентификује локацију речи у меморији. Свака реч, која је смештена у меморијском уређају или систему, има своју јединствену адресу. Адреса се увек специфицира као бинарни број, октални, хексадецимални или децимални. На слици 2.2 илустрована је меморија која се састоји од осам речи. Свака од ових осам речи има одређену адресу која се састоји од 3-битног броја који се креће од 000_2 до

¹ Universal Serial Bus – један од стандарда за повезивање уређаја на рачунар

2. Меморијски модули

Адресе	
0 0 0	Реч 0
0 0 1	Реч 1
0 1 0	Реч 2
0 1 1	Реч 3
1 0 0	Реч 4
1 0 1	Реч 5
1 1 0	Реч 6
1 1 1	Реч 7

Слика 2.2 Свака реч има своју бинарну адресу

111₂. Када се позивамо на специфичну реч у меморијској локацији, да би је идентификовали, користимо њен адресни код.

Код операција читавања приступа се бинарној речи, која је смештена на одређеној меморијској локацији (адреси), а затим се њен садржај преноси на другу меморијску локацију.

На пример, уколико желимо да употребимо четврту реч са слике 2.2, морамо извршити операцију читавања са адресе 100. Операција читавања назива се и „операција доношења”¹, пошто се садржај речи доноси из меморије.

Операција уписивања је операција којом се нови садржај смешта у одређну меморијску локацију. Она се назива још и операција смештања (ускладиштавања). Кад год се податак уписује на меморијску локацију он замењује податак који се претходно на тој меморијској локацији налазио. Процесом уписивања у меморијску локацију стара вредност (садржај) се губи.

Време приступа је мера брзине меморијског уређаја. То је време које је потребно за извођење операције читавања. Тачније, то је

¹ *Fetch*

време које протекне од тренутка пријема команде за читавање до појаве података на излазу меморије. Користи се симбол t_{acc} .

Време циклуса је друга мера за брзину меморијског уређаја. Представља време које је потребно да меморија изврши операције читавања или уписивања и врати се у нормално стање спремна за следећу команду. Циклус је дужи од времена приступа.

Меморије које губе свој садржај¹ захтевају електрично напајање да би се подаци сачували. Уколико се напајање укине сви подаци који су смештени у меморији биће изгубљени. Многе полупроводничке меморије губе садржај по нестанку напајања, док га магнетне меморије не губе.

Меморија са случајним приступом (RAM²) јесте меморија у којој стварна физичка локација речи нема утицаја на трајање читавања или уписивања на/са те локације. Другим речима, време приступа је исто за било коју адресу у меморији.

Меморије са секвенцијалним приступом (SAM³) јесу врста меморије чије време приступа није константно већ се мења у зависности од локације. Одређена реч налази се претраживањем по свим адресним локацијама док се не нађе жељена адреса. Овим се добија време приступа које је много веће него код меморија са случајним приступом. Пример секвенцијалног приступа су траке и дискови.

Меморија са уписивањем/читавањем (RW⁴) је било која меморија у коју се може уписати и прочитати.

Меморије које се могу само читавати (ROM⁵) јесу полупроводничке меморије које су пројектоване за примену у системима где је потребно само читавање (уређаји који раде са фиксним програмом, константним подацима). Техничким речником говорећи у ROM меморију се може уписати садржај (програмирати) само једном. Ову операцију уобичајено обавља произвођач.

¹ *Volatile*

² *Random Access Memory*

³ *Sequential Access Memory*

⁴ *Read/Write*

⁵ *Read Only Memory*

2. Меморијски модули

Статичке меморије¹ су полупроводничке меморије у којима ће смештени подаци остати сачувани док је напон напајања присутан, без потребе за поновним уписивањем података у меморију.

Динамичка меморија² је полупроводничка меморија у којој подаци неће остати сачувани уколико се не обавља периодично уписивање података у меморију. Ова операција назива се освежавање.

Интерна (главна, радна) меморија чува инструкције и податке које тренутно користи централна процесорска јединица. То је веома брза полупроводничка меморија.

Спољња (масовна) меморија служи за складиштење велике количине података ван интерне меморије. Мање је брзине од интерне меморије и задржава садржај без обзира на одсуство напона напајања.

2.2 ГЕНЕРАЛНИ НАЧИН РАДА СА МЕМОРИЈОМ

Иако је свака од врста меморије различитог интерног начина рада постоје одређени, основни принципи рада који су исти за све меморије. Разумевање основних појмова помоћи ће анализи појединих меморија.

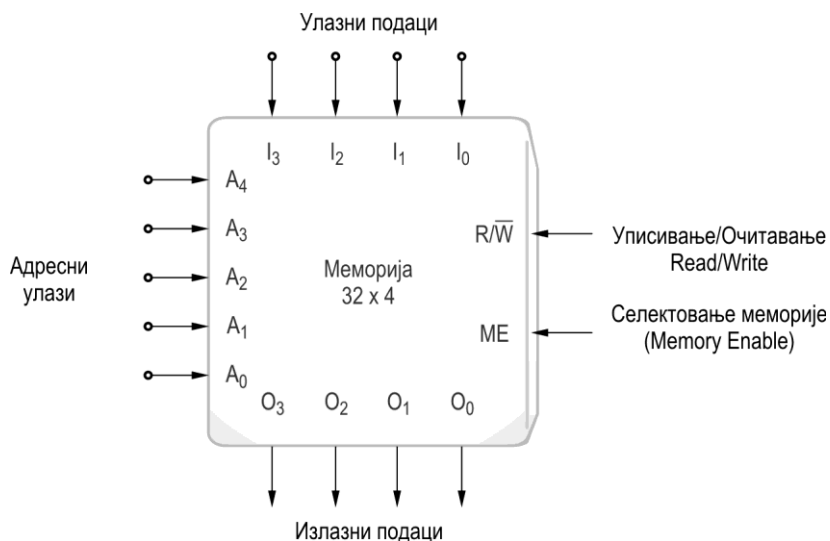
Сваки меморијски систем захтева неколико различитих типова улазно/излазних сигнала који обављају следеће функције:

1. Адресирају меморијску ћелију којој приступају код операције читавања или уписивања;
2. Дефинишу коју од операција ће извршити: операцију читавања или операцију уписивања;
3. Обезбеђују улазне податке који ће бити смештени у меморију за време операције уписивања;
4. Задржавају излазне податке који долазе из меморије за време операције читавања;
5. Селектују (или деселектују) меморију тако да она одговара (или не) на адресне улазе и команду уписивање/читавање.

Слика 2.3 илуструје ове основне функције на поједностављеном дијаграму меморије 32X4 у којој су смештене 32 4-битне речи. Пошто је величина речи 4 бита, постоје четири улазне линије за податке I_0, I_1, I_2 и четири

¹ *Static Memory Devices*

² *Dynamic Memory Devices*



Слика 2.3 Општи дијаграм за 32X4 меморију

излазне линије за податке $O_0 - O_3$. На улазне линије за време операције уписивања доводе се подаци које треба сместити у меморију. За време операције очитавања податак који се очитава из меморије појављује се на излазним линијама за податке.

Ова меморија има 32 различите локације за чување података и због тога 32 различите адресе, које су у опсегу од 00000_2 до 11111_2 (0 до 31 децимално). Зато је потребно пет адресних улаза $A_0 - A_4$ који одређују једну од 32 адресне локације. Код операције очитавања, да би се приступило једној меморијској локацији, потребно је довести на адресне улазе 5-битни адресни код те локације.

Улазна линија за очитавање/уписивање ($R/\bar{W}$) одређује која се меморијска операција извршава. Неки меморијски системи користе независне улазне линије, једну за очитавање¹ (R), а другу за уписивање² ($\bar{W}$). Када се користи једна линија $R/\bar{W}$ уобичајено је да је $R/\bar{W}=1$ команда за операцију очитавања, а $R/\bar{W}=0$ команда за операцију уписивања.

Многи меморијски системи имају начин да онемогуће приступ целој

¹ READ

² WRITE

2. Меморијски модули

меморији или делу меморије, тако да она неће реаговати на сигнале на улазима. На слици 2.3 овај улаз означен је као селектовање меморије (ME¹). Улаз је активан када је на високом нивоу, што значи да се меморијске локације могу читавати. Низак ниво на улазу спречава селектовање меморије, тако да она неће реаговати на адресне и $R/\overline{W}$ улазе. Улаз за селектовање меморије користи се када је потребно више меморијских модула мањег капацитета повезати у једну меморију већег капацитета.

Пример 2.1 *Одређена меморија има капацитет 4k X8*

а) Колико има улаза и излаза за податке?

б) Колико има адресних линија?

в) Колики је капацитет у бајтима?

Решење:

а) Осам улаза и осам излаза, пошто је дужина речи осам бита.

б) Меморија чува $4k=4 \times 1024 = 4096$ речи. Према томе, постоји 4096 меморијских адреса. Пошто је $4096=2^{12}$, захтева се 12-битни адресни код за специфицирање једне од 4096 адреса.

в) Бајт је 8 битова. Ова меморија је капацитета 4096 бајтова.

Пример меморије са слике 2.3 илуструје важне улазно/излазне функције које су заједничке већини меморијских система. Наравно, свака врста меморије има различите улазно/излазне линије које су посебне за ту меморију.

2.3 ТЕХНОЛОГИЈА ПОЛУПРОВОДНИЧКИХ МЕМОРИЈА

Постоје две опште групе у које се полупроводничке меморије сврставају - по процесу производње и по технологији. Биполарне меморије (направљене од биполарних транзистора) и MOS² меморије (направљене од MOSFET³ транзистора). Меморијски модули који се производе коришћењем различитих технологија у општем случају се разликују у електричним и физичким карактеристикама. Њихове карактеристике се

¹ *Memory enable* - приступ меморији је омогућен.

² *Metal Oxide Semiconductor* – NMOS користи N (N канал) типа MOSFET транзисторе а PMOS P (P канал) типа MOSFET транзисторе.

³ *Metal Oxide Semiconductor Field Effect Transistor*

могу међусобно упоредити уз коришћење следећих критеријума:

1. густина (капацитет у битима по интегрисаном колу),
2. брзина,
3. потрошња,
4. цена по биту,
5. имуност на сметње.

Меморије у биполарној и MOS технологији имају своје предности и мане, а према назначеним критеријумима. Уопште узевши, предност биполарних меморија је била у брзини, док је предност MOS меморије по свим другим критеријумима.

2.3.1 Биполарне меморије

Биполарне меморије користе отпорнике, диоде, и биполарне транзисторе за меморије у интегрисаном колу. Комплексност биполарних кола и потрошња, ограничавају њихов капацитет. Биполарне меморије имају најнижи капацитет од свих технологија.

Стандардна транзистор-транзистор логика (TTL¹) и Шоткијева транзистор-транзистор логика (STTL²) јесу биполарне меморије које се највише користе.

Меморије са логиком са спрегнутим емитором (ECL³) су брзе, користе биполарне транзисторе и не раде у засићењу. Потиснуле су их новије верзије меморија. Еквивалентна логика са FET⁴ транзисторима означава се као логика са спрегнутим извором (SCFL⁵).

2.3.2 CMOS меморије

У CMOS⁶ технологији за реализацију логичких функција користе се транзистори и N и P типа. Данас је CMOS технологија доминантна полупроводничка технологија која се користи у производњи микропроцесора, микроконтролера, меморија и интегрисаних кола специфичне

¹ Transistor–Transistor Logic

² Shotky TTL

³ Emitter Coupled Logic

⁴ Field Effect Transistor

⁵ Source-Coupled FET logic

⁶ Complementary Metal Oxide Semiconductor користи комплементаран и симетричан пар P (P канал) типа и N (N канал) типа MOSFET транзисторе.

2. Меморијски модули

Технологија	Брзина	Потрошња	Капацитет	Имуност на сметње	Цена
TTL/S-TTL	велика	висока	низак	ниска	висока
ECL	највећа	највећа	најнижи	најнижа	највиша
NMOS	средња (близу TTL)	ниска	највећи	висока	најнижа
CMOS	средња	најнижа	висок	највиша	ниска

Слика 2.4 Поређење технологија израде меморија

намене (ASIC¹). Главна предност CMOS над NMOS и биполарном технологијом је значајно мања потрошња и већа густина паковања у интегрисаним колима.

2.4 ROM MEMORIJE

Ова врста меморија пројектована је да може да чува податке који се не мењају уопште, или се не мењају често. За време нормалног рада нови подаци не могу се уписати у ROM меморију², али се могу прочитати из ROM меморије. Код неких ROM меморија подаци се уносе за време производње; код другог типа ROM меморија (PROM) подаци се могу касније унети. Процес уноса података назива се програмирање ROM меморија. Подаци се, једном када су уписани, у неким врстама ROM меморија не могу мењати; у другим могу се обрисати и поново програмирати.

ROM меморија се користи за чување података који се не мењају током рада система. Највећа примена ROM меморије је чување произвођачких програма³ у микрорачунару. Подаци који се чувају у ROM меморије остају сачувани и после нестанка напона напајања. Микрорачунар, после укључења, одмах започиње са извршавањем програма који је смештен у ROM меморији.

¹ *Application Specific Integrated Circuits*

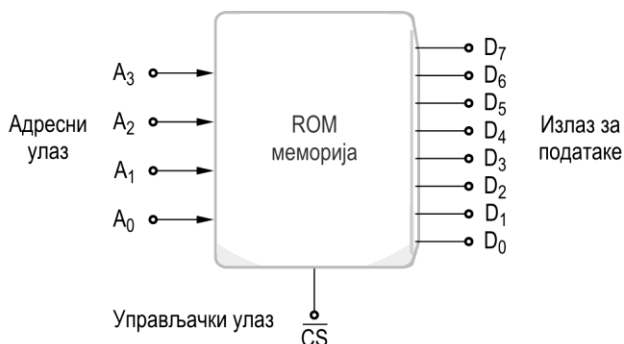
² *Read Only Memory* већ у себи садржи реч меморија. У овом уџбенику ROM користимо као ознаку меморије.

³ *Firmware*

2.4.1 Блок дијаграм ROM меморије

Типични блок дијаграм ROM меморије приказан је на слици 2.6а. Садржи три скупа сигнала: адресни, управљачки и излаз за податке. Анализирајући слику 2.6а види се да ROM меморија садржи 16 речи пошто има $2^4=16$ могућих адреса. Свака реч садржи по 8 битова (осам је излаза за податке), тако да је ово 16X8 меморија. Други начин да се опише ROM меморија је да се каже да она чува (складишти) 16 бајтова података.

а)



б)

Реч	A ₃	A ₂	A ₁	A ₀	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
0	0	0	0	0	1	1	0	1	1	1	1	0
1	0	0	0	1	0	0	1	1	1	0	1	0
2	0	0	1	0	1	0	0	0	0	1	0	1
3	0	0	1	1	1	0	1	0	1	1	1	1
4	0	1	0	0	0	0	0	1	1	0	0	1
5	0	1	0	1	0	1	1	1	1	0	1	1
6	0	1	1	0	0	0	0	0	0	0	0	0
7	0	1	1	1	1	1	1	0	1	1	0	1
8	1	0	0	0	0	0	1	1	1	1	0	0
9	1	0	0	1	1	1	1	1	1	1	1	1
10	1	0	1	0	1	0	1	1	1	0	1	1
11	1	0	1	1	1	1	0	0	0	1	1	1
12	1	1	0	0	0	0	1	0	0	1	1	1
13	1	1	0	1	0	1	1	0	1	0	1	0
14	1	1	1	0	1	1	0	1	0	0	1	0
15	1	1	1	1	0	1	0	1	1	0	1	1

в)

Реч	A ₃	A ₂	A ₁	A ₀	D ₇ –D ₀
0		0			DE
1		1			3A
2		2			85
3		3			AF
4		4			19
5		5			7B
6		6			00
7		7			ED
8		8			3C
9		9			FF
10		A			BB
11		B			C7
12		C			27
13		D			6A
14		E			D2
15		F			5B

Слика 2.5 а) Блок дијаграм ROM меморије; б) подаци у бинарном систему;
в) подаци у хексадецималном систему

2. Меморијски модули

Да би се обезбедило повезивање више ROM меморија интегрисаних компонената на исту магистралу података, излази за податке већине ROM интегрисаних кола су са отвореним колектором или са три стања.

Управљачки улаз омогућава селектовање интегрисаног кола. Ово је практично улаз који омогућава или онемогућава појаву података на излазу ROM меморије. Произвођачи користе различите ознаке за управљачке улазе, као што су на пример селектовање интегрисаног кола¹ ($\overline{CS}$) или омогућавање излаза² ($\overline{OE}$), а који омогућавају појаву сигнала на излазима за податке интегрисаног кола. Улаз је активан када је на ниском нивоу (слика 2.5а). Уочимо да не постоји улаз $R/\overline{W}$ (очитавање/уписивање) пошто се у ROM меморију не може уписивати под нормалним радним условима.

2.4.2 Операција очитавања

Претпоставимо да је ROM меморија испрограмирана подацима приказаним у табели на слици 2.5б. Постоји 16 различитих речи података које су смештене у 16 локација различитих адреса. На пример, реч податка смештена на локацији чија је адреса (адресни код) 0011_2 је 10101111_2 . Наравно, подаци су смештени у бинарној форми у ROM меморију, али као краћи користимо хексадецимални начин представљања (слици 2.5в).

Да би се прочитао податак из ROM меморије, неопходно је урадити две ствари: довести одговарајући адресни код на адресне улазе, и активирати управљачке улазе. На пример, уколико се жели прочитати податак смештен на локацији 0111_2 ROM меморије са слике 2.5 треба довести адресни код $A_3A_2A_1A_0 = 0111_2$ на адресне улазе, и активан (низак ниво) а на $\overline{CS}$ улаз. Адресни код биће декодован унутар ROM меморије да би се одабрала жељена реч, 1110110011_2 , која ће се појавити на излазима за податке D_7-D_0 . Уколико је $\overline{CS}$ на високом нивоу, биће онемогућена појава сигнала на излазу ROM меморије, тј. излази за податке D_7-D_0 биће у стању високе импедансе.

2.5 АРХИТЕКТУРА ROM МЕМОРИЈЕ

Интерна архитектура (структура) ROM интегрисаног кола веома је комплексна, и са становишта овог уџбеника није је неопходно детаљно

¹ *Chip Select*

² *Output Enable*

познавати. Корисно је анализирати поједностављен дијаграм интерне архитектуре за 16X8 ROM меморије, који је приказан на слици на слици 2.6. Постоје четири основна дела: декодер редова, декодер колона, меморијски регистар¹ и излазни бафер.

2.5.1 Меморијски регистар

Меморијски регистар је део у коме су чувају подаци који су програмирани у ROM меморији. Број меморијских ћелија у регистру једнак је дужини меморијске речи. Због ефикасније реализације регистри су распоређени као квадратна матрица. Позиција сваког регистра може се одредити према одређеној колони и врсти. На пример, регистар 0 је у врсти 0 и колони 0 (ознака 0/0), а регистар 9 је у врсти 9 и колони 9 (ознака 9/9).

Осам линија за податке сваког регистра везано је за интерну магистралу података која се простира кроз цело интегрисано коло. Сваки регистар има два улаза за селектовање (E^2), који треба да су на високом нивоу, да би се подаци из регистра поставили на магистралу података.

2.5.2 Адресни декодер

Довођењем адресног кода $A_3A_2A_1A_0$ на адресне улазе одређено је који регистар из групе ће бити селектован и поставити 8-битну реч податка на магистралу података. Адресне линије A_1A_0 доводе се на декодер 1 од 4, који активира линију за селектовање редова. Адресне линије A_3A_2 доводе се на други декодер 1 од 4, који активира линију за селектовање колона. Само један регистар ће се наћи и у селектованом реду и у селектованој колони, и то је онај који смо желели да селектујемо.

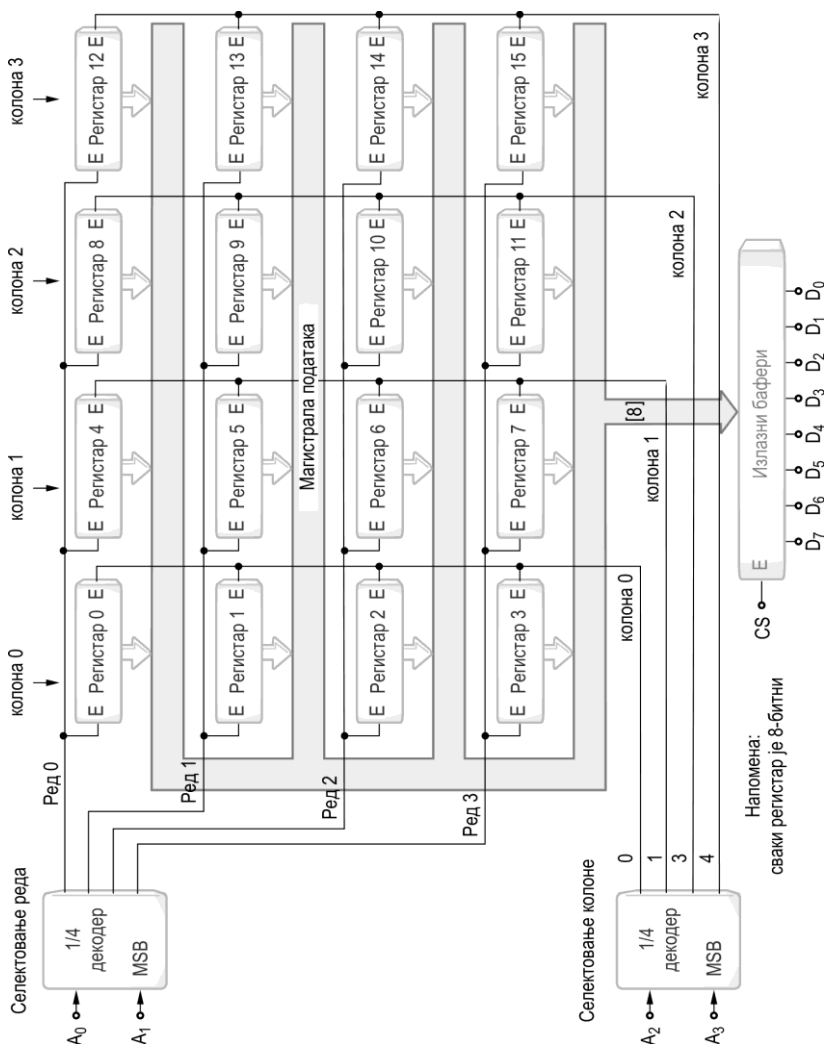
2.5.3 Излазни бафери

Регистар који је селектован адресним улазима поставиће податке на магистралу података. Ови подаци се уписују у излазни бафер, који ће проследити податке ка спољашњем уређају, под претпоставком да је на ниском нивоу. Уколико је на високом нивоу, излазни бафери су у стању високе импедансе.

¹ Register Array

² Enable

2. Меморијски модули



Слика 2.6 Архитектура ROM меморије 16X8

Архитектура већине интегрисаних ROM меморија слична је архитектури ROM меморије приказане на слици 2.6. У зависности од капацитета меморије регистри у неким ROM модулима неће бити организовани у квадратну матрицу.

На пример, Intel 2708 је NMOS/CMOS ROM меморија капацитета 1024 8-битних речи. Њених 1024 регистра организовано је у матрицу 64X16.

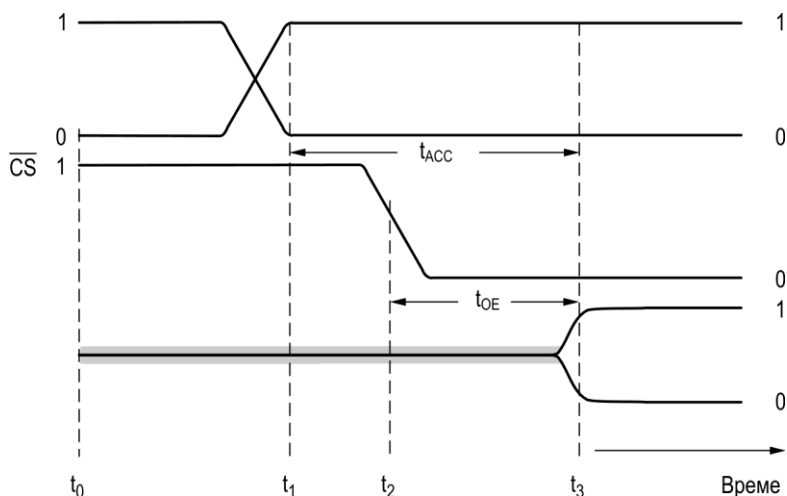
2.6 ВРЕМЕНСКИ ДИЈАГРАМ ОЧИТАВАЊА ROM MEMORIJE

Постоји временска разлика (слика 2.7) између тренутка када се сигнали доведу на адресни улаз ROM меморије и тренутка када се појаве подаци на његовим излазима. Ова временска разлика последица је пропагације (простирања) и назива се време приступа (t_{ACC}). Мера је брзине рада ROM меморије.

Други важан параметар је време селектовања излаза¹ (t_{OE}). Представља кашњење између тренутка када је сигнал на улазу активан и тренутка када се на његовом излазу појаве важећи подаци.

2.7 ВРСТЕ ИНТЕГРИСАНИХ ROM MEMORIJA

Укратко ћемо објаснити различите врсте интегрисаних ROM меморија, да бисмо видели како се разликују у начину програмирања и способности брисања и поновног програмирања.



Слика 2.7 Типичан временски дијаграм за операцију читавања ROM меморије

¹ Output Enable

2. Меморијски модули

ROM меморија која се програмира преко маске (MPROM¹) врста је ROM меморије код које садржај уписује произвођач, а према спецификацији корисника. Фотографски негатив (маска) користи се за стварање електричних веза у интегрисаном колу. За различите информације које се уписују у ROM меморију потребне су одговарајуће маске. Због високе цене и појаве нових метода програмирања меморија произвођачи све мање користе MPROM технологију.

Меморија који се може програмирати (PROM²) јесте врста меморије коју корисник може сам да програмира пошто меморија није испрограмирана за време производње. Једном када се испрограмира садржај се не може брисати или поново програмирати. Уколико програм у PROM меморији има грешку или га је потребно мењати, PROM меморија се мора заменити новом. Брзина биполарних PROM меморија (време приступа) је 30-50ns.

Избрисиву PROM меморију (EPROM³) корисник може програмирати, брисати и поново програмирати већи број пута (слика 2.8). Једном када је испрограмирана, EPROM меморија податке може врло дуго (више од 10 година) да чува. Када се EPROM меморија програмира онда се на одговарајуће улазе доводе специјални напонски нивоии одређеног трајања. Брисање се обавља излагањем ултравиолетном зрачењу⁴. Време приступа је од 200-250ns. EPROM меморија има два важна недостатка. Прво, да би се обрисала и поново програмирала мора се скинути са подножја. Друго, кад је већ обрисана, комплетан садржај меморије је уништен; поново се мора програмирати чак и кад је потребно само једну меморијску реч изменити.

Код PROM меморије чији се садржај брише електричним путем (EEPROM⁵) (као и флеш⁶ меморије) може се обрисати и поново програмирати било која поједина реч меморије. Пошто се електричним путем брише и

¹ Mask-Programmed ROM

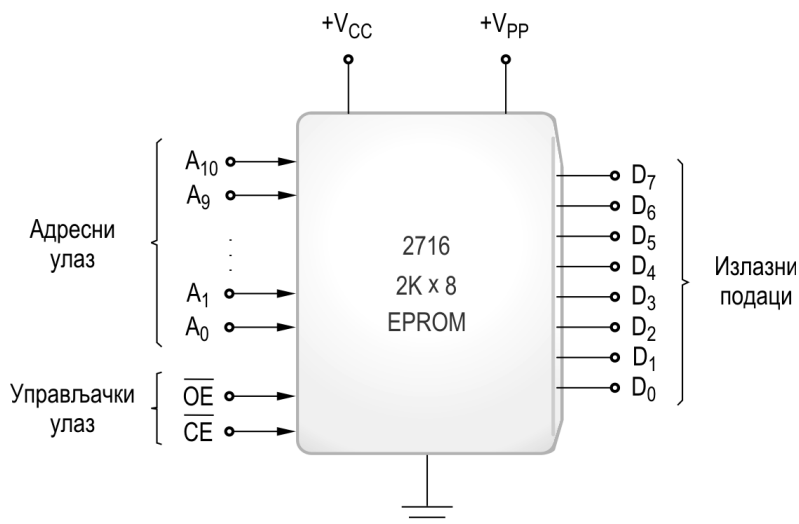
² Programable ROM

³ Erasable PROM

⁴ Ове меморије се означавају UVEPROM Ultraviolet Erasable Programmable Read Only Memory

⁵ Electrical Erasable PROM - E²PROM

⁶ Flash



Слика 2.8 Симбол за 2716 EPROM

програмира, не захтева се скидање са процесорске плоче¹.

Брзина уписа Е²PROM меморије је око 2ms по податку, а време приступа је мање од 200ns. Код флеш меморије уписивање је око 2ms по блоку података, а време приступа такође мање од 200ns.

2.8 ПОЛУПРОВОДНИЧКА RAM МЕМОРИЈА

Полупроводничка RAM меморија је врста меморије код које се свакој локацији подједнако брзо приступа. Када је реч о полупроводничким меморијама под појмом RAM подразумевамо меморије у које се може и уписивати и из којих се може читавати - RWM² меморија. Уобичајено је да се под RAM меморијом подразумева полупроводничка RWM меморија.

RAM се користи за привремено смештање програма и података. Како рачунар извршава програм садржај адресних локација RAM меморије се непрекидно мења. Да би рад рачунара био што бржи потребно је и брзо читавање и уписивање у RAM меморију.

¹ On board

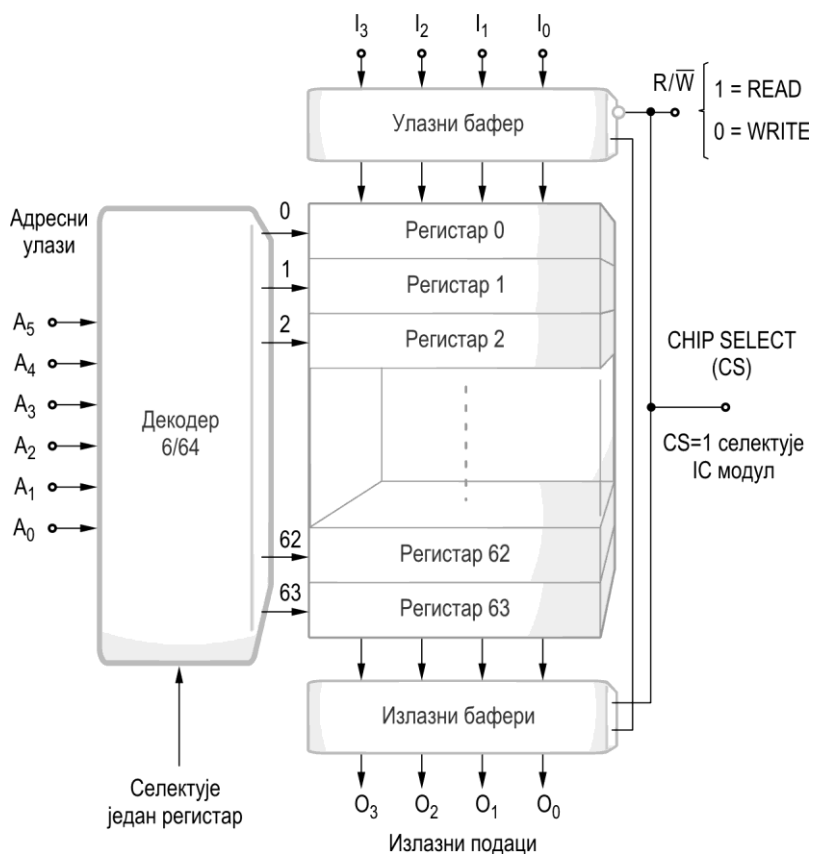
² Read Write Memory

2. Меморијски модули

Основни недостатак RAM меморије је што је то избрисива меморија, тј. што губи све податке када дође до нестанка напона напајања. Важна предност RAM меморије је да подаци могу да се уписују и читавају са подједнаком лакоћом.

2.8.1 Архитектура RAM меморија

Посматраћемо RAM меморију као низ регистара, од којих се у сваком налази једна реч податка и сваки регистар има јединствену адресу. Капацитет RAM меморије и величина могу се проширити комбиновањем меморијских, интегрисаних модула.



Слика 2.9 Интерна организација RAM меморије капацитета 64X 4

Слика 2.9 показује поједностављену архитектуру RAM меморије у коју се смештају 64 речи од по 4 бита (тј. меморија је 64×4). Меморијске локације означене су адресама од 0 до 63_{10} . Да би се селектовала једна од 64 локације (за уписивање или читавање), на улаз декодерског кола потребно је довести одговарајућу бинарну адресу (адресни код). Пошто је $64=2^6$, декодер захтева адресни код од 6 бита. Сваки адресни код активира један, одређени декодерски излаз, који селекује одговарајући регистар. На пример, на улаз је постављен адресни код $A_5 A_4 A_3 A_2 A_1 A_0 = 011010_2$. Пошто је $011010_2 = 26_{10}$, декодерски излаз 26 биће на високом нивоу, и селектоваће регистар 26, било за операцију читавања било за операцију уписивања.

Статичка RAM меморија

Операције које смо до сада описивали односиле су се на статичку RAM меморију. Статичка RAM меморија је она која чува податке док је напон напајања присутан на интегрисаном модулу. Уколико је напајање присутно статичке меморијске ћелије (печ), остају неограничено у одређеном стању (уписаног податка).

Статичка RAM меморија производи се у биполарној и MOS технологији. Као што је већ претходним одељцима истакнуто, предност биполарних меморија је у брзини а MOS меморија у већем капацитету. Да би се сачували подаци користе се батерије¹ које обезбеђују стално напајање. Брзине „обичних” статичких меморија су око 100ns, а кеш² меморија 20-25ns.

2.8.2 Динамичка RAM меморија

Основна разлика између статичке и динамичке RAM меморије је у томе што динамичка меморијска ћелија чува податке ограничено време, после чега је податак изгубљен. Да би се подаци дуже сачували у динамичкој меморији захтева се освежавање у правилним временским размацима. Потреба за освежавањем је недостатак динамичких RAM меморија. Предност динамичких RAM меморија је велика густина паковања која омогућава прављење интегрисаних меморија великог капацитета. Цена динамичких меморија је ниска што је последица једноставности меморијске ћелије. Брзина приступа је 20-80ns, а циклус освежавања је 120-150ns.

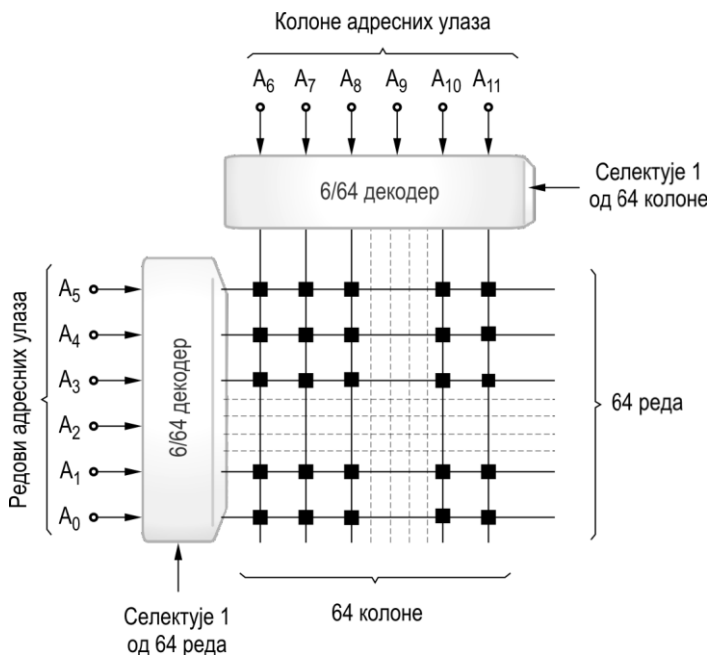
¹ Back up

² Cash

Структура динамичке RAM меморије

Већина динамичких RAM меморија (DRAM¹) може се представити као матрица од регистара величине једног бита. Садржај се чува као наелектрисање малог кондензатора. Пошто се, због пражњења кондензатора (цурења), наелектрисање мења потребно је довољно често обављати освежавање.

На слици 2.10 приказано је 4096 ћелија, организованих као матрица 64X64. Свака ћелија заузима одређену врсту и одређену колону матрице. Дванаест адресних улаза потребно је за селектовање једне ћелије: битови мање тежине A_0 - A_5 селектују врсту, а битови веће тежине селектују колону. Сваки 12-битни адресни код селектује јединствену ћелију у коју се уписује или из које се очитава податак. Структура на слици 2.10 је за 4kX1 DRAM (тј. чува 4096 1-битних речи).



Слика 2.10 Организација за динамичке RAM меморије 4kX1

¹ Dynamic RAM

2.9 ПРОШИРИВАЊЕ ДУЖИНЕ РЕЧИ И КАПАЦИТЕТА

У великом броју примена интегрисаних меморија захтевају се капацитет и дужине речи коју један интегрисани модул не задовољава. Неколико меморијских модула мора се комбиновати да би се добили жељени капацитет и дужина речи. Анализираћемо на пар примера како се то може урадити.

2.9.1 Проширивање дужине речи

Претпоставимо да нам је потребна меморија у коју може да се смести 16 8-битних речи, а на располагању нам је RAM меморија 16X4 са заједничким I/O линијама. Два оваква интегрисана модула можемо комбиновати да бисмо добили жељену меморију (слика 2.11).

Пошто сваки модул може да складишти 16 4-битних речи а желимо да сместимо 16 8-битних речи, сваки модул може да складишти половину од укупног броја речи. Другим речима, RAM-0 садржи четири бита вишег реда а RAM-1 четири бита нижег реда сваке од 16 речи. Комплетна 8-битна реч може се добити на излазу RAM меморије који је везан за магистралу података.

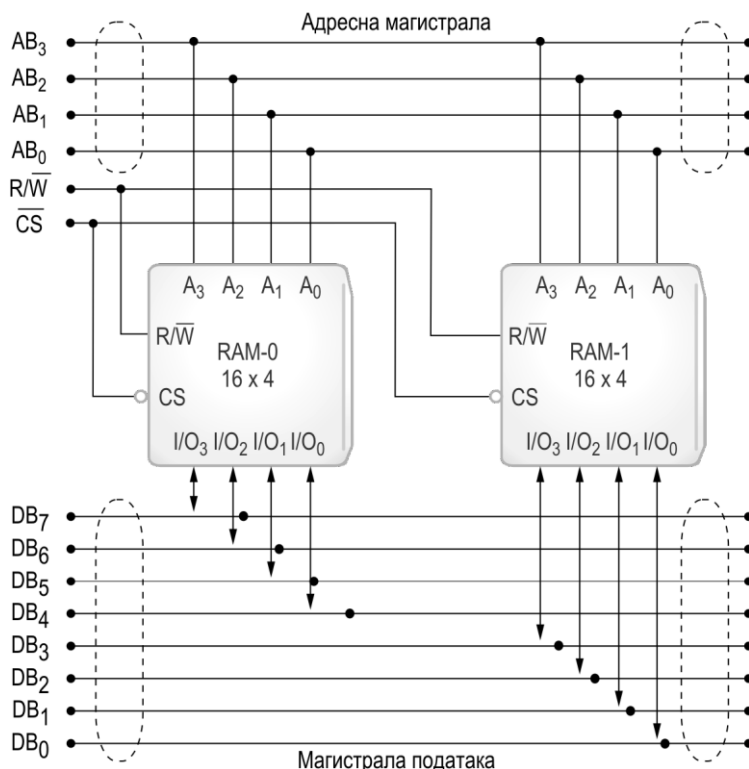
Било која од 16 речи селекује се довођењем одговарајућег адресног кода на 4-линијску адресну магистралу (AB_3, AB_2, AB_1, AB_0).

Једном када је адреса селекована може се са те адресе под контролом $R/\overline{W}$ и $\overline{CS}$ линија, уписивати или читавати. За читавање, линија $R/\overline{W}$ треба да буде на високом нивоу, а $\overline{CS}$ на ниском нивоу. То доводи до тога да се I/O линије RAM меморије понашају као излазне. RAM-0 поставља селековану 4-битну реч на више линије магистрале података, а RAM-1 поставља селековану 4-битну реч на четири доње линије магистрале. Магистрала података садржи потпуну 8-битну реч, која се сада може послати до других модула (нпр. регистара, процесора...).

Код уписа података контролне линије су на ниском нивоу ($R/\overline{W}=0$ и $\overline{CS}=0$), па се I/O линије RAM меморије понашају као улазне. На магистралу података поставља се са спољашњег уређаја 8-битна реч која се уписује. Виша четири бита биће уписана у селековану локацију RAM-0, а нижа 4 бита би ће уписана у RAM-1.

У суштини, комбинација две интегрисане RAM меморије понаша се као једна интегрисана RAM меморија капацитета 16X8.

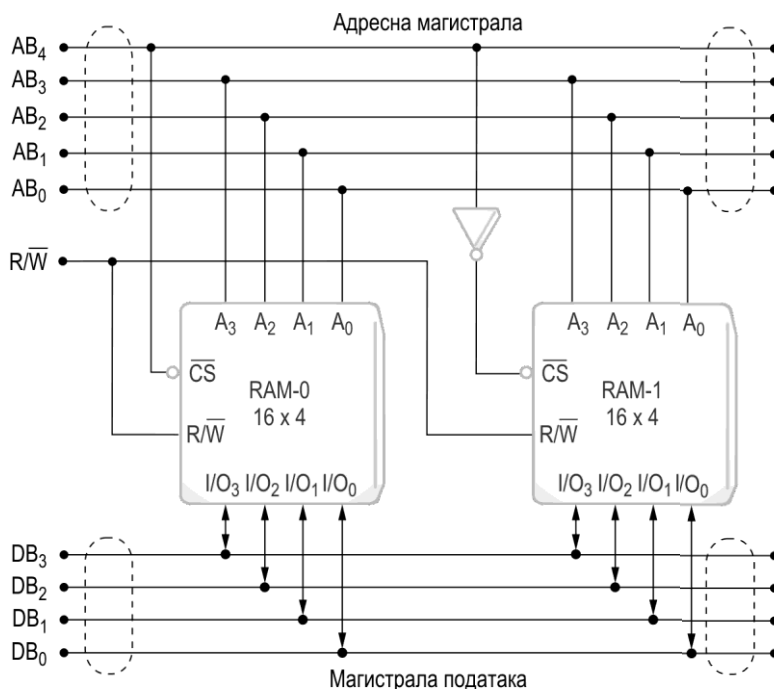
2. Меморијски модули



Слика 2.11 Комбиновање две 16 X 4 RAM меморије да би се добила 16 X 8 меморија

2.9.2 Проширивање капацитета

Претпоставимо да нам је потребна меморија која може да складишти 32 4-битне речи и да имамо на располагању 16X4 модуле. Жељену меморију можемо добити комбинацијом више 16X4 модула, као што је приказано на слици 2.11. Сваки од RAM меморијских модула користи се за складиштење 16 4-битних речи. I/O улази за податке сваког RAM модула повезани су на заједничку 4-линијску магистралу. У датом тренутку може бити селектован само један од RAM модула, тако да не долази до сукобљавања на магистрали података (истовременог појављивања података из различитих модула). Ово се обезбеђује тако што се на $\overline{CS}$ улазе доводе различити логички сигнали.



Слика 2.12 Комбиновање два 16 X 4 меморијска модула
да би се добила 32X 4 меморија

Пошто је укупан капацитет овог меморијског модула 32X4, потребне су 32 различите адресе, односно пет линија адресне магистрале. Виша адресна линија AB_4 користи се за селектовање једног од RAM модула, (преко улаза) у који се или уписује или из кога се читава. Друге четири адресне линије AB_0 - AB_3 користе се за селектовање једне од 16 меморијских локација из селектованог RAM модула.

Када је $AB_4=0$, $\overline{CS}$ сигнал који се доводи на улаз RAM-0 модул га селекује за операцију читавања или уписивања. Даље, било којој адресној локацији у оквиру RAM-0 може се приступити преко AB_3 - AB_0 линија. Четири адресне линије из опсега 0000_2 до 1111_2 селекују жељену адресну локацију. Опсег адреса локација у оквиру RAM-0 је:

$$AB_4 AB_3 AB_2 AB_1 AB_0 = 10000_2 \text{ до } 11111_2$$

2. Меморијски модули

Видели смо како се може вршити проширивање дужине речи и капацитета меморије комбиновањем меморијских кола. У многим меморијским системима захтевају се оба типа проширивања. За веће капацитете потребно је употребити одговарајуће компоненте са већим бројем адресних линија.

2.10 ПИТАЊА И ЗАДАЦИ

1. Дефинисати следеће појмове:
а) меморијска ћелија б) меморијска реч в) адреса
г) бајт д) време приступа
2. Одређена меморија има капацитет 10kX24.
а) Колико битова садржи једна реч податка?
б) Колико речи података се може ускладиштити?
в) Колико меморијских ћелија садржи ова меморија?
3. Одређена меморија има капацитет 10MX32.
а) Колико битова садржи једна реч податка?
б) Колико речи података се може ускладиштити?
в) Колико меморијских ћелија садржи ова меморија?
4. Коју вредност треба довести на адресне линије A_0, A_1, A_2, A_3 на слици 2.6 да би се читала вредност из регистра 14?
5. Која је основна предност EEPROM меморије над EPROM меморијом?
6. Интегрисано коло MCM6209C је 64kx4 статичка RAM меморија. Колико интегрисаних кола је потребно за модул 4Mx8.
7. Тачно-нетачно: када се меморијска интегрисана кола комбинују да формирају модул за повећање величине речи података или капацитета меморије њихови CS улази увек су међусобно повезани. Објаснити.
8. Тачно-нетачно: када се меморијска интегрисана кола повезују у модул већег капацитета линије података интегрисаних кола везују се за исте линије магистрале података. Објаснити.
9. Нацртати и објаснити архитектуру ROM меморије?
10. Нацртати и објаснити структуру динамичке RAM меморије?

3. Увод у рачунаре

Осамдесете и деведесете године XX века спомињу се као године рачунарске револуције. Тада су се произвели рачунари чије су брзине и рачунарска снага биле хиљадама пута веће него што је то било код првих комерцијалних рачунара. Цене и димензије ових рачунара нису се разликовале од оних за писаће машине. Ова револуција наступила је као резултат развоја интегрисаних кола која повезује велики број електронских компонената на једном силиконском колу, као што су кола са високим степеном интеграције (LSI¹) и кола се веома високим степеном интеграције (VLSI²). Такав степен интеграције може да постави велики број транзистора - сада и милионе комада на један једини елеменат тзв. чип. Овакав технолошки напредак довео је до тога да је могуће направити цео мозак рачунара као једно интегрисано коло које се назива микропроцесор. Додатак од релативно малог броја других чипова формира комплетан микрорачунар.

Од првог дигиталног рачунара из 1940. до данашњих моћних рачунара и револуционарног микрорачунара, постоји веома мала промена у основним принципима рада. Иако је технологија отишла далеко од времена када су први рачунари ушли у употребу, наши модерни рачунари још увек раде са јединицама и нулама, користећи исте основне логичке операције као и њихови претходници. У овом поглављу размотрићемо концепте, принципе и операције који су заједнички за све типове рачунара.

3.1 ШТА РАЧУНАР МОЖЕ ДА РАДИ?

У највећем броју случајева човек може да ради све што може и рачунар, али рачунар то ради много већом брзином и тачношћу. Ово је тачно без обзира на чињеницу да рачунари извршавају само једну по једну операцију. Човек може да узме листу од 10 бројева и израчуна збир свих,

¹ Large Scale Integration

² Very Large Scale Integration

једном операцијом читавајући бројеве један за другим и додајући један другом колону по колону. Рачунар с друге стране може да сабере само два броја, тако да би сабирање десет бројева захтевало девет додатних корака. Наравно, чињеница да је рачунару потребно мање од микросекунде по кораку, објашњава његову ефикасност.

Рачунар је бржи и тачнији од човека али, за разлику од човека, мора му се дати комплетан скуп наредби (инструкција) које му говоре шта тачно да ради у сваком кораку. Овај скуп инструкција, назван програм, направљен је од стране једног или више људи за сваки задатак који је дат рачунару да уради. Програми су смештени у меморији рачунара у бинарно кодираном облику, при чему свака инструкција има свој јединствен код (шифру - ознаку). Рачунар читава из меморије кодове и изводи операције које одговарају тим кодовима.

Рачунари су нашли своје место и примењују се у свим областима у којима то раније није било изводљиво из економских разлога (нпр. за играчке, у домаћинству...). Брзи пораст у броју и разноврсности рачунарских апликација, као и пад цена поставио их је на незаменљиво место у свим областима као што су пословне апликације, процесна контрола, наука, технологија, медицина, образовање, домаћинства...

3.2 КАКО РАЧУНАРИ РАЗМИШЉАЈУ?

Раунари не размишљају! Рачунару програмер обезбеђује програм који се састоји од инструкција и података који специфицирају сваки детаљ о томе шта да ради и када да ради. Рачунар је само брза машина која може да оперише подацима, да рачуна, да доноси закључке на основу упоређивања - све под контролом програма.

Уколико програмер направи грешку у програму или стави погрешне податке, рачунар ће, апсолутно тачно, дати погрешан резултат.

Поставља се питање: „Како рачунар изводи програм који се састоји од инструкција?” Одговор на ово питање добија се уколико се погледа рачунарска архитектура (уређеност њихових различитих елемената) и прође корак по корак процес који рачунар пролази у извршавању програма.

3.3 КОЛИКО ТИПОВА РАЧУНАРА ПОСТОЈИ?

Као што је напоменуто сви рачунари раде на врло сличан начин. Постоји неколико фактора који чине да се рачунари разликују један од другог. Ови фактори су физичка величина, цена, брзина, дужина речи...

Физичка величина и цена саме по себи су јасне. Брзина се односи на брзину којом рачунар може да извршава инструкције. Рачунарска реч се односи на број битова које садржи основна јединица података коју рачунар може да обради и смести. На пример, рачунар са 16-битном величином речи може да обрађује податке величине 16 битова. Рачунар ће тада бити окарактерисан као 16-битни рачунар.

Примарно, на основу ових фактора рачунаре можемо да класификујемо у три категорије: микрорачунари, рачунари средње величине и велики рачунари¹.

3.3.1 Микрорачунари

Без обзира што не постоји јединствена дефиниција микрорачунар је термин који се користи за рачунар са микропроцесором као процесором. Друга карактеристика ових рачунара је да заузимају мали простор.

Савремени стони рачунари, конзоле за видео игрице, преносиви рачунари², преносиви рачунари изузетно малих димензија³ и различите друге врсте уређаја малих димензија⁴ укључујући мобилне телефоне, калкулаторе као и индустријске системе специјалне намене (затворене системе)⁵ могу се сматрати микрорачунарима, а према горе поменутој дефиницији.

Ови чланови рачунарске породице су најмањи и најјефтинији. Њихова мала димензија је резултат LSI и VLSI технологије. Микрорачунари користе интегрисана кола у CMOS технологији. Могу да раде са величином речи од 4, 8, 16, 32 и 64 бита.

¹ Mainframe

² Laptop computers

³ Tablet PC

⁴ Handheld devices

⁵ Embedded systems

3.3.2 Рачунари средње величине

У категорију рачунара средње величине¹ данас се убрајају радне станице и сервери. Некада се за ову категорију рачунара (између микрорачунара и великих рачунара) користио назив мини рачунари. Користе се у процесној контроли, истраживањима, клијент-сервер апликацијама, рачунарским мрежама...

3.3.3 Велики рачунари

Велики рачунари су били највећих димензија, најскупљи, и најмоћнији рачунари. Могли су се наћи у великим корпорацијама, банкама, универзитетима и истраживачким лабораторијама. Данас их замењује већи број међусобно повезаних сервера, смештених заједно (сервер сале) уз системе за складиштење података.

3.3.4 Супер рачунар

Супер рачунар је водећи рачунар посматрано у односу на брзину обраде података. Користи се у решавању проблема квантне физике, метеоролошким прогнозама, климатским истраживањима, молекуларном моделовању, симулацијама у физици, криптоанализама... Да би се постигла довољно велика брзина у обради велике количине података користе се векторска обрада² и SIMD³ процесор. Већина супер рачунара има сличну архитектуру. Свака од њих је скуп (кластер) MIMD⁴ мултипроцесора а сваки од процесора је SIMD процесор. Супер рачунари се значајно међусобно разликују у зависности од броја мултипроцесора по кластеру, броја процесора по мултипроцесору и броја истовремених инструкција које један SIMD процесор може да изврши. У оквиру ове хијерархије постоји:

- Кластер рачунара⁵ који је скуп рачунара умрежених брзим везама. Сваки од рачунара је под посебним оперативним системом.
- MIMD је техника која обезбеђује паралелан рад. Рачунари који

¹ *Midrange Systems*

² Подразумева истовремене математичке операције над више података. Супротно векторској је скаларна обрада која подразумева математичке операције над једним податком у једном тренутку.

³ *Single Instruction, Multiple Data*

⁴ *Multiple Instruction stream, Multiple Data stream*

⁵ *Computer cluster*

користе MIMD имају већи број процесора који раде асинхронно и независно. У сваком тренутку различити процесори могу да извршавају различите инструкције на различитим деловима података.

- Мултипроцесорски рачунар¹ је рачунар који ради под једним оперативним системом и који користи више од једног процесора. На софтвер на апликационом нивоу број процесорских јединица нема никаквог утицаја.
- SIMD процесор извршава исту инструкцију над више скупова података. Процесор може да буде процесор опште намене или процесор специјалне намене – векторски процесор².

3.4 ОСНОВНА СТРУКТУРА РАЧУНАРА

Без обзира на разлике у перформансама између различитих врста рачунара сваки рачунар има пет основних јединица: аритметичко/логичку јединицу, меморијску јединицу, управљачку јединицу, улазну јединицу и излазну јединицу. Слика 3.1 је блок дијаграм који представља општи рачунарски систем и међусобну везу различитих јединица (блокова).

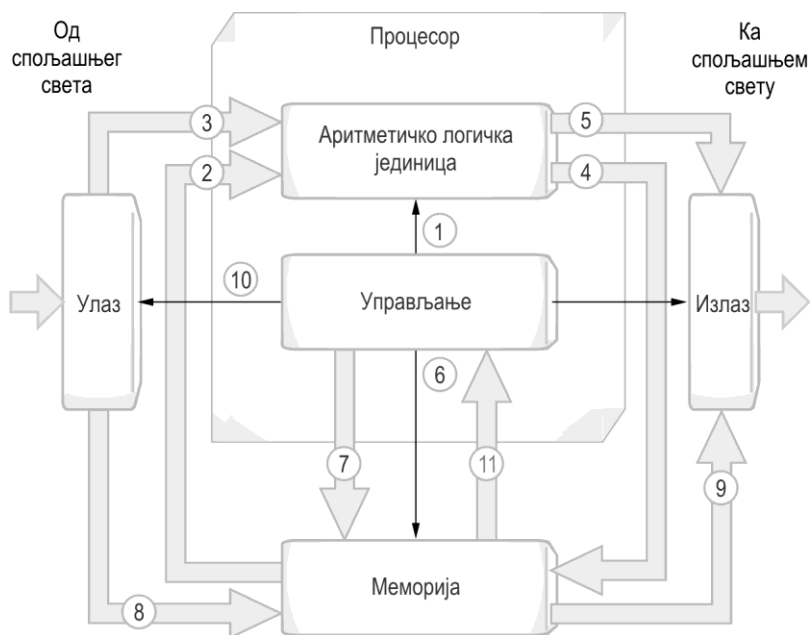
Уочавамо да је управљачка јединица у центру дијаграма пошто функционише као мозак рачунара, управљајући радом свих других делова. Стрелице у дијаграму показују правац којим теку подаци или управљачки сигнали. Велика стрелица показује групу паралелних линија која представља реч података, инструкцију или адресу; мања стрелица приказује управљачке сигнале, који су обично једна или неколико сигналних линија. У следећим одељцима користићемо уоквирене бројеве поред стрелица ради описа појединих јединица.

3.4.1 Меморијска јединица

У овој јединици смештена је група битова (речи) која репрезентује инструкцију коју рачунар треба да изврши, и податке са којима треба радити у програму. Меморија служи такође и као привремено складиште

¹ *Multiprocessing computer*

² *Vector processor*



Слика 3.1 Општи блок дијаграм рачунарског система

резултата операција које извршава аритметичко/логичка јединица (ALU¹) - стрелица 4. Меморијском јединицом управља управљачка јединица, која сигнализира операције читавања (READ) или уписивања (WRITE) - стрелица 6 и обезбеђује одговарајућу меморијску адресу - стрелица 7. Подаци који треба да се упишу у меморију могу да дођу из аритметичко/логичке јединице или из улазне јединице - стрелица 8, а све под контролом управљачке јединице. Подаци који су учитани из меморије могу да се шаљу ка аритметичко/логичкој јединици - стрелица 2 или ка излазној јединици - стрелица 9. Као што је већ напоменуто рачунар има интерну и екстерну меморију. Интерна меморија је релативно малог капацитета, брза, и смешта програме и податке које рачунар тренутно извршава (краткотрајно смештање). Полупроводничке RAM² и ROM³

¹ Arithmetic Logic Unit

² Random Access Memory

³ Read Only Memory

меморије обично се користе као интерна меморија. Екстерна меморија је меморија великог капацитета, која се користи за смештање програма и података које рачунар тренутно не користи и смешта на дужи период.

3.4.2 Аритметичко/логичка јединица

Аритметичко/логичка јединица (ALU), обавља аритметичке и логичке операције над подацима. Управљачка јединица шаље сигнале аритметичко/логичкој јединици која одређује које операције ће се извршавати над подацима (стрелица 1). Управљачка јединица одређује и извориште података - меморијску јединицу (стрелица 2) или улазну јединицу (стрелица 3). Резултат операције може бити послат у меморију ради ускладиштења (стрелица 4) или ка спољашњој јединици (стрелица 5).

3.4.3 Улазна јединица

Улазна јединица се састоји од склопова који омогућавају да подаци и информације који долазе споља улазе у интерну меморију рачунара или аритметичко/логичку јединицу (стрелице 3 и 8). Ови склопови обично се називају периферије, пошто су они организационо одвојени од склопова који чине мозак рачунара. Неке типичне улазне периферије су тастатура, аналогно/дигитални конвертори... Персонални рачунари користе тастатуру као улазни уређај. Некада се и магнетна трака и диск јединице посматрају као улазне периферије. Њихова главна функција је да чувају (складиште) податке и програме који се, после команди из управљачке јединице, преносе у интерну меморију рачунара.

3.4.4 Излазна јединица

Излазна јединица састоји се од периферијских уређаја који преносе податке и информације из интерне меморије или аритметичко/логичке јединице ка спољашњем „свету” (стрелице 5 и 9). Типичне излазне периферије су LED¹ екран, штампачи, видео монитори, ТВ пријемници дигитално/аналогне конвертори...

Диск меморијске јединице могу се посматрати и као излазне периферије (јединице масовне меморије); подаци и програм из интерне меморије смештају се у једној од спољњих меморија на трајно чување.

¹ Light Emitting Diode

3.4.5 Управљачка јединица

Функција ове јединице је сада јасна. Она управља радом свих других делова рачунара обезбеђујући временске и управљачке сигнале. Она је као диригент у оркестру који води рачуна да је сваки од инструмената у доброј синхронизацији. Управљачка јединица узима инструкцију из меморије тако што шаље адресу (стрелица 7) и команду за читавање (стрелица 6) ка меморијској јединици. Инструкција (реч) из меморије долази у управљачку јединицу (стрелица 11). Ова инструкцијска реч, која је у бинарно кодираном облику, декодује се у управљачкој јединици (преко логичких кола) да би се одредило која операција треба да се изведе. Када се операције одреде управљачка јединица генерише секвенцу сигнала који је извршавају.

Можемо рећи да управљачка јединица има задатак да узме, декодује и изврши инструкције које су смештене у меморији.



Слика 3.2 а) поједностављени дијаграм рачунара; б) дијаграм микрорачунара

3.4.6 Процесор

Аритметичко/логичка и управљачка јединица обично су повезане у једну целину која се назива централна процесорска јединица (CPU¹) или процесор. Он је „мозак“ рачунара јер комбинује кола која генеришу све управљачке сигнале који су потребни за извршавање инструкција са колима која стварно извршавају операције које инструкције захтевају.

3.4.7 Поједностављен дијаграм рачунарског система

Пошто смо се упознали са функцијама пет основних јединица рачунарског система можемо комбиновати неке од јединица да би добили једноставнији дијаграм. Слика 3.2 показује како се рачунар може представити као три блока. Процесор замењује аритметичко/логичку и управљачку јединицу. Улазно/излазни блок је комбинација улазних и излазних јединица.

3.4.8 Микропроцесор, процесор и микрорачунар

У микрорачунару процесор је интегрисано коло које се назива микропроцесор. Термини „процесор“ и „микропроцесор“ користе се често као синоними. У рачунарству је уобичајно да се микропроцесор назива микропроцесорска јединица (MPU²).

Сумираћемо следеће чињенице:

1. Процесор је комбинација аритметичко/логичке и управљачке јединице.
2. Када је процесор интегрисано коло назива се микропроцесор и означавамо га са MPU.
3. Процесор повезан са меморијом и улазно излазним јединицама представља микрорачунар.

3.5 МИКРОПРОЦЕСОР, МИКРОКОНТРОЛЕР И ПРОЦЕСОР

Микропроцесор је интегрисано коло које садржи сва потребна кола за извршавање функција процесора. Постоји више различитих врста микропроцесора које производе различити произвођачи. Иако сви они

¹ Central Processing Unit

² Micro Processing Unit

3. Увод у рачунаре

обављају суштински исте функције, сваки од њих има јединствене карактеристике које их чине различитим од других микропроцесора. Основне разлике су у дужини речи, броју и типу инструкција које могу да се изврше, типу спољашњих управљачких сигнала и величини меморије коју могу да адресирају.

У овом уџбенику биће анализиран рад процесора једног 8-битног микроконтролера¹ MC68HC11 који је базиран на микропроцесору Motorola MC6800. Микроконтролер садржи процесор, меморију и улазно излазна кола, часовнике и др. у једном интегрисаном колу (чипу). У првом делу уџбеника биће описан рад поједностављеног модела процесора микроконтролера MC68HC11 и његове комуникације са улазно излазним јединицама и меморијом. Касније ће бити анализиран и рад микроконтролера MC68HC11. Важно је напоменути да све што се научи о раду микроконтролера MC68HC11 може се применити на већину микроконтролера и микрорачунара.

3.6 РАЧУНАРСКА РЕЧ

Претходна анализа међусобног односа различитих јединица рачунарског система поједностављена је, у циљу лакшег разумевања. Да бисмо ову интеракцију објаснили на мало детаљнији начин, морамо се подсетити на неколико врста информација које се преносе и са којима се стално ради у оквиру рачунара.

Као што је напоменуто, основна јединица информације у рачунару је реч. Иако је реч направљена од неколико битова, рачунар третира реч као једну целину и смешта сваку реч у одређену меморијску локацију. Реч у микропроцесорима је обично дугачка 8 (бајт), 16 или 32 бита.

Реч смештена у рачунарској меморији може да представља неколико типова информација:

- бинарне нумеричке податке,
- кодиране податке,
- адресе које су део инструкција.

¹ MC68HC11 (MC6811 или HC11) је фамилија микроконтролера (μC) које је пројектовала и произвођила Motorola, а сада их производи *Freescale Semiconductor*. Детаљнији подаци могу се наћи на Интернет страници: <http://www.freescale.com/webapp/sps/site/taxonomy.jsp?nodeId=01624684498635>.

3.7 БИНАРНА РЕЧ ПОДАТАКА

Ово су речи које једноставно представљају нумеричку величину у бинарном бројном систему. На пример локација у меморији 8-битног (један бајт) микрорачунара може да садржи реч 01110011_2 , што одговара броју 115_{10} . Пример 16-битне речи података је:

$$1010000101001001_2$$

што је еквивалент за 41289_{10}

Великом дужином речи може се представити велики број података. Са 8-битном речи највећа реч података (11111111_2) еквивалентна је са $2^8-1=255_{10}$. Са 16-битном речи највећа вредност је $2^{16}-1=65535_{10}$. Са 32 бита (4 бајта) може се представити број већи од 4 билиона.

3.7.1 Реч податка са знаком

Рачунар се не би могао употребити ако би радио само са позитивним бројевима. Због тога рачунари за представљање бројева користе други комплемент са знаком. Подсетимо се да је бит са највећом тежином - бит знака 0 за позитиван и 1 за негативан број, тако да ће бројеви $+9_{10}$ и -9_{10} бити представљени у 8-битном рачунару.



У систему са знаком могу се са 8 битоа представити само бројеви од -128_{10} до $+127_{10}$, а са 16-битова -32768_{10} до $+32767_{10}$.

3.7.2 Комбиновање речи податка

Често је рачунару потребно да обрађује податке који су већи од податка одређеног величином речи. У таквим случајевима подаци се могу представити преко две или више речи које могу бити смештене у узастопне локације. На пример, микрорачунарски систем са микроконтролером MC68HC11, ради са 8-битним податком. 16-битни број као што је

3. Увод у рачунаре

1010101100101001_2 може се сместити у две узастопне меморијске локације на следећи начин:

Меморијска адреса	Садржај меморијске адресе	
0030_{16}	10101011_2	← 8 виших битова од 16-битног броја
0031_{16}	00101001_2	← 8 нижих битова од 16-битног броја

Видимо да је 8 битова веће тежине 16-битног броја смештено на меморијској адреси 0030_{16} и назива се бајт веће тежине. Слично, 8 битова мање тежине 16-битног броја (бајт мање тежине) смештено је на меморијској адреси 0031_{16} . Комбинација ова два бајта сачињава комплетан 16-битни број. Процесор микроконтролера MC68HC11 обрађиваће 16-битни број у два корака, бајт по бајт.

3.7.3 Хексадецимално представљање бројева

Због једноставнијег писања подаци се могу представити у хексадецималном бројном систему. На пример број $+116_{10}$ може се представити као један бајт 01110100_2 . Његове репрезентације - хексадецимална, децимална и октална су:

$$\begin{aligned}01110100_2 &= 74_{16} \\ 01110100_2 &= 164_8\end{aligned}$$

Важно је напоменути да је употреба хексадецималне репрезентације погоднија за корисника рачунара. У рачунарској меморији смештене су јединице и нуле, и то је оно што рачунар обрађује.

3.8 КОДИРАНИ ПОДАЦИ

Податак који рачунар обрађује не мора да буде бинарни број. Други начин представљања податка је BCD¹ код, где група од 4 бита представља једну децималну цифру. Тако, 8-битна реч може да представља 2 децималне цифре. Неки рачунари могу да извршавају аритметичке операције са BCD кодираним бројевима као део њиховог нормалног инструкцијског репертоара. Други, нарочито неки микрорачунари,

¹ Binary Coded Decimal

захтевају посебан напор програмера да би извршили BCD аритметику.

Реч податка не представља само нумеричке податке. Често се користи за представљање слова и других специјалних знакова који користе кодове као што су 7-битни ASCII¹ кôд. ASCII кôд користе сви произвођачи рачунара. Основни ASCII кôд користи 7 битова, а са 8. битом, који се користи за парност или проширење скупа знакова, добија се једнобајтни ASCII кôд. Следећи пример показује како порука, користећи ASCII кôд, може да буде смештена у узастопним меморијским локацијама. Садржај сваке од локација дат је у бинарном и хексадецималном коду. Користећи табелу за кôд може се одредити порука. Уочимо да је бит највеће тежине бит парности и да је први карактер смештен на локацији 012A₁₆.

Меморијска адреса	Садржај меморијске адресе (бинарни)	Садржај меморијске адресе (хексадецималан)
012A ₁₆	11001001	C9
012B ₁₆	10111101	BD
012C ₁₆	01010110	56
012D ₁₆	10101111	AF
012E ₁₆	11010010	D2
	↑ ASCII	

Декодована порука је Омов закон, $I = U/R$.

Једнобајтни ASCII кôд је посебно подешен за рачунаре са 8-битном величином речи. У ствари, остали рачунари (величине речи 16, 32) такође користе једнобајтни ASCII кôд. На пример, 16-битни рачунар може да спакује два бајта у једну меморијску реч тако да свака реч представља два карактера. Ово је илустровано на следећи начин:



¹ American Standard Code for Information Interchange

3.8.1 Интерпретација речи податка

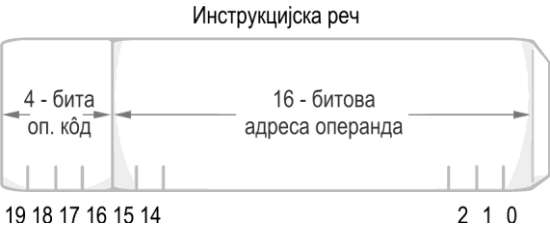
Претпоставимо да је речено да је одређена реч податка у микрорачунарској меморији 01010110_2 . Ова реч може се интерпретирати на више начина. Може да буде бинарна репрезентација броја 86_{10} може да буде BCD репрезентација 56_{10} , или може да буде ASCII код карактера В. Како ће податак бити интерпретиран? Зависи од програмера пошто она (или он) смештају податак у меморију заједно са инструкцијама које заједно чине програм. Програмер зна који тип речи податка се користи и мора се обезбедити да рачунар интерпретира податке тачно (преко програма који извршава, а који се састоји од инструкција).

3.9 ИНСТРУКЦИЈСКА РЕЧ

Програм се састоји од скупа (секвенце) бинарно кодираних инструкција које процесор мора да прочита из меморије, дешифрује (декодује) и изврши. За неке рачунаре инструкцијска реч, од које је сачињен програм, може да носи два типа информација: операцију која треба да се изврши и адресу операнда (податка) над којим се операција извршава. Рачунари који имају велику дужину речи могу све то да сместе у једну рачунарску реч, док микрорачунарима треба више од једне речи. Пре него што видимо какав је формат инструкција у 8-битном микрорачунару, посматраћемо формат инструкцијске речи за рачунаре са великом дужином речи.

Слика 3.3 показује пример инструкцијске речи хипотетичног 20-битног рачунара. 20 битова је подељено у два дела. Први део (битови 16-19) садрже код операције (операциони код). Четворобитни код указује на операције које процесор треба да изврши. Сваки од могућих операционих кодова представља поједине операције, као што су сабирање или одузимање. Други део (битови 0-15) је адреса операнда, односно адреса у меморији одакле ће подаци (операнд) бити узети или где ће бити упућени, зависно од операције која се извршава.

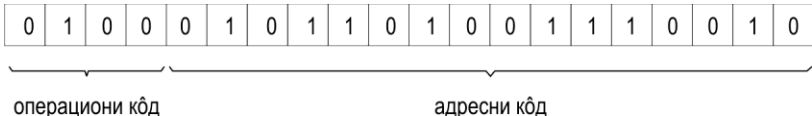
Са 4 бита, који се користе као операциони код, постоји могућност од $2^4=16$ различитих операционих кодова, где сваки од њих означава другу операцију. Ово значи да је рачунар, користећи ову инструкцијску реч, ограничен на 16 различитих инструкција које може да изврши. Рачунар који има више инструкција захтева више битова за операциони код. У



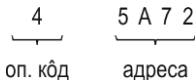
Слика 3.3 Типични формат інструкційсє речи

сваком случају, свака инструкција коју рачунар може да извршава, има специфичан операциони кођ који рачунар (управљачка јединица) мора да интерпретира (декодује).

Инструкцијска реч дата на слици 3.3 је са 16 битова резервисаних за адресни код операнда. Са 16 битова постоји $2^{16}=65536$ различитих могућих адреса. Тако, инструкцијска реч може да специфицира 16 различитих инструкција и 65536 адреса операнда. На пример 20-битна инструкцијска реч може да буде:



Операциони код 0100₂ представља једну од 16 могућих операција; претпоставимо да је то код за сабирање (ADD). Адресни код (адреса операнда) је 0101101001110010₂, или 5A72₁₆ (хексадецимално). У ствари, комплетна реч инструкције може се представити хексадецимално као:



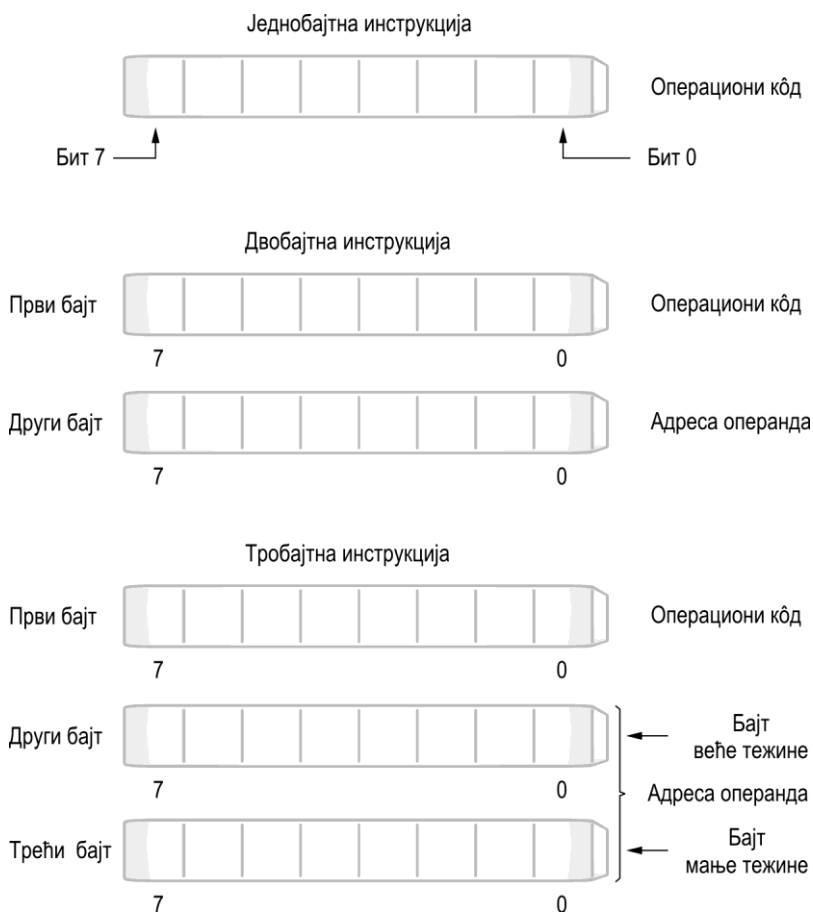
Ова инструкцијска реч указује рачунару да уради следеће: узми реч податка смештену на адресној локацији 5A72₁₆, пошаљи у аритметичко логичку јединицу, сабери са бројем у акумулатору¹. Збир ће бити смештен у акумулатору (претходни садржај акумулатора је изгубљен).

¹ Користи се и термин акумулаторски регистар.

3.9.1 Формат инструкције у микрорачунарима

Осмобитни процесор MC68HC11 не може да смести операциони кôд и адресу операнда у једну 8-битну реч; зато код ње постоје три инструкцијска формата: једнобајтни, двобајтни и тробајтни. Ово је илустровано на слици 3.4

Једнобајтна инструкција садржи само 8-битни кôд операције; није специфициран операнд. Пошто не постоји адреса операнда ове



Слика 3.4 Формати инструкција у 8-битном процесору

инструкције се користе код оних операција које не захтевају приступ делу меморије са подацима. На пример инструкција - обриши садржај акумулатора CLRA¹, која налаже процесору да постави све битове у акумулатору на нулу. Пошто је акумулатор регистар који је део процесора не постоји потреба за адресом операнда. Процесор MC68HC11 користи операциони код 01001111₂ (или 4F₁₆) да би представио операцију CLRA. Различити процесори користе различите операционе кодове да би репрезентовали CLRA операцију. Генерално, различити процесори имају различит скуп операциони кодова због разлике у пројектовању управљачких кола.

Први бајт у двобајтној инструкцији приказан на слици 3.4 је операциони код, а други бајт је 8-битни код адресе, који одређује адресу операнда. Ова два бајта увек су смештена у меморији назначеним редоследом. То је илустровано на следећи начин:

Меморијска адреса (хексадецимална)	Садржај меморијске адресе		Опис
	Бинарно	Хекс.	
D200	10010110	96	← Операциони код за LDAA
D201	0101110	5E	← Адреса на којој су подаци који треба да се читају у акумулатор
D202	10010111	97	← Операциони код за STAA
D203	10001010	8A	← Адреса где ће садржај акумулатора бити смештен

У левој колони је списак адреса локација у меморији у којима је сваки од бајтова смештен. Друга и трећа колона дају стварне речи смештене у свакој меморијској локацији у бинарном или хексадецималном облику. У колони са десне стране налази се опис значења сваке меморијске речи.

Бајт смештен на адреси D200₁₆ је операциони код, део прве инструкције; тј. операциони код процесора MC68HC11 за инструкцију LDAA². Бајт смештен на D201₁₆ део је прве инструкције који се односи на адресу операнда. Комплетна двобајтна инструкција налаже процесору да уради следеће:

¹ Clear Accumulator

² Load Accumulator A

3. Увод у рачунаре

Узми податке који се налазе на меморијској адреси $005E_{16}$ и стави их у акумулатор. Садржај адресе $005E_{16}$ остаје непромењен.

Бајт смештен на адреси $D202_{16}$ је операциони кођ за следећу инструкцију; тј. операциони кођ за инструкцију STAA¹. Бајт смештен на $D203_{16}$ је део адресе операнда друге инструкције. Комплетна двобајтна инструкција налаже процесору MC68HC11 да уради следеће:

Узми садржај акумулатора и смести га на меморијску адресу $008A16$. Садржај акумулатора остаје непромењен.

Уколико посматрамо тробајтну инструкцију на слици 3.4, видимо да је први бајт операциони кођ. Други и трећи бајт формирају 16-битну адресу операнда. Пример тробајтне инструкције је следећи:

Меморијска адреса (хексадецимална)	Садржај меморијске адресе		Опис
	Бинарно	Хекс.	
0020	10111011	BB	← Операциони кођ за ADD
0021	00110101	35	← Виши бајт адресе операнда
0022	11110110	F6	← Нижи бајт адресе операнда

Инструкција налаже процесору да уради следеће:

Узми податке смештене у меморији на адреси $35F6_{16}$, додај их садржају акумулатора, и резултат смести у акумулатор.

Јасно је да је тробајтна инструкција исто што и двобајтна осим што се користи дужи кођ за адресу. Неки процесори, нпр. процесор MC68HC11, дозвољавају да је адреса операнда било 8-битна (један бајт) или 16-битна (два бајта). Видећемо да процесор MC68HC11 увек користи 16-битну адресу, чак и код двобајтне инструкције: аутоматски се додаје 00_{16} као виши бајт на једнобајтну адресу. На пример једнобајтна адреса $7F_{16}$ постаје адреса $007F_{16}$.

Као што смо у претходној анализи видели једнобајтне, двобајтне и тробајтне инструкције увек започињу једнобајтним операционим кодом.

¹ Store the Accumulator A

Такав формат користи већина 8-битних процесора. Међутим процесор MC68HC11 користи изванредан број инструкција које одступају од тог правила. На пример све инструкције које користе интерни Y регистар (један од интерних регистара процесора MC68HC11) имају специфичан пред бајт¹. Он указује да инструкција користи интерни Y регистар процесора MC68HC11. То значи да постоје специјални случајеви када процесор MC68HC11 за операциони код корист два бајта: предбајт и стварни операциони код за инструкцију коју треба да изврши.

3.9.2 Извршавање програма

Програм се састоји од инструкција које тачно дефинишу рачунару шта да ради, корак по корак. За 8-битни рачунар програм ће се састојати од једнобајтне, двобајтне и тробајтне инструкције смештене у меморији. Програм може да буде смештен у RAM меморију (уколико постоји могућност да ће се програм касније мењати) или у ROM меморију (уколико програм остаје за стално у меморији).

Процесор ће узети (учитати), декодовати (дешифровати) и извршити инструкцију из меморије једну по једну, почевши од унапред дефинисане почетне адресе. Пошто 8-битни рачунар користи више од једне меморијске речи по инструкцији, процесор мора да обави више операција читавања, тј. узимања, пре него што изврши саму инструкцију. Ово је илустровано на слици 3.5

Процесор увек почиње узимањем бајта операционог кода, а затим декодује операциони код да би одредио шта да уради следеће. Операциони код је тај који указује процесору да ли треба да узме адресу операнда пре извршавања инструкције, и да ли су потребне једна или две операције узимања да би се добила адреса операнда. Процесор непрекидно понавља серију корака док не наиђе на операциони код за инструкцију заустављања WAI², што означава заустављање свих операција.

У зависности од типа инструкције мењаће се и време потребно да процесор изврши инструкцију. То није само последица разлике у адреси

¹ Prebyte

² Wait for Interrupt – заустављање свих операција до појаве прекида.



Слика 3.5 Серија корака у извршавању програма процесора MC68HC11

операнда, већ и због различитих секвенци догађаја који се одвијају за време фазе извршавања.

Пример 3.1 *Процесор ради са 8-битном речју и 16-битном адресом.*

- Колики је максималан број различитих инструкција које процесор може да извршава?
- Колики је максималан капацитет интерне меморије са којом може процесор да комуницира?

Решење:

- Са 8-битним операциони кодом постоји $2^8=256$ могућих операционих кодова, сваки представља различиту инструкцију. На први поглед то значи да процесор може да извршава 256 различитих инструкција. Пројектанти процесора MC68HC11 захтевали су већи број инструкција од 256. Због тога су резервисали неколико

специјалних кодова (18_{16} , $1A_{16}$ и CD_{16}). Било који од ових бајтова указује да инструкција која следи користи два бајта као операциони код. Тако је интеграцијом предбајта и операционог кода добијено 308 различитих инструкција.

- б) 16-битни адресни код одређује $2^{16} = 65\,536$ различитих меморијских локација. Максималан капацитет ће тада бити $65\,536 \times 8$ или $64K \times 8$.

Пример 3.2 *Колико операција узимања треба да обави процесор MC68HC11 при извршавању инструкције додавања (ADDA) податка из меморијске локације \$A2FF са акумулатором?*

Решење:

Четири операције узимања се захтевају: једна за узимање операционог кода, још две друге за узимање адресе операнда (\$A2FF), и четврта за узимање података који се додају акумулатору за време фазе извршавања.

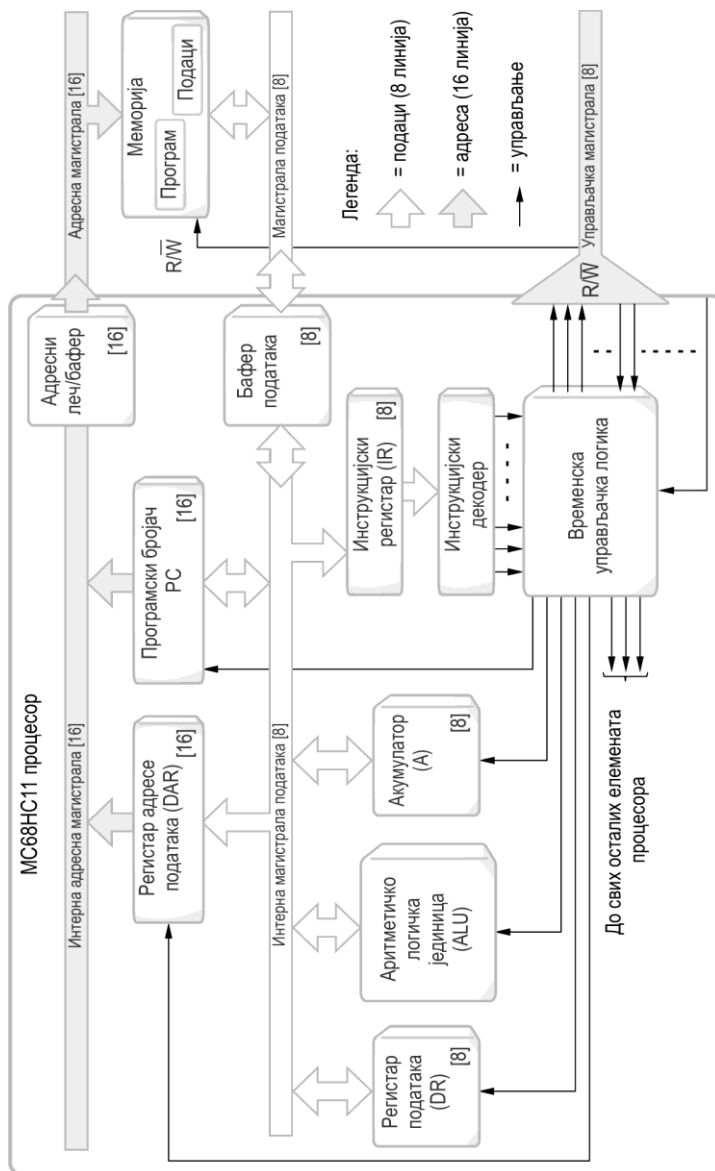
3.10 ПРОЦЕСОР MC68HC11 - ПОЈЕДНОСТАВЉЕНА ВЕРЗИЈА

Пошто смо објаснили основне кораке које процесор MC68HC11 чини за време извршавања инструкција, можемо се мало детаљније осврнути на сам процесор. Он је веома комплексан модул који садржи највећи део логичких кола, већи број регистара са различитим функцијама и велики број сигналних линија које повезују њих са другим елементима у микрорачунару.

Поједностављен дијаграм процесора MC68HC11 приказан је на слици 3.6. Прво ће бити описане функције сваког од елемента на дијаграму а затим ће бити описано у каквој су они међусобној вези за време извршавања програма. Процесор је део уоквирен дебелом црном линијом, а приказана је и њена веза са меморијом. Због једноставности нису представљени улазно/излазни блокови; претпоставља се да су програм и подаци већ у меморији.

У процесору постоје три спољашње-екстерне магистрале¹ (ван процесора) и две унутрашње-интерне (унутар процесора). Спољашња адресна магистрала или само адресна магистрала и магистрала података

¹ Bus



Слика 3.6 Поједностављен блок дијаграм процесора MC68HC11

је проширење истих магистрала у оквиру процесора MC68HC11, тако да стварно постоје само три различите магистрале: адресна магистрала, магистрала података и управљачка магистрала.

Адресна магистрала има 16 линија. Она носи 16-битни адресни код из процесора MC68HC11 ка меморијској јединици да би се одабрала меморијска локација којој процесор приступа због операција читавања (READ) и уписивања (WRITE). Адресна магистрала је једносмерна пошто се информације преносе (теку) само у једном правцу.

Магистрала података је 8-линијска двосмерна магистрала преко које 8-битне речи могу да буду послате из процесора у меморију (операција уписивања), или из меморије ка процесору (операција читавања). Иако се назива „магистрала података“ информације које се преносе овом магистралом нису увек подаци који се обрађују. Често су то инструкцијски кодови (операциони код, адреса операнда) које процесор MC68HC11 узима из меморије

Под именом „управљачка магистрала“ подразумева се група свих управљачких и временских сигнала потребних за синхронизовање рада процесора са осталим деловима рачунара. Неки од управљачких сигнала су излазни (из процесора ка улазно/излазним јединицама), а други су улазни (у процесор од улазно/излазних јединица). Највећи број 8-битних процесора, као што је процесор MC68HC11, имају 8-битну магистралу података и 16-битну адресну магистралу.

3.11 РЕГИСТРИ ПРОЦЕСОРА MC68HC11

Подсетимо се да процесор садржи аритметичко/логичку и управљачку јединицу. Она такође садржи неколико регистара који се користе за смештање различитих типова података који су потребни процесору MC68HC11 за обављање њених функције. Другим речима ови регистри служе као наменске меморијске локације, а налазе се унутар интегрисаног кола MC68HC11.

Сваки од регистара на слици 3.6 означен је својим правим именом и скраћеницом, а број у загради специфицира величину регистра у битовима. У овом уџбенику ће се користити ове скраћенице. Види се да је највећи број регистара везан за интерну (унутрашњу) магистралу

података, тако да се информације могу пренете између ових регистара и меморијске јединице користећи магистралу података.

3.11.1 Програмски бројач

Програмски бројач је регистар који ради као 16-битни бројач и који управља редоследом узимања инструкције из меморије. У сваком тренутку садржај програмског бројача (PC¹) показује адресу у меморији са које ће се преузети следећи бајт операционог кода.

Кад процесор треба да добије (узме) бајт инструкцијског кода управљачки сигнали из блока за временске и управљачке сигнале постављају садржај програмског бројача на интерну магистралу и памте га у адресном леч² баферу, који управља спољашњом адресном магистралом. На тај начин се меморији представља 16-битни адресни код једнак броју у програмском бројачу. Да би се припремио за следећу операцију узимања програмски бројач (PC) се увећава.

3.11.2 Регистар адресе податка

Програмски бројач обезбеђује адресу у меморији инструкцијског кода за време операције узимања. За време фазе извршења инструкције, када процесор треба да приступи меморији, регистар адресе податка (DAR³) обезбеђује адресу операнда у меморији. На пример, за време извршења инструкције „смести акумулатор” – STA⁴ регистар адресе податка садржи адресу операнда, тј. локацију у меморији где садржај акумулатора треба да се смести. Сигнали из временског и управљачког блока сместиће садржај регистра адресе податка на интерну магистралу и држати га у леч-баферу адресе. Програмски бројач увек садржи адресу инструкције која се налази у делу меморије за програме (програмске меморије). Регистар адресе податка увек садржи адресу податка (операнда) који се налази у делу меморије за податке.

3.11.3 Инструкцијски регистар

Да би добио (узео) код операције из програмске меморије процесор

¹ Program Counter

² Latch - држање

³ Data Address Register

⁴ Store the Accumulator

MC68HC11 помоћу својих управљачких кола генерише сигнале који раде следеће:

1. постављају садржај програмског бројача на адресну магистралу;
2. постављају линију читавања/уписивања на вредност 1 ($R/\overline{W}=1$) што указује на операцију читавања; и
3. узимају читану (резултујућу) меморијску реч са магистрале података и смештају је у инструкцијски регистар (IR^1). Тако на крају овог процеса, инструкцијски регистар садржи операциони код инструкције.

Инструкцијски регистар чува операциони код док га кола инструкцијског декодера декодују и јавља временској и управљачкој логици коју секвенцу управљачких сигнала треба да генерише да би се извршила инструкција .

3.11.4 Акумулатор и регистар података

За време фазе извршења инструкције ова два регистра садрже операнде над којима аритметичко/логичка јединица извршава операцију. Регистри су напуњени операндима из меморије. Резултат операције се преноси у акумулатор (A^2). Треба уочити да се преко магистрале података подаци из меморије преносе и у акумулатор и у регистар података (DR^3), а да се само подаци из акумулатора могу послати натраг у меморију.

3.12 ИЗВРШАВАЊЕ ПРОГРАМА

Сада смо спремни да сагледамо како различити регистри и кола процесора MC68HC11 раде заједно у стварном програму. У овом тренутку нас само интересује како програм ради. Зато ћемо искористити веома једноставан програм који сабира два 8-битна бинарна броја. Пре него што погледамо пример програма морамо да знамо које се инструкције могу користити у овом програму. Сваки процесор има листу инструкција за коју је пројектован да ради и која се означава као његов инструкцијски скуп⁴. Инструкцијски скуп процесора MC68HC11 чини 306 различитих

¹ *Instruction Register*

² *Accumulator*

³ *Data Register*

⁴ *Instruction set*

3. Увод у рачунаре

инструкција, али ћемо за сада користити само један мали део тог скупа који је приказан у табели 3.1.

Операција	Мнемоник	Операциони кôд	Опис инструкције
Пуњење акумулатора (<i>Load Accumulator A</i>)	LDA	B6	Акумулатор се пуни подацима из меморијске локације са адресе која је дата са два бајта која се налазе одмах после операционог кода
Смештање акумулатора (<i>Store Accumulator A</i>)	STAA	B7	Садржај акумулатора се смешта у адресу која је дата у двобајтној дреси операнда
Додавање акумулатора (<i>Add to Accumulator A</i>)	ADDA	BB	Подаци са меморијске локације која је одређена двобајтном адресом операнда додаје се акумулатору
Одузимање (<i>Subtract from Accumulator A</i>)	SUBA	B0	Исто као и сабирање само се подаци одузимају од садржаја акумулатора
Заустављање	WAI	3E	Заустави све операције
Скок (<i>Jump</i>)	JMP	7E	Пуни програмски бројач двобајтном адресом операнда и узима операциони кôд са те адресе (као GO TO вишим програмским језицима)
Гранање ако је резултат операције нула (<i>Branch if Equal to Zero</i>)	BEQ	27	Уколико последња операција даје као резултат нулу, додај бајт који је иза операционог кода, и узми следећу инструкцију са те нове адресе. У супротном узети следећу инструкцију.

Табела 3.1 Део инструкцијског скупа процесор MC68HC11

Прва колона у табели указује на операцију коју инструкција извршава. У другој колони су симболичне ознаке (мнемоници) инструкција. Мнемоник је обично скраћеница од два или три слова, која се користи да би представила инструкцију. У трећој колони дати су хексадецимални операциони кодови инструкција. Последња колона даје опис операције коју изводе инструкције.

Посматрајући ову табелу можемо уочити да осим WAI инструкције, која не захтева адресу операнда, све остале инструкције користе двобайтну адресу операнда. Инструкција BEQ је специфична и њу ћемо касније детаљније описати.

3.12.1 Пример за извођење програма

У даљем тексту дат је кратак програм који користи неке од инструкција из претходне табеле. У првој колони дате су адресе меморијских локација у којима је програм смештен, а друга колона даје њихов садржај. Обе (адресе и садржај) су представљене у хексадецималној нотацији. Треба имати на уму да се сви подаци у рачунару памте у бинарном облику. У нашој анализи подразумевамо да је програм већ у меморији, и за сада не водимо рачуна како се он тамо нашао.

<i>Меморијска адреса</i>	<i>Меморијска реч</i>	<i>Мнемоник</i>	<i>Опис</i>
D020	B6	LDAA	Напуни акумулатор A
D021	10		Адреса
D022	01		операнда X
D023	BB	ADDA	Додај Y садржају A
D024	10		Адреса
D025	02		операнда Y
D026	B7	STAA	Смести A
D027	10		Адреса где
D028	03		A треба сместити
D029	3E	WAI	Заустављање извршавања програма

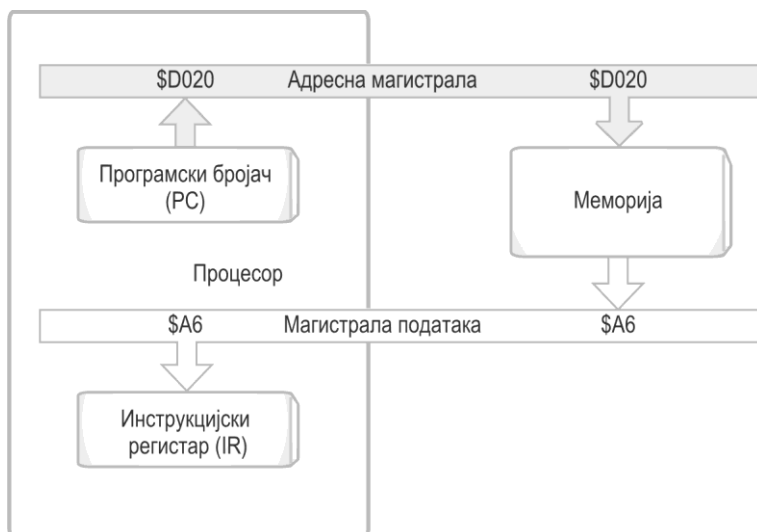
3. Увод у рачунаре

Намена овог програма је да сабере два 8-битна броја (X и Y) који су смештени у делу меморије за податке на адресама \$1001 и \$1002. Збир се смешта на адресу \$1003.

Извршавање програма

Да би започео извођење програма програмски бројач мора да буде постављен на адресу прве инструкције. У примеру који анализирамо почетна адреса је \$D020. Претпоставићемо да је управљачко/временска логика (TCL¹) већ поставила програмски бројач на \$D020 и нећемо за сада водити рачуна о томе како је то изведено.

1. *Узимање операционог кода за LDAA.* Управљачко/временска логика (TCL) поставља садржај програмског бројача $[PC]=D020_{16}$ на адресну магистралу тако да меморија добија тај адресни код. Управљачко/временска логика поставља линију за читавање/уписивање на вредност 1 ($R/\overline{W}=1$), тј. операцију читавања. Меморија одговара стављајући реч \$A6 са адресе \$D020 на магистралу података. Управљачко/временска логика омогућава



Слика 3.7 Процесор читава (узима) операциони код са адресе \$D020

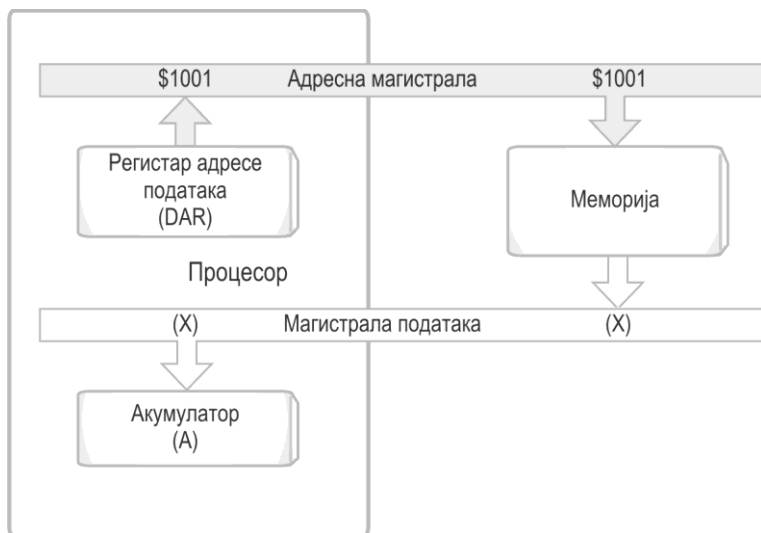
¹ Time Control Logic

да се та реч уписује у инструкцијски регистар (IR). Ова операција може се илустровати сликом 3.7 и симболично представити: $[M] \rightarrow IR$.

2. *Инкрементирање (увећавање за 1) програмског бројача.* Програмски бројач се увећава за један тако да је његов садржај: $[PC] = D021_{16}$. Симболично се представља: $[PC] + 1 \rightarrow PC$. Овим кораком програмски бројач се припрема за нову операцију читавања (узимања) следећег бајта из програмског дела меморије.
3. *Декодовање операционог кода.* Инструкцијски декодер декодује операциони код (\$A6) и сигнализира управљачко/временској логици да почне да генерише секвенце управљачких сигнала потребне да би се наставило са читавањем осталих делова LDAA инструкције и њеним извршавањем.
4. *Очитавање адресе операнда (бајта веће тежине).* Управљачко/временска логика (TCL) поставља садржај програмског бројача $[PC] = D021_{16}$ на адресну магистралу и линију $R/\overline{W} = 1$. Меморија одговара постављајући реч \$10 на магистралу података. Овим бајтом пуни се виши бајт регистра адресе податка (DAR). Треба имати на уму да је регистар адресе податка (DAR) 16-битни регистар (два бајта). Овај корак се може представити на следећи начин: $[M] \rightarrow DAR$.
5. *Увећава се садржај програмског бројача на \$D022 ($[PC] = D022_{16}$).*
6. *Очитавање адресе операнда (бајта мање тежине).* Управљачко/временска логика поставља \$D022 на адресну магистралу, и поставља линију $R/\overline{W} = 1$. Меморија одговара постављајући реч \$01 на магистралу података, и управљачко временска логика је смешта као бајт мање тежине у регистар адресе податка (DAR). Сада регистар адресе податка (DAR) садржи адресу првог операнда: \$1001. Овај корак може се представити на следећи начин: $[M] \rightarrow DAR$.
7. *Увећава се садржај програмског бројача на \$D023 ($[PC] = D023_{16}$).*
8. *Извршава се инструкција LDAA.* Адреса операнда из регистра адресе податка постављена је на адресну магистралу а линија $R/\overline{W} = 1$. Меморија одговара постављајући податак са адресе

\$1001 на магистралу података, а то је операнд X. Управљачко/временска логика (TCL) обезбеђује да се овим податком напуни акумулатор (A). Операција се може представити на следећи начин: $[M] \rightarrow A$ и илустровати сликом 3.8

9. *Узимање операционог кода инструкције ADDA.* Управљачко/временска логика поставља садржај програмског бројача $[PC]=D023_{16}$ на адресну магистралу и поставља линију $R/\overline{W}=1$. Меморија поставља реч \$BB на магистралу података, и управљачко временска логика пуни инструкцијски регистар овим бајтом. Симболично се може представити: $[M] \rightarrow IR$.
10. *Увећава се садржај програмског бројача на вредност \$D024.*
11. *Декодовање операционог кода.* Инструкцијски декодер анализира управљачко/временској логици да започне генерисање секвенци управљачких сигнала потребних за наставак извођења ADDA инструкције.
12. *Узимање адресе операнда (бајта веће тежине).* Адреса \$D024 из програмског бројача поставља се на адресну магистралу, а линија



Слика 3.8 Процесор уписује у (пуни) акумулатор операндом X

- $R/\overline{W}=1$. Меморија поставља реч \$10 на магистралу података која се уписује у регистар адресе података (DAR). Символично се може представити: $[M] \rightarrow DAR_x$.
13. *Увећава* се садржај програмског бројача на вредност \$D025.
 14. *Узимање адресе операнда* (бајт мање тежине). Адреса \$D025 из програмског бројача поставља се на адресну магистралу и $R/\overline{W}=1$. Меморија затим поставља реч податка \$02 на магистралу података да би се њом напунио регистар адресе података. Символички $[M] \rightarrow DARL$. На овом месту регистар адресе података садржи адресу другог операнда: \$1002.
 15. *Увећава* се садржај програмског бројача - $[PC]=D026_{16}$.
 16. *Извршавање ADD*. Адреса операнда \$1002 из регистра адресе податка (DAR) налази се на адресној магистрали, и $R/\overline{C}$ се поставља на висок ниво (1). Меморија поставља податак са локације \$1002 на магистралу података; ово је операнд Y. Временско затим пуни овим подацима регистар података (DR) и даје команду аритметичко/логичкој јединици да их дода акумулатору (који садржи операнд X). Резултат се ставља у акумулатор. Символично $[M]+[A] \rightarrow A$.
 17. *Узимање операционог кода за STA*. Садржај програмског бројача $[PC]=D026_{16}$ смешта се на адресну магистралу и $R/\overline{W}$ се поставља на висок ниво. Меморија затим поставља реч \$B7 на магистралу података, и затим се смешта у инструкцијски регистар (IR).
 18. *Увећавање садржаја програмског бројача* - $[PC]=D027_{16}$
 19. *Декодовање операционог кода*.
 20. *Узима адресу операнда* (бајта веће тежине) из меморијске локације дате са $[PC]=D027_{16}$ и ставља га у бајт веће тежине регистра адресе податка.
 21. *Увећава* се садржај програмског бројача - $[PC]=D028_{16}$
 22. *Узима адресу операнда* (бајта мање тежине) са адресе \$D028 и ставља га у бајт мање тежине регистра адресе податка. Регистра адресе податка (DAR) сада садржи адресу операнда \$1003.
 23. *Увећава* се садржај програмског бројача - $[PC]=D029_{16}$.
 24. *Извршава STA*. Адреса операнда 1003 из регистра адресе податка (DAR) ставља се на адресну магистралу. Управљачко/временска

логика (TCL) затим поставља садржај акумулатора на магистралу података и поставља $R/\overline{W}$ на 0 (тј. брише је), тако да се садржај акумулатора А уписује у меморијску локацију \$1003. Симболично $[A] \rightarrow M$.

25. *Узимање операционог кода за инструкцију WAI са адресе дате са $[PC]=D029_{16}$. Њиме се пуни инструкцијски регистар (IR).*
26. *Увећава се садржај програмског бројача - $[PC]=D02A_{16}$.*
27. *Декодовање операционог кода.*
28. *Извршавање инструкције WAI. Процесор зауставља све даље операције.*

3.13 ИНСТРУКЦИЈЕ ГРАНАЊА И СКОКА

За сада нећемо разматрати комплетан скуп инструкција процесора MC68HC11. У ствари постоје два типа инструкција које ћемо анализирати због начина на који они мењају редослед инструкција које извршава процесор. Пре него што кренемо даље позабавићемо се кратком анализом инструкције JMP и BEQ описаних у табели 3.1

Меморијска адреса	Меморијска реч	Мнемоник	Опис
D329	4F	CLRA	Обриши [A] на 0
D32A	7E	JMP	Скочи на адресу \$D35F по следећу инструкцију
D32B	D3		
D32C	5F		Адреса следеће инструкције
D32D	—		
D32E	—		
D35F	B7		
D360	01	STA	Смешта [A] у меморијску локацију \$0127
D361	27		

3.13.1 Инструкција скока JMP

Пример који је анализиран у претходном поглављу може се описати као праволинијски. Тако се зове зато што процесор узима инструкције из меморије са адресе \$D020 и наставља редом док не наиђе на

инструкцију WAI на локацији \$D029. Ово се реализује увећавањем садржаја програмског бројача после сваке операције узимања.

Иако су праволинијски програми једноставни за писање и разумевање, они могу да реше ограничени број задатака. Многи програми користе технику која се назива „петље”. Петље омогућавају процесору да прекине нормални праволинијски редослед извођења програма и „скочи”, или се грана до других делова програмске меморије да би наставио извођење програма.

Инструкција JMP (скок) налаже процесору да скочи до одређене меморијске адресе (не оне која је по редоследу) за своју следећу инструкцију. У вишим програмским језицима овој инструкцији одговарала би GO TO инструкција.

У програму који следи приказан је пример коришћења JMP инструкције:

Ово је део програма који је смештен у меморији. Претпоставимо да процесор извршава овај програм. Претпоставимо даље да је управо завршио извршавање CLRA инструкције на \$D329. Следећи корак је да узме код инструкције JMP са адресе \$D32A. Када процесор одреди да је то инструкција JMP зна да се двобајтна адреса налази иза операционог кода. Узима адресу (\$D35F) у два корака. Затим замењује текући садржај програмског бројача (\$D32D) са \$D35F. Са програмским бројачем који садржи адресу \$D35F процесор ће следећи операциони код узети са ове адресе.

Другим речима уместо да узме инструкцију са адресе \$D32D, процесор „скаче” преко адреса \$D32D и \$D32E и наставља са извршавањем програма са адресе \$D35F („иде на адресу \$D35F ”). Као што се види у овом примеру JMP инструкција је тробајтна инструкција. Бајт операционог кода праћен је са двобајтном адресом која указује процесору где да скочи.

3.13.2 Инструкција гранања BEQ

Поред безусловног скока постоји још један начин да се промени редослед извршавања инструкција. То су инструкције за условно гранање. Процесор MC68HC11 има више инструкција за условно гранање.

3. Увод у рачунаре

Ми ћемо да се задржимо на BEQ¹ (гранање уколико је резултат 0) као представнику ове групе инструкција. Први бајт је наравно операциони кôд. Други бајт је бинарни број са знаком, такозвано релативно адресно одступање (офсет) или једноставно померај. Одступање није адреса, него број који треба додати програмском бројачу да би се формирала нова адреса. Ово је илустровано програмском секвенцом:

Меморијска адреса	Меморијска реч	Мнемоник	Опис
D200	B2	SUBA	Одузми садржај адресе \$029B од
D201	02		садржаја акумулатора. Остави резултат у
D202	9B		акумулатору.
D203	27	BEQ	Уколико је резултат претходне операције 0
D204	14		настави са извршавањем прг. на адреси \$D219.
D205	3E	WAI	Уколико није, иди на адресу \$D205.
D219	B7	STAA	Смести садржај акумулатора
D21A	01		на адресу \$0127.
D21B	27		

Претпоставимо да је процесор управо завршио извршавање SUB инструкције на \$D200-\$D202 и да је у акумулатору резултат. Процесор затим узима операциони кôд на \$D203. Када детектује да је то BEQ инструкција, он узима бајт са одступањем са локације \$D204, и доноси одлуку о наставку извршавања програма на основу резултата последње операције. Пошто је резултат инструкције SUBA тачно нула (тј. $00_{16} = 00000000_2$), процесор ће додати бајт одступања на текући садржај програмског бројача. Садржај програмског бројача је $\$D205_{16}$ и после узимања бајта одступања биће:

$$\begin{array}{rcl} [\text{PC}] & = & D205_{16} \\ + \text{одступање} & = & 14_{16} \\ \hline \text{нови } [\text{PC}] & = & D219_{16} \end{array}$$

Нови садржај програмског бројача је \$D219, па ће процесор своју следећу инструкцију узети са те адресе. Уколико резултат SUBA операције

¹ Branch if Equal zero

није једнак нули, процесор неће додати одступање програмском бројачу већ ће наставити са извршавањем инструкције по реду (тј. без скока). У вишим програмским језицима инструкција гранања је врло слична IF-THEN инструкцији.

3.14 ХАРДВЕР, СОФТВЕР И ФИРМВЕР

Рачунарски систем састоји се од хардвера, софтвера и фирмвера. Хардвер означава све електронске, механичке и уређаје за смештање података од којих се физички састоји рачунар. Количина хардвера мења се у широким границама од малих микрорачунара до суперрачунара, али је једна ствар заједничка за све рачунаре - хардвер је бескористан без програма.

Софтвер се односи на програме који управљају активностима рачунарских система од укључења, па све до искључења. Сав софтвер спада у једну од две категорије: апликативни софтвер или системски софтвер. Апликативни софтвер се користи за специфичне задатке као што су статистичка анализа, управљање индустријским процесима, видео игре... Системски софтвер је општији од апликативног, он подржава сав апликативни софтвер управљајући основним функцијама рачунара. На пример, системски софтвер управља процесом иницијализације кроз који рачунар пролази одмах по укључењу. Још један пример је системски софтвер који нам омогућава да уђемо у програме на језику какав је, на пример C, или C++, Java, C#...

Сав софтвер мора да се налази у рачунарској интерној меморији у моменту када треба да се изврши. Уобичајено, софтвер се пребацује из спољне меморије у интерну меморију (као што је нпр. трака, компакт диск, флеш меморија...). Када је програм у ROM меморији он се често назива и фирмвер, пошто је увек на располагању¹.

3.15 ПРОГРАМСКИ ЈЕЗИЦИ - МАШИНСКИ ЈЕЗИК

Програмирање подразумева писање програма у неком коду или на језику који рачунар може да интерпретира или извршава. Данас постоје многи програмски језици: они који користе нуле и јединице и они који користе

¹ *Firmware* – ожичени софтвер

3. Увод у рачунаре

изразе на енглеском језику. Сваки од језика има своје предности и мане у односу на друге језике. Ма који језик програмер користио, програм на крају мора бити преведен на једини језик који рачунар разуме, а то је језик нула и јединица - машински језик. Рачунар изводи специфичне операције на основу специфичних бинарних низова (операционих кодова), који формирају скуп наредби (инструкцијски сет). Ти бинарни низови представљају стварни језик који рачунари (машине) разумеју и са којим раде.

Произвођачи дефинишу машински језик (скуп инструкција) који њихови рачунари разумеју. У општем случају рачунари различитих произвођача имаће и различит машински језик. На пример, процесор MC68HC11 за инструкцију сабирања користи кôд 10111011, док MOS Technology 6502 користи кôд 01101101. То наравно значи да програм, написан на машинском језику једног рачунара, неће радити на другом. Од овог правила постоји тек неколико изузетака, кад произвођач намерно копира скуп операционих кодова другог произвођача.

Задатак¹ сваког програма је да генерише машински кôд, за одређени низ инструкција, у меморију рачунара. Због тога, машински програм учитан у меморију рачунара назива се објектни програм. Када програмер напише програм на папиру у машинском језику, он ће се састојати од низа нула и јединица. На пример кратак извршни кôд за сабирање два броја за процесор MC68HC11 је:

1	0	1	1	0	1	1	0
0	1	0	0	1	0	0	1
1	1	1	1	1	1	1	1
1	0	1	1	1	0	1	1
0	1	0	0	1	0	0	1
1	1	1	1	1	1	1	0
1	0	1	1	0	1	1	1
0	1	0	1	0	0	1	1
0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1

Јасно, функција овог програма није баш очигледна, сем за онога који га је написао, или можда за некога ко напамет зна све кодове инструкција

¹ *Object*

процесор MC68HC11. Чак и програмеру који је написао програм неће бити једноставно да, после извесног времена, одгонетне програм.

Писање програма директно у бинарном машинском језику има много мана. Прво, врло је тешко, сем за најједноставније програме, написати програм без иједне грешке, јер после извесног времена сви низови нула и јединица почињу да личе једни на друге. Грешке је тешко пронаћи у мору нула и јединица. Такође, процес уноса програма је заморан и лако може да доведе до грешке - сваки бајт се уноси преко осам прекидача на пулту рачунара, што за програм од нпр. само 100 бајтова износи 800 различитих положаја прекидача.

3.15.1 Хексадецимални машински језик

Рад са машинским језиком може се олакшати коришћењем окталног или хексадецималног кода (хексадецимални се стандардно користи). Програмер пише програм користећи хексадецимални код да би са њим представио операциони код, адресе операнда и податке. Горњи програм у хексадецималном коду изгледао би овако:

```
B6  
49  
FF  
BB  
49  
FE  
B7  
53  
00  
3F
```

Писање програма у хексадецималном коду је очигледно једноставније и мање подложно грешкама, јер сваки унос захтева само два знака уместо осам. У хексадецималном коду је и грешку лакше наћи.

Хексадецимални програм уноси се у рачунар преко тастатуре. Наравно, хексадецимални програм мора бити претворен у бинарни објектни програм пре него што се унесе у меморију. За ту сврху постоји програм у ROM меморији који се извршава док се уносе подаци са тастатуре (програм за надгледање тастатуре). Програм за надгледање детектује који тастери су притиснути, конвертује их у бинарни код и уноси у меморију.

3. Увод у рачунаре

Програмирање у хексадецималном коду, иако погодније од бинарног, далеко је од погодног. Такође, хексадецимални програм је још увек тежак за читање и разумевање.

3.16 АСЕМБЛЕРСКИ ЈЕЗИК

Програмирање на машинском језику (бинарном или хексадецималном) је најнижи ниво програмирања. Тај назив се користи зато што програмер комуницира са рачунаром на основном (најнижем) нивоу. Неке од тешкоћа комуницирања на том нивоу превазиђене су на следећем вишем нивоу језика - асемблерском језику.

Уместо коришћења бинарног или хексадецималног кода за писање инструкција, у асемблеру се користе симболичне ознаке или мнемоници. Мнемоници се лако памте јер им се називи и бирају према њиховој намени. Сваки произвођач процесора уз свој процесор даје и листу мнемоника који се користе. Асемблерски програм често се назива и изворни¹ програм, јер ће он касније бити преведен на машински објектни програм. Асемблерска верзија хексадецималног програма за сабирање два броја (за процесор MC68HC11) је:

<i>Меморијска адреса</i>	<i>Садржај меморијске адресе</i>	<i>Мнемоник инструкције</i>	<i>Коментар</i>
D200	B6	LDAA \$1001	; Узми први податак
D2001	10		
D202	01		
D203	BB	ADDA \$1002	; Додај други податак
D204	10		
D205	02		
D206	B7	STAA \$1003	; Смести збир у меморију
D207	10		
D208	03		
DE09	3E	WAI	; Заустави се

Свака линија садржи једну комплетну инструкцију. Подсетимо се да

¹ Source

се свака инструкција може састојати од једног, два или три бајта.

У колони „меморијска адреса” налазе се само адресе бајта операционог кода за сваку инструкцију. На пример, операциони кођ за LDAA инструкцију биће на адреси D200₁₆, а адреса операнда на меморијским адресама D201₁₆ и D202₁₆. Операциони кођ за ADDA биће на адреси D203₁₆, итд.

Подсетимо се да колона „мнемоник” даје мнемоник и адресу операнда за сваку инструкцију. Знак за долар (\$) испред адресе операнда значи да је она (адреса) дата у хексадецималном кођу. Према томе, LDAA\$1001 је асемблерска инструкција са значењем: „напуни акумулатор садржајем меморијске локације са хексадецималне адресе 1001”, док STAA\$1003 значи: „напуни меморијску локацију на адреси \$1003 садржајем акумулатора”.

Колона „коментар” често се користи за преглед асемблерских програма (листинзима) да би било очигледније шта програм ради.

3.16.1 Асемблирање програма

Асемблерски изворни програм лакше се пише и лакше прати од машинског, али се он на крају ипак мора превести на машински, пре него што се учита у меморију и изведе. Тај посао преводјења (конверзије) назива се асемблирање изворног програма.

Један од могућих метода за овај процес је ручно асемблирање, што једноставно значи да за сваки мнемоник програмер треба да пронађе одговарајући кођ и тиме замени сваку линију асемблерског програма машинским кодом, који се затим уноси у меморију и извршава. На пример, да би се асемблирала прва линија из горњег примера треба пронаћи операциони кођ за мнемоник LDAA.



У овом примеру изворни кођ се асемблира у три бајта хексадецималног објектног кода. Овакав начин асемблирања је дуготрајан, заморан и подложен грешкама.

Асемблери

Асемблирање може да уради и сам рачунар. Да би рачунар то могао да уради он мора да има специјални програм који се назива асемблер. Асемблер се може налазити у ROM меморији (резидентни асемблер), или се може учитати у RAM меморију са диска. Величина резидентног асемблера варира у границама од 2-16kB, у зависности од броја опција и могућности које поседује. Микрорачунари углавном користе мини-асемблере (16 до 32kB), којима се само копира процес ручног асемблирања. Асемблери, једноставни за рад и који подржавају синтаксу виших језика а нису резидентни заузимају преко 100MB на развојном рачунару.

Типичан процес асемблирања може се представити следећим корацима:

1. Мини-асемблерски програм смешта се у RAM меморију (ако већ није у ROM меморији);
2. Програмер уноси асемблерски програм у RAM меморију, или директно са тастатуре или са диска;
3. Даје се команда рачунару да изврши асемблерски програм;
4. Изворни програм се конвертује у објектни кођ (програм), који се смешта у RAM меморију;
5. Објектни кођ се може запамтити, тако да асемблирање не мора да се понавља сваки пут када желимо да извршимо програм;
6. Објектни програм може да се изврши.

Мане асемблерског језика

Асемблерски језик је свакако велико побољшање у односу на машински језик, поготову ако се на рачунару налази и асемблерски програм. Ипак, програмирање у асемблеру није баш лако јер захтева детаљно познавање микрорачунара који се користи. Треба познавати: регистре, инструкцијски скуп процесора, начине адресирања меморије и улазно/излазних уређаја и још много других детаља. Још једна мана асемблерских језика је да они нису преносиви; програм написан за један процесор неће радити на другом.

3.16.2 Виши програмски језици

Да би се превазишле поменуте мане програмирања у асемблеру развијени су програмски језици вишег нивоа. Језик вишег нивоа

омогућава програмирање рачунара коришћењем речи енглеског језика и математичких симбола, без потребе да се познаје интерна архитектура рачунара или његов скуп инструкција. У вишим језицима једна линија кода може имати више корака који одговарају инструкцијама машинског језика. За илустрацију довољно је погледати једну линију кода у језику C:

$$\text{RESULT} = A + B$$

Очигледно је да ова инструкција вишег језика представља програм за сабирање два броја, који би се у машинском језику састојао од инструкција које би рачунару указале да треба да узме један број, стави га у акумулатор, узме други број, дода га акумулатору и резултат смести у меморију. Такође је очигледно да је овакав програм много лакше написати и разумети него што је то могуће у машинском или асемблерском језику.

У употреби се налази велики број виших програмских језика, од којих је сваки пројектован за решавање специфичних проблема. Најчешће коришћени језици данас су: C, C++, C# и Java.

Главни задатак било ког вишег језика је да ослободи програмера бриге о детаљима програмирања на машинском или асемблерском језику. Уместо тога, програмер се може концентрисати на решавање проблема користећи се директним командама и правилима виших језика.

Друга предност је што се исти програм може изводити на рачунарима базираним на различитим фамилијама процесора, под претпоставком да они „разумеју” језик који се користи у програму.

3.16.3 Интерпретери и преводиоци

Као што је коришћење асемблерског језика захтевало постојање посебног асемблерског програма, тако и коришћење вишег језика захтева специјални програм. Тај специјални програм може бити класификован или као интерпретер или као преводилац¹. Оба су много већа него асемблерски програм и захтевају много више меморијског простора на развојном рачунару.

Интерпретери и преводиоци разликују се по процесу који користе при преводу вишег језика. Интерпретер узима једну по једну инструкцију

¹ Компјailer (енг. *compiler*)

3. Увод у рачунаре

вишег језика (нпр. $RESULT = A + B$) и позива потпрограм који обавља тражену секвенцу операција. Преводаца све инструкције преводи у низ машинских инструкција и ствара објектни код у меморији, који се затим и извршава.

Другим речима, интерпретер одмах извршава програм без креирања комплетног објектног кода, превodeћи све инструкције сваки пут када се извршавају (зато је и спорији). Преводаца све инструкције преведе само једном и смести их у меморију као објектни код који се онда покреће сваки пут када је то потребно.

И виши језици имају својих мана које их чине некорисним за неке примене. Они обично стварају програме који се извршавају много спорије и захтевају много више меморије од машинских или асемблерских програма.

Један од разлога је тај што се преводјење на машински код врши према правилима која су оптимизована за општи случај, па се према томе не могу користити све предности детаљног познавања рада једног рачунара, које програмеру омогућавају да сваки проблем друкчије реши, користећи минималан број инструкција. Други разлог је што процес преводјења ипак захтева неко време (истина код преводиоца је то само једном ако је програм без грешака). Интерпретери су посебно спори, са временом извршења које је 10 до 50 пута дуже него што би било да је програм написан у асемблеру.

Поред тога, виши језици нису баш прилагођени великој класи микрорачунарских апликација које се односе на управљање процесима који треба да раде у реалном времену. Овакве апликације захтевају да се излазним уређајима шаљу подаци и управљачке информације, а примају подаци и статусне информације од улазних уређаја. Врло често су управљачке и статусне информације величине неколико бита и односе се на хардвер. Виши језици су пројектовани да лако раде са децималним бројевима, и много су неефикаснији од асемблера када се ради са битовима.

У неким апликацијама брзо извођење програма је од суштинског значаја, тако да је спорост виших језика велика мана. Апликације као што су управљање машина и процеса или комуникације захтевају да процесор изврши програм најбрже што може. Називају се и апликације у реалном времену, јер програм управља процесом или догађајима који се одвијају

у моменту када се он извршава. Асемблер је још увек незаменљив за ове апликације због велике брзине извршавања.

3.17 ПИТАЊА И ЗАДАЦИ

1. Како може 8-битни процесор да смести 24-битну реч податка?
2. Одређени микрорачунар поседује следећи формат инструкције:
 - операциони код: 8 битова,
 - адреса операнда: 16 битова.
 - а) Колико различитих инструкција се може обезбедити?
 - б) Колико меморијских локација се може адресирати?
3. Зашто се предбајт (као део инструкцијског скупа) користи код MC68HC11?
4. Које три магистрале су део сваког микрорачунара? Која од њих је једносмерна?
5. Које три операције процесор обавља увек када извршава неки програм?
6. Описати 16-битне регистре процесора.
7. Где аритметичко/логичка јединица смешта резултат операција?
8. Која инструкција се користи за учитавање података из меморије у акумулатор?
9. Која инструкција се користи за смештање података из акумулатора на одређену меморијску адресу?
10. Испитати асемблерски програм који следи и одговорити на следећа питања:

Меморијска адреса	Меморијска реч
C000	B6
C001	A0
C002	07
C003	B0
C004	A0
C005	07
E23D	3E

- а) За време извршавања програма колико пута се адреса \$A007 појављује на адресној магистрали?

3. Увод у рачунаре

- б) Колико пута процесор извршава операцију читавања (READ) меморије?
 - в) Колико пута процесор извршава операцију уписивања (WRITE) меморије?
 - г) Колико пута се нова реч упише у инструкцијски регистар?
 - д) Колико пута се нова реч упише у регистар адресе податка?
 - ђ) Колико пута се нова реч учита у акумулатор А?
 - е) Који је садржај акумулатора А на крају овог програма?
11. Које инструкције се користе за промену редоследа извршавања процесора?
 12. Какву улогу има инструкција скока JMP?
 13. Описати како се користи инструкције условног скока.
 14. Како инструкција BEQ користи ознаке (флегове) регистра услова?
 15. Какву улогу има инструкција скока JMP?
 16. На којој адреси ће се наставити извршавање програма уколико је текући садржај про грамског бројача \$A000, резултат претходне операције је 0, извршава се инструкција BEQ а одступање је \$0A?
 17. Објаснити разлику између фирмвера и софтвера?
 18. Каква је предност програмирања у асемблерском језику у поређењу са машинским језиком?
 19. Навести неке од мана програмирања у асемблерском језику.
 20. Који се виши програмски језици данас најчешће користе?

4. Структура и начин рада микрорачунара

У 3. поглављу описали смо како микрорачунар узима и извршава програм смештен у меморији. Наш задатак био је да анализирамо начин рада који се одвија под контролом процесора. У опису смо намерно изоставили детаљне електричне везе, сигнале и времена, да би поједноставили анализу. У овом поглављу додаћемо још неке детаље који су неопходни за разумевање начина рада микрорачунара. Користићемо поједностављену верзију процесора MC68HC11, а остале компоненте биће анализирани у 5. поглављу.

4.1 ЕЛЕМЕНТИ МИКРОРАЧУНАРА

Веома је важно поново погледати разлику између микрорачунара (μC) и микропроцесора (μP^1). Микрорачунар садржи неколико елемената, као што је илустровано на слици 4.1. Главни елемент је процесор, који је једно LSI интегрисано коло и који садржи аритметичко/логичку и управљачку јединицу микрорачунара. Због густине паковања већина процесора у једном интегрисаном колу реализована је у NMOS, CMOS и HCMOS технологији. Технологија HCMOS допринела је повећању брзине, смањењу потрошње и високој отпорности на сметње.

Меморијска јединица се код готово свих микрорачунара састоји од RAM и ROM меморијских модула. RAM меморијски модул могу сачињавати једно коло или више RAM меморијских интегрисаних кола, комбинованих тако да се добије жељени капацитет. RAM меморијски део користи се за смештање података и програма који се могу мењати за време рада рачунара. Такође се може користити за смештање

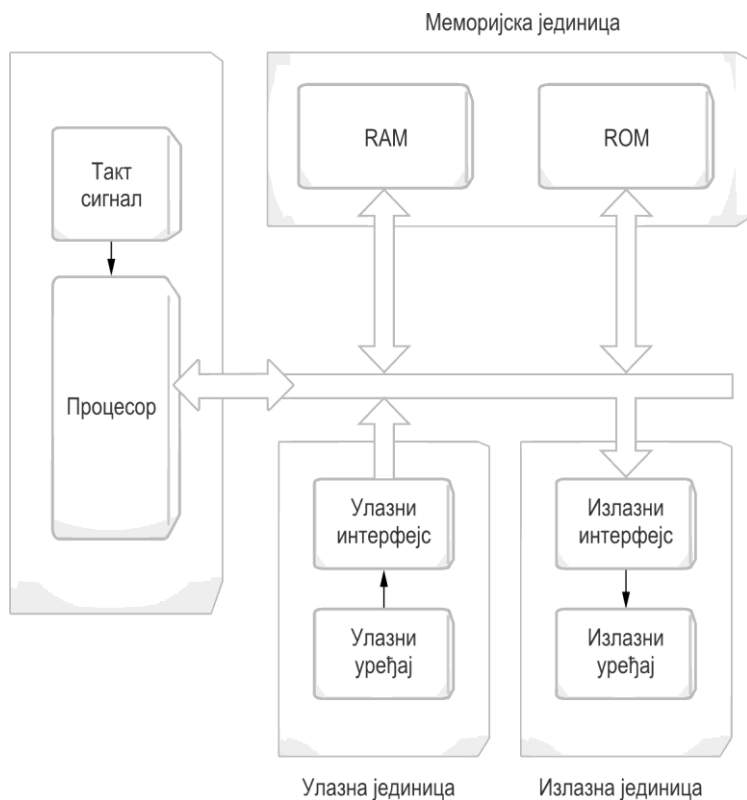
¹ Када се анализира микропроцесор као модул, често се користи скраћеница μP ; када се анализира његов рад у оквиру микрорачунара користи се скраћеница MPU (*MicroProcessor Unit*); када се анализира рад микроконтролера користи се скраћеница MCU (*Microcontroller Unit*).

4. Структура и начин рада микрорачунара

међурезултата и коначних резултата за време извршавања програма.

ROM меморијски део састоји се од једног или више интегрисаних кола, а смештају се подаци и програми који се не могу мењати. На пример у ROM меморији могу бити смештени програми за комуникацију са тастатуром или табела ASCII кодова потребних за слање података ка штампачу.

Улазни и излазни део садржи интерфејсна кола која су потребна за комуникацију улазно/излазних модула и осталих делова рачунара. Интерфејсна кола су интегрисана кола која су произвођачи пројектовали за прикључивање различитих врста улазно/излазних модула. У неким случајевима интерфејсно коло може да буде једноставно као што је бафер регистар.



Слика 4.1 Основни елементи микрорачунара

Процесор је LSI интегрисано коло које обједињује функције управљачке јединице (узимање, интерпретирање и извршавање) и аритметичко/логичке јединице, али нема меморију и улазно/излазна интерфејсна кола која су потребна да би се направио комплетан рачунар. Микрорачунар је значи процесор са меморијом и улазно/излазним модулима.

Како се LSI технологија константно развија и укључује све више компонената у једном интегрисаном колу тако произвођачи нуде микрорачунар у једном интегрисаном колу – микроконтролер (нпр. сва интегрисана кола из фамилије MC68HC11). Они садрже процесор, одређену количину RAM меморије, ROM меморије, EEPROM меморије, часовник¹, А/Д конвертор и нека улазно/излазна кола. Микроконтролер у једном интегрисаном колу садржи све елементе потребне да ради као потпуни микрорачунар, иако су некад потребни додатна меморија и улазно/излазни модули да се прошире његове могућности.

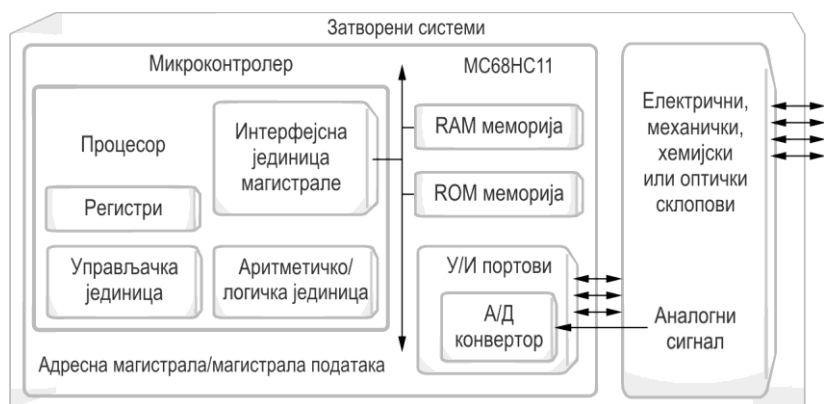
4.1.1 Затворени системи

Затворени микрорачунарски систем² састоји се од микрорачунара који је повезан са механичким, хемијским или електричним уређајима, програмиран за специфичну намену и реализован као целовит систем. Најчешће је реч о апликацијама у реалном времену са свим обавезама које то условљава. Било који електрични, механички или хемијски систем који читава улазне податке, који треба да одлучује о нечему, који треба да изврши нека израчунавања или који треба да изврши неке анализе могуће је реализовати као затворен рачунарски систем. Интерфејси претварају сигнале добијене од сензора у облик који микрорачунар може да прихвати. На пример тахометар је сензор који мери брзину обртања и генерише напон пропорционалан брзини. Микрорачунарски софтвер доноси потребне одлуке, израчунавања и анализе. Додатни електронски интерфејси претварају излазе микрорачунара у потребан облик. Микроконтролер као што је MC68HC11 садржи: процесор, меморију и улазно/излазне модуле. Захваљујући томе микрорачунар базиран на микроконтролеру MC68HC11 захтева минимални број додатних компонената. Софтвер за затворене

¹ *Timer*

² *Embedded microcomputer system*

4. Структура и начин рада микрорачунара



Слика 4.2 Пример затвореног система

системе обично разрешава ограничен скуп задатака. Микрорачунар је затворен или скривен унутар уређаја. С друге стране рачунари опште намене, као што је персонални рачунар, могу да обављају различите задатке у зависности од програма.

Затворени системи управљају већином уређаја који се данас користе. Физичка величина затворених система креће се у широком опсегу од дигиталних часовника и МРЗ¹ уређаја за слушање музике до великих стационарних уређаја као што су семафори, контролери у фабрикама или системи који управљају нуклеарним електранама. Комплексност затворених система мења се од једноставних система у које је уграђен само један микроконтролер до веома сложених система са великим бројем међусобно повезаних микроконтролера.

Док су неки затворени системи веома сложени, део тих система захтева минималну величину меморије и програма, не садржи оперативни систем а софтвер је минималне комплексности. Уобичајени улазно/излазни модули су: прекидачи, релеји, соленоиди, LED диоде, мали или наменски LCD² монитор, радио-примопредајник, сензори температуре, влажности, нивоа светлости... Затворени системи обично немају тастатуру, екран,

¹ Скраћеница од *MPEG-1 Audio Layer 3* – формат који се користи за компресију дигиталних аудио сигнала. *Moving Picture Experts Group* (MPEG) је радна група интернационалних организација за стандардизацију ISO/IEC која се бави развојем стандарда за кодирање аудио и видео сигнала.

² *Liquid Cristal Display* – екран (монитор) који садржи течни кристал.

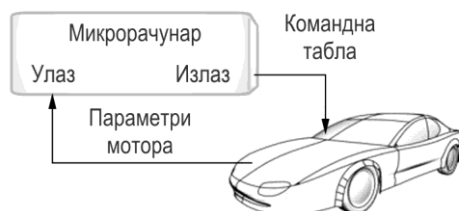
диск или било коју другу, за персоналне рачунаре уобичајену, улазно/излазну јединицу.

4.2 ТИПИЧНА ПРИМЕНА МИКРОРАЧУНАРА

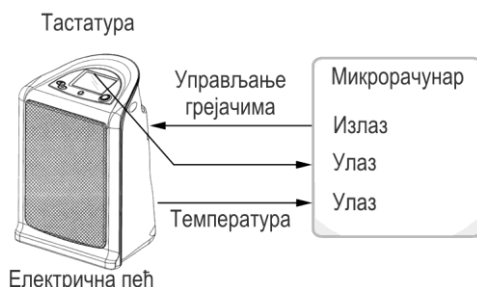
У типичној примени микрорачунар прима улазну информацију или податак, извршава секвенцу инструкција (које на неки начин обрађују улазне информације), а затим генерише излазну информацију или податак. Улазно/излазне операције, наравно, укључују повезивање са спољашњим светом. Функције обраде извршавају се под контролом програма који је смештен у меморији (RAM, ROM).

На пример, када микрорачунар контролише командну таблу мотора аутомобила (слика 4.3а) улазна информација се састоји од режима рада мотора као што су температура, притисак уља, потрошња горива... Свака од ових физичких величина претвара се у електричну величину преко неког претварача. Електрична величина се затим претвара у дигиталну форму и преноси у микрорачунар где се обрађује и претвара у погодну форму за приказивање и шаље на излаз, ка командној табли.

а)



б)



Слика 4.3 Примена микрорачунара

4. Структура и начин рада микрорачунара

Други пример је електрична рерна којом се управља помоћу микрорачунара (слика 4.36). Улазне податке уноси корисник преко тастатуре да укаже микрорачунару на жељену температуру и колико дуго треба рерна да буде укључена. Микрорачунар прима ову улазну информацију и извршава програм који генерише излазни сигнал, који управља грејачима рерне. Температура рерне претворена је у електрични сигнал а затим у дигиталну форму и један од улаза микрорачунара. На основу упоређивања величине сигнала са температурног улаза и жељене температуре микрорачунар управља грејачима.

4.3 ОРГАНИЗАЦИЈА МИКРОРАЧУНАРА

Организација микрорачунара односи се на начин на који су елементи микрорачунара међусобно повезани. Организација највећег броја 8-битних микрорачунара слична је оној која је дата на слици 4.4 на примеру¹ са процесором MC68HC11. Микроконтролер MC68HC11 може да ради на више начина (MOD²). Један од њих - нормалан проширени начин³ омогућава да се микроконтролеру може додати спољашња меморија и улазно/излазни модули. Дијаграм на слици 4.4 показује како су основни елементи процесора повезани међусобно, уз коришћење три магистрале. Иако овај дијаграм изгледа комплексан, он још увек не показује све електричне везе. Све информације и сигнале везане за активност процесора преносе три магистрале. Процесор је непрекидно укључен у процес слања (уписивање) или пријема (очитавање) података ка/из меморије или улазно/излазних модула или уређаја.

4.3.1 Портови В и С

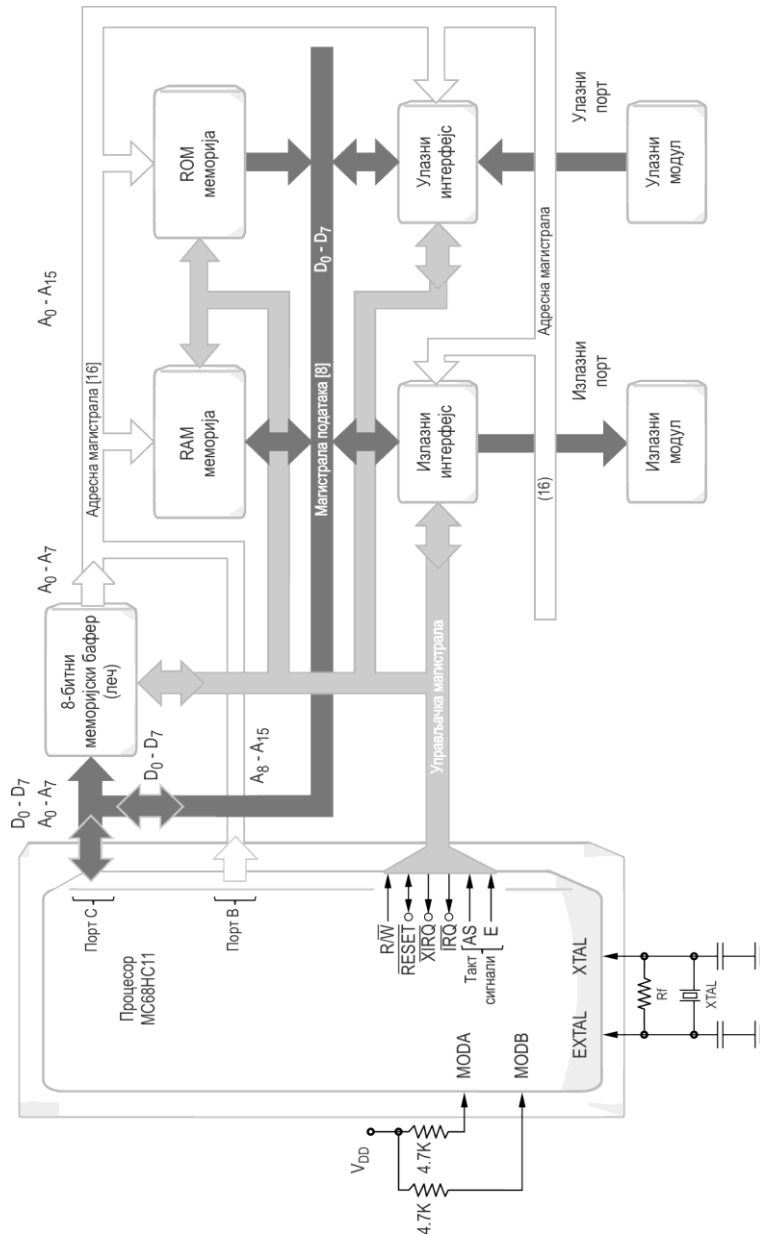
Процесор (микроконтролера) MC68HC11 има пет различитих портова⁴: А, В, С, D и Е. У овом поглављу анализираћемо само В и С

¹ На слици 4.4 дат је поједностављени дијаграм.

² На енглеском *mode*. У овом уџбенику користићемо ознаку MOD због лакшег усаглашавања са оригиналном документацијом произвођача.

³ *Normal expanded mode*

⁴ Портовима се генерално назива интерфејс ка „спољашњем свету“ неког уређаја. Пример су серијски и паралелни портови персоналног рачунара. У случају микроконтролера MC68HC11 термин се односи на прикључке интегрисаног кола преко којих се дигитални подаци преносе од и ка микроконтролеру.



Слика 4.4 Организација микрорачунара са микроконтролером MC68HC11

4. Структура и начин рада микрорачунара

портове. Процесор MC68HC11 поставља се у нормални проширени режим рада када су његови улази MODA и MODB постављени на вредност 1 (табела 4.1). Порт В преузима улогу адресних линија вишег реда ($A_8 - A_{15}$) адресне магистрале. Порт С има различиту намену у зависности од стања Е-такт сигнала и контролног (управљачког) адресног импулса (AS^1):

- улогу адресних линија нижег реда ($A_0 - A_7$) адресне магистрале или
- двосмерне (бидирекционе) магистрале података ($D_0 - D_7$).

Порт	A	Једно интегрисно коло		Проширена врста рада и тестирање	
В	0	PB0	Излаз	A_8	Адресни излаз
В	1	PB1	Излаз	A_9	Адресни излаз
В	2	PB2	Излаз	A_{10}	Адресни излаз
В	3	PB3	Излаз	A_{11}	Адресни излаз
В	4	PB4	Излаз	A_{12}	Адресни излаз
В	5	PB5	Излаз	A_{13}	Адресни излаз
В	6	PB6	Излаз	A_{14}	Адресни излаз
В	7	PB7	Излаз	A_{15}	Адресни излаз
С	0	PC0	Улаз/Излаз	A_0	Мултиплексирање адреса/подаци
С	1	PC1	Улаз/Излаз	A_1	Мултиплексирање адреса/подаци
С	2	PC2	Улаз/Излаз	A_2	Мултиплексирање адреса/подаци
С	3	PC3	Улаз/Излаз	A_3	Мултиплексирање адреса/подаци
С	4	PC4	Улаз/Излаз	A_4	Мултиплексирање адреса/подаци
С	5	PC5	Улаз/Излаз	A_5	Мултиплексирање адреса/подаци
С	6	PC6	Улаз/Излаз	A_6	Мултиплексирање адреса/подаци
С	7	PC7	Улаз/Излаз	A_7	Мултиплексирање адреса/подаци
		STRA	Улазни импулс ²	AS	Контролни адресни импулс ⁴
		STRB	Излазни импулс ³	R/W	Одабир: учитавање или уписивање

Табела 4.1 Намена портова у зависности од врсте рада

¹ Adress Strobe

² Input Strobe

³ Output Strobe

⁴ Input Strobe

За вишенаменску употребу порта С примењује се временско мултиплексирање. То је техника која омогућава да се порт С у различитим временским интервалима понаша као адресна магистрала или магистрала података. За памћење битова нижег реда A_0 - A_7 адресне магистрале користи се 8-битни лек бафер.

4.3.2 Адресна магистрала

Адресна магистрала састоји се од 16 излаза који представљају 16-битну адресу која се шаље из процесора ка свим меморијама и улазно/излазним модулима. Подсетимо се да процесор може сам да постави логичке нивое на адресну магистралу. Шеснаестобитном адресом може се адресирати 65536 локација, било да су то RAM меморија, ROM меморија или улазно/излазни модули (склопови¹).

Сваки пут када процесор жели да комуницира са (нпр. READ-из, или WRITE-ка) одређеном меморијском локацијом или улазно/излазним модулом, мора да постави адресни код на прикључке порта С (A_0 - A_7) и порта В (A_8 - A_{15}). Адресни битови се затим декодују да би селектовали жељену меморијску локацију или улазно/излазни модул. Овај процес декодовања захтева кола за декодовање, о чему ће бити говора касније.

За време операције читавања пинови D_0 – D_7 порта С имају функцију улаза преко којих се прихватају подаци које су на магистралу података поставили селектована меморија или улазно/излазни модули.

4.3.3 Управљачка магистрала

Управљачку магистралу сачињавају улазни и излазни сигнали који имају улогу усаглашавања (синхронизације) рада процесора и других делова микрорачунара. У овом делу анализираћемо само оне сигнале управљачке магистрале који се најчешће срећу код микрорачунарских система.

- Сигнал за читавање/уписивање² ($R/\overline{W}$) логички је сигнал који процесор генерише да би осталим деловима микрорачунара указао на смер којим се преносе подаци преко магистрале података. За операцију читавања користи се сигнал $R/\overline{W}=1$, а за уписивање $R/\overline{W}=0$. Ово дозвољава меморији и улазно/излазним модулима да

¹ Device

² Read/Write

одговоре правилно, зависно да ли процесор шаље податке, или очекује да прими податке.

- Управљачки адресни импулс (AS) има функцију команде за упис података у лек (8-битног бафера за држање адреса).
- Сигнал $\overline{\text{RESET}}$ може имати улогу улазног или излазног сигнала зависно од тога да ли њиме управљају унутрашња или спољашња кола процесора. Без обзира како се њиме управља овај сигнал је обично висок, а низак је када се процесор тек укључује, или када се жели довести систем у дефинисано почетно стање. Процесор реагује на $\overline{\text{RESET}}$ сигнал иницијализацијом интерних кола и пуњењем програмског бројача адресом прве инструкције коју треба извршити. Друга важна функција $\overline{\text{RESET}}$ сигнала је задржавање вредности улаза MODA и MODB. Када $\overline{\text{RESET}}$ сигнал прелази са нижег на виши логички ниво вредност улаза MODA и MODB се памти у специјалним улазним регистрима процесора. На основу њих процесор одређује начин рада. Једном када је начин рада постављен може се мењати само после поновне промене на $\overline{\text{RESET}}$ улазу. Највише пажње биће посвећено анализи нормалног проширеног начина рада (MODA=MODB=1).
- Захтев за прекид ($\overline{\text{XIRQ}}$, $\overline{\text{IRQ}}$) чине улазни прикључци који када су активни имају намену да „скрену пажњу“ или прекину рад процесора. Најчешће их користе улазно/излазне јединице за пренос података. Прикључак омогућава немаскирајући прекид. У току иницијализације (после укључивања) процесор неће реаговати на прекиде док се они софтверски не омогуће. Улаз се обично користи за прекид када се детектује пад напона напајања, а улаз за асинхроно прекидање процесора MC68HC11 ради обављања неке временски критичне активности.

4.3.4 Такт сигнали

Микроконтролер MC68HC11 као такт сигнал¹ система (системски такт сигнал) користи Е-такт сигнал. Он се генерише у самом интегрисаном колу MC68HC11. Учестаност Е-такт сигнала одређена је спољашњим

¹ Clock signal

колом које је повезано на улазе интегрисаног кола EXTAL и XTAL. Коло се састоји од: кристала кварца и одређеног броја међусобно повезаних отпорника и кондензатора. На слици 4.5 приказана су два начина везивања кристала кварца¹. Прикључци осцилатора могу се повезати са кристалним осцилатором или споља генерисаним извором такт сигнала који је CMOS компатибилан. На ове прикључке доводи се четири пута већа учестаност такт сигнала од жељене учестаности (брзине Е-такт сигнала). Е-такт сигнал користи се као основни такт сигнал у односу на који се сви сигнали референцирају. Када је Е-такт сигнал низак (адресни део циклуса магистрале) дешавају се интерни процеси а када је Е-такт сигнал висок адресирају се подаци. Е-такт сигнал је активан под условом да је осцилатор активан (STOP сигнал зауставља рад свих такт сигнала).

Да би се омогућила „штедња“ енергије постоје одговарајући начини рада² (WAIT и STOP) којима се програмски управља. Ова особеност чини MC68HC11 фамилију нарочито атрактивном за системе који се напајају из батерија.

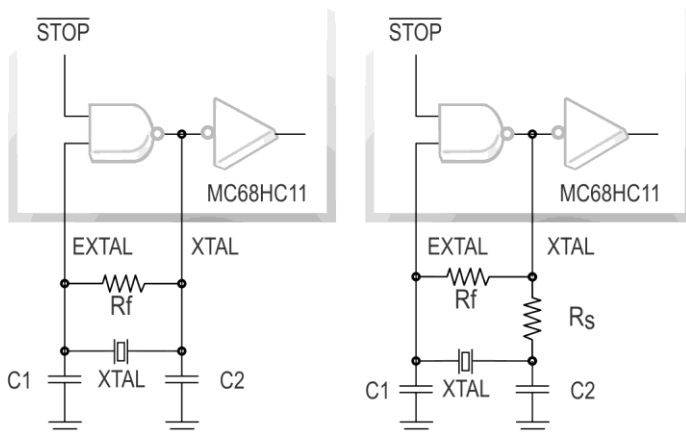
Осцилатор MC68HC11 користи НИ коло чији је један од улаза повезан са сигналимa који искључују осцилатор када је MC68HC11 у STOP режиму рада. Други улаз је EXTAL улазни прикључак микроконтролера. Излаз НИ кола је XTAL излазни прикључак микроконтролера и он се оставља отвореним када се користи спољњи CMOS компатибилан такт сигнал. На слици 4.5 приказане су унутрашње и спољашње компоненте које чине кристални осцилатор³.

Слика 4.5а приказује начин повезивања за кристал више учестаности (веће од 1MHz), а слика 4.5б начин повезивања за рад на нижим учестаностима (мањим од 1MHz). У типичним применама користи се кварц учестаности 8MHz, чиме се добија Е-такт сигнал учестаности 2MHz.

¹ Препоручена од стране Motorola [M68HC11 reference manual]

² STOP и WAIT се односе на режим рада процесора а разликују се од проширеног начина рада и начина рада једног интегралног кола што представљају режим рада самог интегралног кола микроконтролера.

³ Назива се *Pierce* осцилатор. Такође је познат и као паралелни резонантни кристални осцилатор (*crystal oscillator*).



Слика 4.5 а) повезивање кристала за више учестаности
б) повезивање кристала за ниже учестаности

Постоје и други такт сигнали који су интерни (искључиво се користе унутар процесора MC68HC11 нпр. сигнали PH2 и PH1 (слика 4.6).

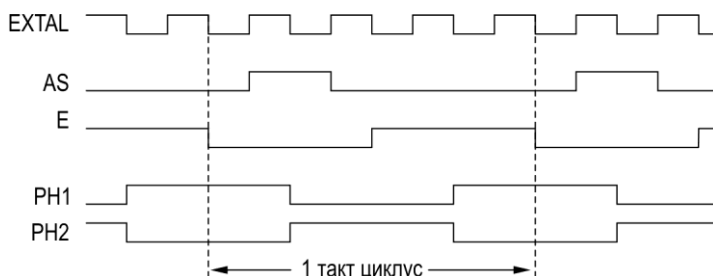
Такт сигнали синхронизују сваку активност која се дешава унутар и ван процесора MC68HC11. На пример програмски бројач се инкрементира са позитивном ивицом PH2 сигнала, а смештање података са магистрале података у акумулатор A са негативном ивицом Е-такт сигнала. Док је Е-такт сигнал на ниском нивоу сигнал AS је на високом нивоу. За то време подаци о адреси биће постављени на адресну магистралу преко портова В и С.

Такт циклус процесор MC68HC11 започиње силазном ивицом Е-такт сигнала и завршава се његовом улазном ивицом. Ови сигнали су непреклапајући.

Сигнали PH1 и PH2 су сигнали исте учестаности али фазно померени. Може се видети да се улазна ивица PH2 јавља после силазне ивице PH1 и обрнуто. За PH1 и PH2 сигнале каже се да су непреклапајући сигнали.

4.3.5 Повезивање процесора MC68HC11 са улазно/излазним модулима

Дијаграм на слици 4.4 показује један улазни и један излазни модул (склоп, блок). У пракси постоји неколико улазно/излазних склопова



Слика 4.6 Такт сигнали и њихов међусобни однос код MC68HC11

повезаних са процесором. Процесор MC68HC11 може да комуницира са улазно/излазним скоповима преко магистрале података на исти начин као са RAM или ROM меморијом.

Обично улазно/излазни модул не може бити повезан директно на магистралу података, већ преко интерфејсног кола. Интерфејсно коло се користи кад год улазно/излазни модул користи напонске нивое, временске сигнале или формат података различит од оног који користи процесор MC68HC11.

На пример, типични улазни модул је тастатура која серијски шаље ASCII кодиране податке који представљају притиснуте тастере. Магистрала података процесора прихвата податке само у облику 8 паралелних битава. Интерфејсно коло се користи да конвертује серијске улазне податке у паралелне.

4.3.6 Улазно/излазни портови

Место где се улазни модул везује за интерфејсно коло (слика 4.4) зове се улазни порт. Слично, место где је излазни модул везан за његово интерфејсно коло је излазни порт. Већина процесора (микроконтролера) има неколико улазно/излазних портова.

На пример MC68HC11 има пет портова (A, B, C, D и E). Сваки од њих пројектован је да прихвати одређени тип улазно/излазних уређаја (нпр. паралелни порт за штампач, серијски порт или SPI ¹ порт).

¹ *Serial Peripheral Interface Bus*

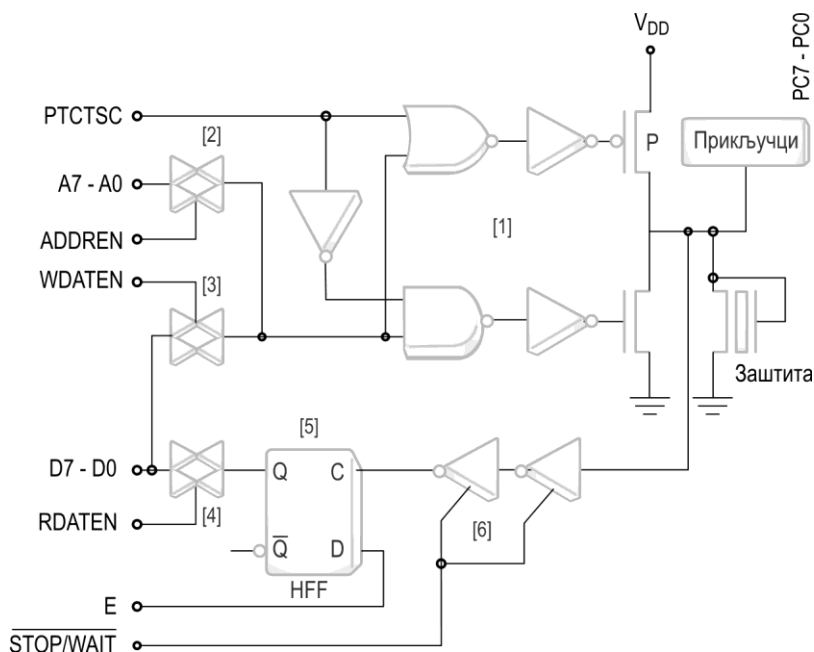
4.4 ВРЕМЕНСКИ ДИЈАГРАМ ПРИ УПИСИВАЊУ И ОЧИТАВАЊУ

Процесор MC68HC11 стално обавља операције уписивања и очитавања док извршава програм. Операцију очитавања (READ) процесор извршава сваки пут кад ради следеће:

1. узима операциони код (део инструкције) из програмске меморије (RAM или ROM меморија);
2. узима адресу операнда (део инструкције) из програмске меморије;
3. узима податке из меморије или улазног блока за време фазе извршења инструкције (нпр. LDAA, ADDA, SUBA).

Процесор извршава операцију уписивања (WRITE) када ради једну од следећих операција:

1. преноси податке из интерног регистра процесора (као што је акумулатор) ка локацији RAM меморије (на пример STAA инструкција);



Слика 4.7 Начин повезивања прикључака порта С у проширеном начину рада

2. преноси податке из интерног регистра процесора ка излазном модулу.

Обзиром на број операција узимања инструкција, које процесор мора да одради за време извршења програма, број операција читавања је већи него број операција уписивања.

У проширеном начину рада (као што је већ поменуто) порт С је временски мултиплексирана адресна магистрала или магистрала података. За време прве половине циклуса адресе се шаљу са порта С. За време друге половине циклуса подаци се уписују на порт С или читавају у порт С (слика 4.7). Излазни бафери прикључака [1] могу бити или селектовани или неселектовани помоћу PTCTSC сигнала. Овај сигнал је на напонском нивоу логичке нуле када адреса или податак треба да буду послати преко порта С.

Када је PTCTSC на напонском нивоу логичке јединице овај излазни бафер није селектован и излазни прикључци порта налазе се у стању високе импедансе (док се подаци читавају у порт С). Информације које се шаљу преко порта С могу да се пренесу преко једног од излаза¹ [2] или [3]. Када је сигнал за селектовање адресе (ADDREN²) на активном нивоу излаз за пренос [2] је селектован, да би повезао адресне линије ADDR[7:0] на излазни бафер одговарајућих прикључака порта С.

Када је сигнал за селектовање уписивања (WDATEN³) на напонском нивоу логичке јединице излаз за пренос [3] је селектован да би омогућио повезивање линија за податке DATA[7:0] на излазни бафер одговарајућих прикључака порта С. Када је читавање, подаци долазе са порта С на улазни бафер [6]. Подаци из бафера [6] затим се уписују у HFF⁴ [5] за време позитивног дела Е-такт сигнала. Подаци се уписују у процесор преко улаза за упис [4] док је сигнал за селектовање уписа (RDATEN⁵) на активном нивоу. Пошто процесор читава податке за време једне четвртине циклуса и пошто Е-такт сигнал падне на низак напонски ниво,

¹ *Transmission gate*

² *Address enable*

³ *Write data enable*

⁴ HFF (*Half Flip-Flop*) је лач који је транспарентан док је улазни такт сигнал висок, а памти (печује) док је такт сигнал низак

⁵ *Read data enable*

4. Структура и начин рада микрорачунара

HFF [5] такође омогућава одређени ниво синхронизације за долазеће податке. У нормалној врсти рада сигнал заустави/чекај¹ (STOPWAIT) је на напонском нивоу логичке јединице бафери [6] нису селектовани. Када је микроконтролер у врсти рада заустави/чекај са малом потрошњом², сигнал STOPWAIT је на напонском нивоу логичке нуле и бафери [6] су селектовани. Док су бафери селектовани [6] међунивои и нивои³, услед пребацивања са једног на други напонски ниво, на прикључцима порта С неће изазвати релативно велике струје што се нормално очекује за CMOS улазе.

У проширеном начину рада временски распоред порта С укључује четири врсте операција. Операција уписивања је идентична, независно у коју се адресу уписује. Други тип операције је читавање са екстерне адресе. Последње две операције су читавање из интерне адресе (меморијске локације или регистра унутар микроконтролера MC68HC11). За тестирање подаци се читавају из интерних локација да би могли да се надгледају помоћу логичког анализатора.

У нормалном раду подаци се не читавају из интерних регистра да не би дошло до колизије са спољашњим модулима.

Постоји интерни управљачки бит (IRV⁴) процесора MC68HC11 који одређује да ли ће (или неће) подаци интерних регистра и меморије бити уписани у порт С.

Логика унутар процесора MC68HC11 генерише сигнале PTCTSC, AD-DREN, WDATEN, и RDATEN да би управљала активностима на порту С у зависности од операција које треба извршити. На порту В обезбеђен је виши бајт адресе у трајању од једне четвртине циклуса у односу на силазну ивицу Е-такт сигнала. На слици 4.8 приказан је идеализован временски распоред ових сигнала.

4.4.1 Временски дијаграми за операцију читавања

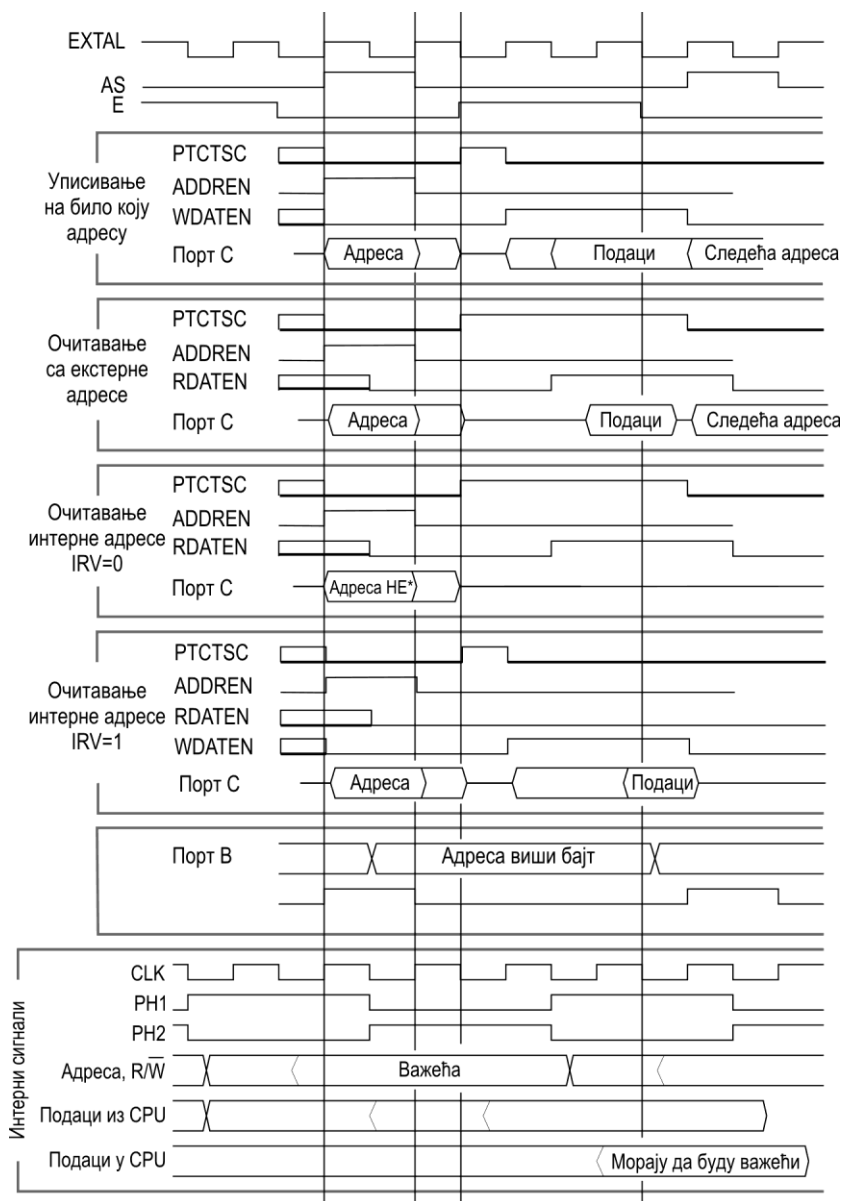
Сада смо спремни да на примеру операције читавања детаљније анализирамо временску повезаност између различитих сигнала на

¹ Stop/wait

² Low-power modes

³ Intermediate or switching levels

⁴ Internal read visibility control bit



Слика 4.7 Идеалан временски распоред сигнала на портovima C и B

4. Структура и начин рада микрорачунара

магистралама процесора MC68HC11. Све активности везане су за такт сигнале PH2, E и AS, као што је приказано временским дијаграмом на слици 4.9. Временски дијаграм приказује активности на управљачким линијама $R/\overline{W}$ и AS, порта В и порта С за време једног такт циклуса у коме процесор извршава операцију читавања. Подсетимо се да се преко порта В преноси виши бајт адресе, а да порт С у различитим временским тренуцима преноси и нижи бајт адресе и податке. Постоји пет критичних временских тачака које су везане за пет ивица такт сигнала.

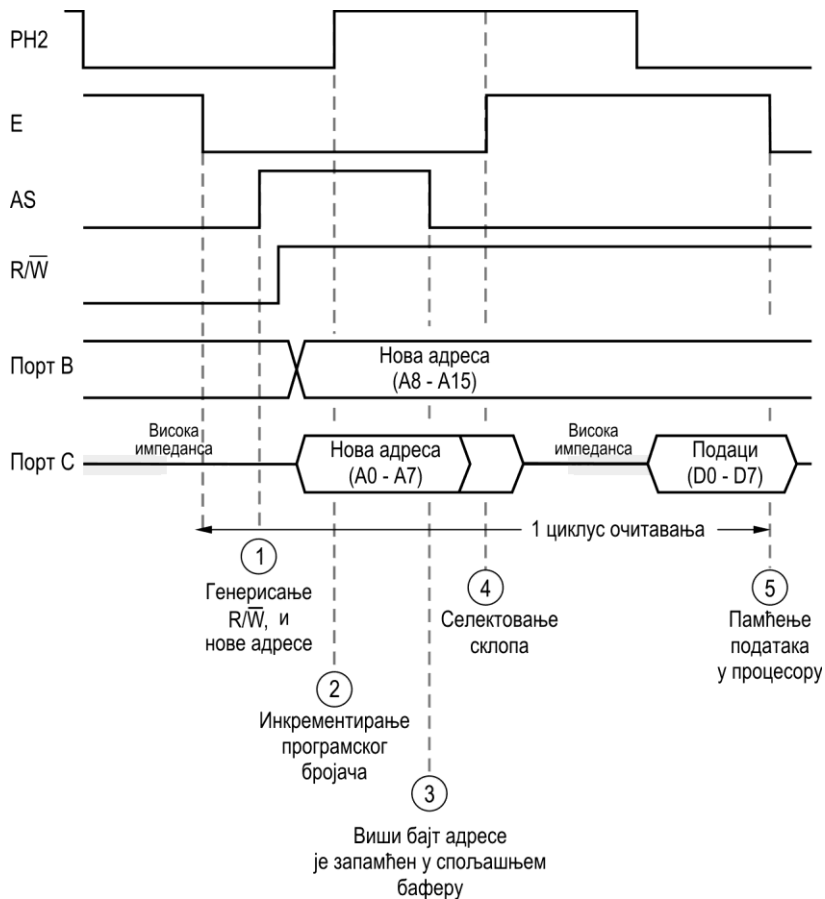
1. Тачка 1 је узлазна (растућа ивица) AS сигнала. Претпоставићемо да је $R/\overline{W}$ сигнал постављен пре појаве промене (транзиције) такт сигнала. Сматраћемо и да се до овог тренутка на порту В налазио „стари“ део адресе (A_8-A_{15}) коју је процесор поставио у претходном такт циклусу. Магистрала података (порт С) биће у стању високе импедансе (високо Z), односно у пливајућем¹ стању, пошто ниједан од модула везаних за магистралу података неће добити дозволу за повезивање² (тј. неће бити селектован) до друге половине READ циклуса.

Када се појави растућа ивица AS сигнала процесор одговара постављањем $R/\overline{W}=1$ да би сигнализирао операцију читавања (READ) и постављањем нове адресе (A_0-A_7) на порт С и (A_8-A_{15}) на порт В. Уочимо да је дијаграм на слици незнатно померен у односу на тачку 1 због пропагационог кашњења интерних кола процесора. Такође уочимо да облик сигнала на адресној магистрали (порт В и С) указује на могуће транзиције (1-0 и 0-1) пошто ће се неке од 16 адресних линија мењати у једном правцу, а неке у другом.

2. Ниједан од сигнала на магистрали не мења се у тачки 2, на растућој ивици PH2. Процесор користи ову ивицу такта да би инкрементирао програмски бројач, интерни регистар процесора.
3. Тачка 3 је опадајућа ивица AS сигнала. У овом тренутку се нижи бајт адресе који долази са порта С памти у 8-битном баферу. Нова адреса (нижи бајт са излаза бафера и виши бајт на порту

¹ Floating

² Enabled



Слика 4.9 Активност на магистрали за време циклуса читавања

В) сада се налази на адресној магистрали и селекује меморију или улазно/излазни модул за операцију читавања. Убрзо после опадајуће ивице AS сигнала процесор уклања нижи бајт адресе.

- Тачка 4 је опадајућа ивица Е-такт сигнала. Када је Е-такт сигнал на високом напонском нивоу само одабрана меморија и улазно/излазни модул биће селектовани. На магистрали података појавиће се подаци (неке линије магистрале су на нивоу логичке јединице, а друге на нивоу логичке нуле).

Меморија или улазно/излазни модули који нису селектовани немају никаквог утицаја на магистралу података пошто се налазе у стању високе импедансе.

5. На силазној ивици Е-такт сигнала (тачка 5) процесор узима податке са магистрале података (порт С) и памти их у неком од својих унутрашњих регистара, као што су на пример инструкцијски регистар или акумулатор. За исправан рад неопходно је да селектована меморија или улазно излазни модули поставе податке пре опадајуће ивице Е-такт сигнала.

Пример 4.1: После селектовања подаци се на излазу одређене врсте PROM меморије појављују са кашњењем од 500ns. Да ли се ова меморија може користити код MC68HC11 који ради са такт сигналом од 2MHz?

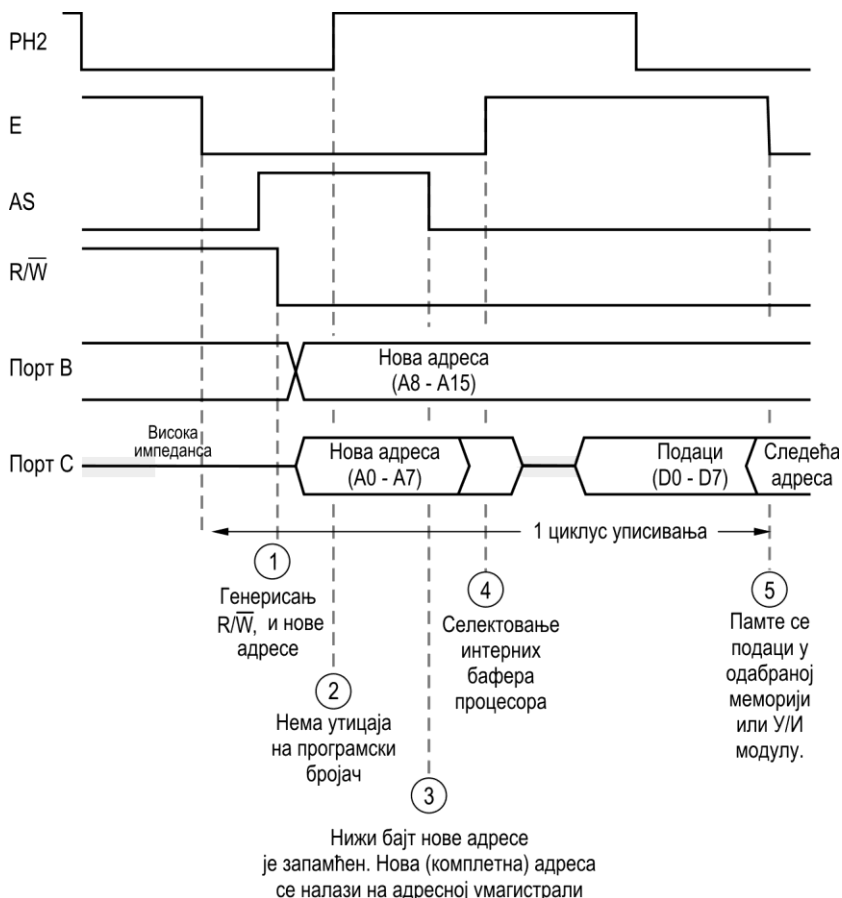
Решење:

Не. Са 2MHz такт сигналом трајање импулса (виши ниво) Е-такт сигнала биће око 250ns. Опадајућа ивица Е-такт сигнала наступиће пре него што се подаци из PROM меморије појаве на магистрали података. Пошто се на магистрали података нису појавили „прави” подаци (из адресиране локације PROM меморије) неки случајни садржај биће уписан у интерне регистре процесора.

4.4.2 Временски дијаграм операције уписивања

На слици 4.10 приказане су активности на магистрали за време трајања једног такт циклуса процесора, у коме она обавља операцију уписивања (WRITE). Означимо још једном 4 критична тренутка која одговарају ивицама такт импулса:

1. Претпоставићемо да је $R/\overline{W}=1$ пре растуће ивице AS сигнала. Магистрала података (порт С) биће у „пливајућем стању” (стање високе импедансе) пошто ниједан модул неће бити селектован док се не појави друга половина Е-такт сигнала. Када се растућа ивица AS сигнала појави временско/управљачка логика процесора поставља $R/\overline{W}=0$ и указује да је на реду операција уписивања (WRITE). Такође поставља нову адресу на адресну магистралу (портови В и С). Ова нова адреса селектује



Слика 4.10 Активност на магистрали за време циклуса уписивања

меморију или улазно/излазни модул у који процесор треба да упише податке.

2. Растућа ивица PH2 сигнала не користи се за увећавање програмског бројача као што је то било за време циклуса читавања, пошто се програмски бројач не користи за време операције уписивања. Подсетимо се да се регистар адресе податка (а не програмски бројач) користи да обезбеди меморијску адресу за време операције уписивања (нпр. инструкција STAA).

3. Када се појави опадајућа ивица AS сигнала (тачка 3) нижи бајт адресе операнда (са порта C) биће запамћен у спољњем 8-битном баферу на исти начин као код операције читавања. Комплетна адреса која се сада налази на адресној магистрали селекује меморију или улазно/излазни модул у који процесор треба да упише податке. Непосредно после силазне ивице AS сигнала на порту C ће се налазити неисправни подаци¹.
4. Када се појави растућа ивица Е-такт сигнала (тачка 4) излазни подаци ће из његових интерних регистара бити постављени на магистралу података (порт C). Излазни бафери, меморија или улазно/излазни модули се налазе у стању високе импедансе пошто је у току операција уписивања.
5. По појави опадајуће ивице Е-такт сигнала подаци са магистрале података (порт C) памте се у селекованој меморији или улазно/излазном модулу. Пошто Е-такт сигнал прелази на низак ниво процесор ће послати следећу адресу на портове В и С. Та адреса се односи на следећу операцију читавања. Треба имати на уму да за време операције уписивања програмски бројач већ садржи следећу адресу. Ова адреса је „припремљена” у претходној операцији читавања.

4.5 АКТИВНОСТИ ЗА ВРЕМЕ ИЗВРШАВАЊА ИНСТРУКЦИЈА

Сада када смо се упознали са временским сигналимa за време читавања и уписивања размотрићемо активности на магистрали података за време извршавања инструкција. За илустрацију основне идеје није потребно анализирати активности на магистрали за све инструкције, већ само за неке типичне. За сваку од инструкција могу се користити каталози² за комплетан опис активности на магистралама. Подсетимо се да посматрамо нормалан проширен начин рада микроконтролера MC68HC11.

Део асемблерског програма који ћемо анализирати је:

¹ *Invalid data*

² *MC68HC11 Reference Manual*

Адреса	Лабела	Машински код	Мнемоник	Коментар
EF00	START	B0	SUBA \$0201	; Одузети податке са адресе
EF01		02		; \$0201 од [A] и резулатат
EF02		01		; ставити у акумулатор
EF03		B7	STAA \$0202	; Смести садржај [A] на
EF04		02		; адресу \$0202
EF05		02		
EF06		??	??????	; Следећи операциони код

Почиње од адресе \$EF00 и састоји се од SUBA и STAA инструкција. За сваку инструкцију користићемо мнемоник асемблерског језика. Инструкција SUBA \$0201 одузима од садржаја акумулатора A садржај меморијске локације \$0201. Операциони код \$B0 налази се на адреси \$EF00, а адреса операнда (\$0201) налази се на адресама \$EF01 и \$EF02. Знак „\$” користи се да означи да је број који следи (у нашем примеру двобајтна адреса операнда) представљен у хексадецималном бројном систему. Инструкција STAA \$0202 смешта садржај акумулатора у меморијску локацију чија је адреса \$0202.

Овај програмски сегмент захтева осам такт циклуса за своје комплетно извршавање. Они су нумерисани од један до осам на слици 4.11. За сваки од ових такт циклуса дијаграм показује адресу која се појављује на адресној магистрали (порт В и порт С), податак на магистрали података (порт С) и ниво на управљачкој линији $R/\overline{W}$. Вредности на адресној магистрали и магистрали података изражене су у хексадецималним бројевима.

Претпоставићемо да је MC68HC11 у фази извршавања програма и да је спреман да са адресе \$EF00 учита (узме¹) операциони код. То значи да је текући садржај програмског бројача \$EF00 ($[PC]=EF00_{16}$). Претпоставићемо да меморијска локација \$0201 садржи податак \$2A и да је садржај акумулатора $[A]=12_{16}$.

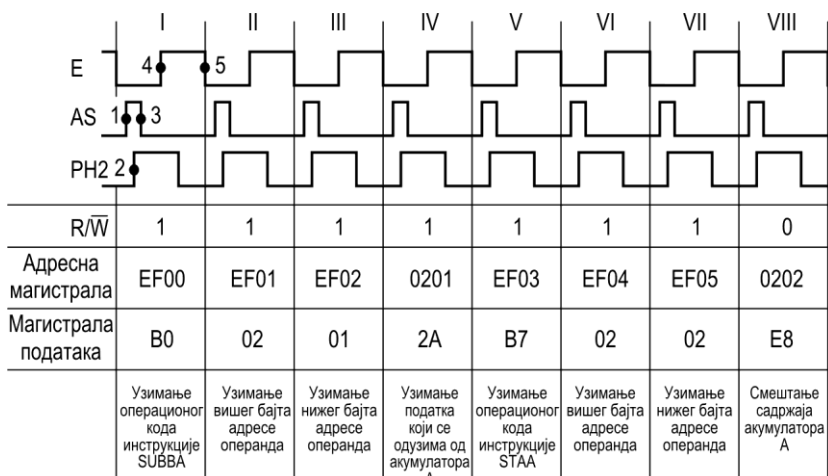
У опису који следи поделићемо такт циклус на пет делова (1, 2, 3, 4 и 5) који одговарају ивицама такт сигнала приказаним на сликама 4.9 и 4.10.

¹ Fetch

4. Структура и начин рада микрорачунара

I. такт-циклус

- Када се појави растућа ивица AS сигнала процесор поставља садржај програмског бројача $[PC]=EF00_{16}$ на адресну магистралу, и поставља управљачку линију $R/\overline{W}=1$ за операцију читавања.
- Процесор инкрементира садржај програмског бројача на вредност $\$EF01$ ($[PC]=EF01_{16}$) када се појави силазна ивица PH2 сигнала. Ово не утиче на адресну магистралу пошто је $\$EF01$ запамћен у меморији са струјним појачавачима која је смештена између спољашње и унутрашње магистрале.
- Када се појави силазна ивица AS сигнала нижи бајт адресе, 00, смешта се са порта C у спољашњи 8-битни бафер, који са друге стране управља адресном магистралом. На адресној магистрали сада се налази $\$EF00$.
- Меморијска локација $\$EF00$ селектована је растућом ивицом Е-такт сигнала, и поставља податак $\$B0$ на магистралу података (порт C).
- Када се појави опадајућа ивица Е-такт сигнала процесор узима реч податка и смешта је у инструкцијски регистар (IR) пошто је то



Слика 4.11 Активности на магистралама за време извршавања дела асемблерског програма

операциони код. Када се операциони код нађе у инструкцијском регистру он се декодује и процесор „сазнаје“ да је реч о тробајтној инструкцији одузимања. Даље, значи, треба узети из меморије двобајтну адресу операнда.

II. такт-циклус

1. Садржај програмског бројача \$EF01 постављен је на интерну адресну магистралу и $R/\overline{W}=1$ (операција узимања).
2. Садржај програмског бројача је инкрементиран на \$EF02.
3. Нижи бајт адресе, \$01, смешта се са порта С у спољни 8-битни бафер. На спољној (екстерној) адресној магистрали је \$EF01.
4. Меморијска локација \$EF01 је селектована и она поставља свој садржај, виши бајт адресе - \$02, на магистралу података.
5. Процесор узима виши бајт адресе са магистрале података и смешта га у виши бајт регистра адресе податка (DAR). Подсетимо се да се регистар адресе податка користи за адресирање операнда, што ће бити потребно у фази извршавања инструкције.

III. такт-циклус

1. Садржај програмског бројача \$EF02 постављен је на интерну адресну магистралу и $R/\overline{W}=1$ (операција узимања).
2. Садржај програмског бројача инкрементиран је на \$EF03.
3. Нижи бајт адресе, \$02, смешта се са порта С у спољни 8-битни бафер. На спољној адресној магистрали је \$EF02.
4. Меморијска локација \$EF02 је селектована и она поставља свој садржај, нижи бајт адресе - \$01, на магистралу података.
5. Процесор узима нижи бајт адресе са магистрале података и смешта га у нижи бајт регистра адресе податка.

IV. такт-циклус

1. Процесор поставља садржај регистра адресе податка ($[DAR]=0201_{16}$) на адресну магистралу и поставља $R/\overline{W}$ да би узео операнд из меморије.
2. У овом циклусу се за адресирање не користи програмски бројач већ регистар адресе податка. Због тога се програмски бројач не инкрементира. У претходном, III циклусу, он је инкрементиран на

4. Структура и начин рада микрорачунара

вредност \$EF03 и припремио је свој садржај за следећу операцију узимања из програмског дела меморије.

3. Бајт мање тежине адресе операнда, \$02, смешта се у спољњи 8-битни бафер. На адресној магистрали сада је адреса \$0201.
4. Меморијска локација \$0201 је селектвана, и ставља своју реч податка \$2A на магистралу података.
5. Процесор прихвата овај податак и смешта га у регистар података (DR) који је директно везан са аритметичко/логичком јединицом (погледати слику 3.6). Аритметичко/логичка јединица одузима садржај регистра податке (\$2A) од садржаја акумулатора (\$12). Резултат (\$E8) аритметичко/логичка јединица смешта у акумулатор.

V. такт-циклус

1. Садржај програмског бројача $[PC]=FE03_{16}$ постављен је на интерну адресну магистралу и $R/\overline{W}=1$ за узимање операционог кода.
2. Процесор инкрементира садржај програмског бројача на вредност \$EF04 ($[PC]=EF04_{16}$).
3. Нижи бајт адресе, \$00, смешта се са порта C у спољашњи 8-битни бафер, који с друге стране управља адресном магистралом. На адресној магистрали сада се налази \$EF00.
4. Меморијска локација \$EF03 је селектвана растућом ивицом Е-такт сигнала и поставља податак \$B7 на магистралу података (порт C).
5. Процесор узима податак са магистрале података и смешта га у инструкцијски регистар (IR) пошто је то операциони код. Када се операциони код нађе у инструкцијском регистру он се декодује и види се да је реч о тробајтној инструкцији STAA. Потребно је из меморије узети двобајтну адресу операнда.

VI. такт-циклус

1. Садржај програмског бројача \$EF04 постављен је на интерну адресну магистралу и $R/\overline{W}=1$ (операција узимања).
2. Садржај програмског бројача је инкрементиран на \$EF05.
3. Нижи бајт адресе, \$04, смешта се са порта C у спољњи 8-битни бафер. На адресној магистрали је \$EF04.

4. Меморијска локација \$EF04 је селектована и она поставља свој садржај, виши бајт адресе \$02, на магистралу података.
5. Процесор узима виши бајт адресе са магистрале података и смешта га у виши бајт регистра адресе податка (DAR). Подсетимо се да се регистар адресе податка користи за адресирање операнда, што ће бити потребно у фази извршавања инструкције.

VII. такт-циклус

1. Садржај програмског бројача \$EF05 постављен је на интерну адресну магистралу и $R/\overline{W}=1$ (операција узимања).
2. Садржај програмског бројача је инкрементиран на \$EF06.
3. Нижи бајт адресе, \$05, смешта се са порта C у спољњи 8-битни бафер. На спољњој адресној магистрали је \$EF05.
4. Меморијска локација \$EF05 је селектована и она поставља свој садржај, нижи бајт адресе \$01, на магистралу података.
5. Процесор узима нижи бајт адресе са магистрале података и смешта га у нижи бајт регистра адресе податка.

VIII. такт-циклус

1. Процесор поставља садржај регистра адресе података ($[DAR]=0202_{16}$) на адресну магистралу и поставља $R/\overline{W}=0$ да би уписао (WRITE) садржај акумулатора A у меморију.
2. У овом циклусу се за адресирање не користи програмски бројач већ регистар адресе податка. Због тога се програмски бројач не инкрементира. У претходном, VII циклусу, он је инкрементиран на вредност \$EF06 и припремио је свој садржај за следећу операцију узимања из програмског дела меморије.
3. Бајт мање тежине адресе операнда, \$02, смешта се у спољњи 8-битни бафер. На адресној магистрали сада је адреса \$0202.
4. Садржај акумулатора A ($[A]=E8_{16}$) поставља се на магистралу података.
5. Податак се смешта у селектовану меморијску локацију \$0202 и на тај начин се завршава извршавање инструкције STAA\$0202.

4.6 АДРЕСНИ ПРОСТОР МИКРОРАЧУНАРА

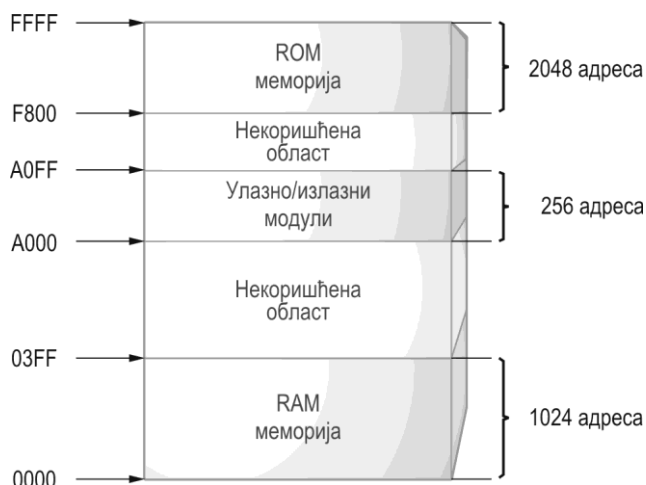
Када процесор извршава операције читавање (READ) или уписивање (WRITE), потребно је да постави адресе на адресну магистралу да би се селектовала једна и само једна меморијска локација, или улазно/излазна локација. Већина 8-битних процесора има 16-битну адресну магистралу која може да специфицира било коју од 65536 различитих адреса. У хексадецималном систему ове адресе су у опсегу од \$0000 до \$FFFF; овај адресни опсег назива се адресни простор микрорачунара. Уобичајено се најнижа адреса \$0000 назива дно адресног простора, а највиша адреса \$FFFF врх адресног простора.

Пројектант микрорачунара треба да подели овај адресни простор између RAM меморије, ROM меморије, и улазно/излазних адреса које треба да буду део система. Начин на који је укупни адресни простор расподељен између ових модула зависи, донекле, од карактеристика процесора. У многим случајевима неки од избора које пројектант прави могу бити произвољни. На слици 4.12 приказано је како адресни простор може бити расподељен код једноставних система са микроконтролером MC68HC11.

4.6.1 Меморијска мапа

Овај дијаграм назива се меморијска мапа иако укључује адресе улазно/излазних модула као и адресе RAM и ROM меморија. Дно адресног простора, од \$0000 до \$03FF₁₆, додељено је RAM меморији. То значи да постоји 1024₁₀ речи (1K) смештених у RAM меморију. Врх адресног простора од F800₁₆ до FFFF₁₆ додељен је ROM меморији; тако, постоји 2048₁₀ речи (2K) смештених у ROM меморији. Адресе од \$A000 до \$A0FF₁₆ додељене су улазно/излазним модулима. Укупно 256 локација резервисано је за улазно/излазне модуле, али као што ћемо видети, то не значи да постоји 256 различитих улазно/излазних модула.

Преостали део адресног простора не користи се; то значи, адресе у опсегу \$0400-\$BFFF и \$A100-\$F7FF нису за сада додељене ниједном модулу. Наравно, неискоришћене адресе могу се користити за додатне улазно/излазне или меморијске модуле.



Слика 4.12 Пример меморијске мапе за микрорачунаре са микроконтролером MC68HC11 (проширени начин рада)

4.6.2 Меморија и улазно/излазни модули у интегрисаном колу

Додељивање меморије знатно је сложеније за процесор MC68HC11 него за процесоре које нису делови микроконтролера. Подсетимо се да је MC68HC11 микроконтролер (MCU¹) интегрисано коло које не садржи само процесор већ и меморију и улазно/излазне модуле, који су унутар интегрисаног кола (интерно) повезани са процесором. Ово омогућава микроконтролеру MC68HC11 да ради као контролер (једно интегрисано коло) без икакве потребе за спољном (екстерном) меморијом или улазно/излазним модулима. То значи да ће део 64K адресног простора бити додељен улазно/излазним модулима и RAM, ROM и EEPROM меморији на самом интегрисаном колу. Величина адресног простора који се додељује меморији и улазно/излазним модулима на интегрисаном колу зависи од верзије микроконтролера MC68HC11 који се користи. У табели 4.2 је листа чланова фамилије микроконтролера MC68HC11. Свака верзија микроконтролера састоји се од истог процесора MC68HC11 (који смо до сада описали) и различитих величина RAM, ROM и EEPROM меморија. У овом уџбенику анализираћемо верзију MC68HC11A8 пошто су остале верзије из ње проистекле.

¹ Microcontroller Unit

4. Структура и начин рада микрорачунара

Као што се из табеле види микроконтролер MC68HC11A8 има: ROM меморију величине 8kB¹, EEPROM меморију величине 512B, RAM меморију величине 256B и 64B додељена улазно/излазним регистрима. Свакој од компонената биће додељен одговарајући адресни простор из опсега 64kB. Меморијама ROM и EEPROM додељен је тачно дефинисан адресни простор. Адресни простор за RAM меморију и улазно/излазне модуле може се мењати². У сваком случају потребно је познавати који адресни простор заузимају модули унутар самог интегрисаног кола да би се избегло да се исти адресни простор додели интерним и екстерним меморијским и улазно/излазним модулима. Такође је потребно познавати како се модулима може омогућити или забранити

<i>Интегрисано коло</i>	<i>Програмска меморија</i>	<i>Величина EEPROM меморије</i>	<i>Величина RAM меморије</i>	<i>Број улазно/ излазних прикључака</i>
MC68HC11A8	8k ROM	512	256	40
MC68HC11A1		512	256	40
MC68HC11D0	-	-	192	32
MC68HC11D3	4k ROM	-	192	32
MC68HC711D3	4k EPROM	-	192	32
MC68HC11E0	-	-	512	36
MC68HC11E1	-	512	512	36
MC68HC811E2	-	2048	512	36
MC68HC11E9	12k ROM	512	512	36
MC68HC11E20	12k ROM	512	768	36
MC68HC711E9	12k EPROM	512	512	36
MC68HC711E20	20k EPROM	512	768	36
MC68HC11F1	-	512	1024	54
MC68HC11K4	24k ROM	640	768	62

Табела 4.2 Чланови фамилије микроконтролера MC68HC11

¹ Ознака значи: k→1024, B→бајт

² Користи се специјалн регистар CONFIG.

рад унутар интегрисаног кола или како им се може доделити други адресни простор у оквиру меморијске мапе.

Пример 4.2 *Одредити број RAM меморијских адреса за меморијску мапу на слици 4.12.*

Решење:

Постоје најмање два начина да се ово уради. Први метод укључује претварање прве и последње адресе у децималну. Хексадецималне RAM адресе су у опсегу 0000_{16} до $03FF_{16}$, и $0000_{16} = 0_{10}$ и $03FF_{16} = 1023_{10}$. Тако, постоји 1024 адреса.

Други метод захтева претварање прве и последње адресе у бинарну на следећи начин:

	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
$0000_{16} =$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$03FF_{16} =$	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1

Уочимо да 6 битова највеће тежине остају на 0, док других 10 битова обухватају комплетан опсег од свих нула до свих јединица. Тако, постоји $2^{10} = 1024$ различитих адреса.

4.6.3 Меморијске странице

65536 адреса су подељене у 256 блокова од по 256 адреса, где се сваки блок назива страница. Слика 4.13 показује како је адресни простор организован по страницама. Свака страница састоји се од свих хексадецималних адреса које почињу са две исте цифре. На пример, страна 00 укључује 256 адреса у опсегу $\$0000$ – $\$00FF$. Слично, страна 6A укључује хексадецималне адресе $\$6A00$ – $\$6AFF$. Користећи организацију са страницама можемо посматрати адресе као да се састоје од редног броја странице и редног броја речи. Да бисмо то илустровали посматрајмо адресу $\$1E66$:

1 E	6 6
└───┘	└───┘
редни број странице	редни број речи

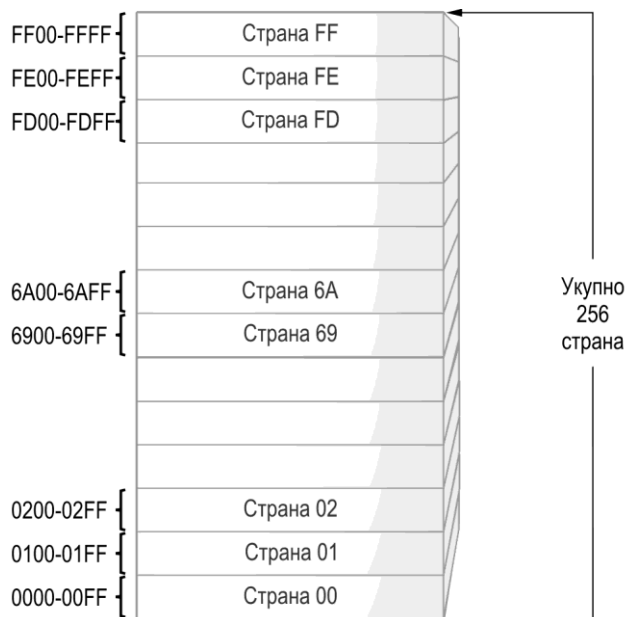
4. Структура и начин рада микрорачунара

Две цифре са највећом тежином су редни број странице, а цифре мање тежине одређују редни број речи на тој страници.

Пример 4.3 *Одредити колико страница је додељено RAM меморији у меморијској мапи на слици 4.12 и то употребити за израчунавање укупног броја RAM адреса. Поновити за ROM меморију и улазно/излазне модуле.*

Решење:

RAM меморија заузима адресе \$0000-\$03FF; ово укључује странице 00, 01, 02 и 03 - укупно четири странице. Пошто сваку страницу чини 256 адреса, укупан број адреса RAM меморије је $4 \times 256 = 1024$. ROM меморија заузима странице \$F8, \$F9, \$FA, \$FB, \$FC, \$FD, \$FE и \$FF тако да се за 8 страница добија $8 \times 256 = 2048$ адреса. Улазно/излазни модули користи све адресе на страни \$A0, а то је укупно 256 адреса.



Слика 4.13 Адресни простор организован по страницама

Пример 4.4 *Одређени микрорачунар има 4096 локација RAM које почињу на адреси \$0000. Колико страница је додељено RAM меморији? Одредите комплетан адресни простор RAM меморије.*

Решење:

$4096/256 = 16$ страница. Странице су \$00, \$01, \$02, \$03,...,\$0C, \$0D, \$0E, \$0F. Комплетан адресни опсег је \$0000-\$0FFF.

Пример 4.5 *Одређени програм који је смештен у ROM меморији заузима целу страницу \$FD. Које адресе овај програм заузима?*

Решење:

\$FD00-\$FDFF.

4.7 МЕМОРИЈСКИ МОДУЛИ

Количина RAM и ROM меморије која је део микрорачунара мења се у широком опсегу, зависно од апликације за коју је рачунар намењен и зависно од цене. Рачунари опште намене као што су персонални рачунари имају преко 2GB RAM меморије и преко 1MB ROM меморије (BIOS¹). Мањи микрорачунари и микроконтролери (нпр. MC68HC11EVB² микроконтролер) који су намењени за апликације писане на асемблеру уобичајено имају 8kB RAM меморије и 8kB ROM меморије. Део микрорачунара са ROM меморијом састоји се од ROM, EPROM и EEPROM интегрисаних кола у којима су смештене 8-битне речи. Постоје интегрисана кола RAM меморије са различитим величинама речи: 1, 4 и 8-битова. Пројектанти могу да користе неколико интегрисаних кола RAM меморије да би добили модул RAM меморије са 8-битном величином речи.

Под RAM меморијским модулом подразумева се комбинација интегрисаних кола RAM меморије међусобно тако повезаних да могу да сместе 8-битну реч. На слици 4.14а приказано је како се могу повезати 1kX1 интегрисаних кола RAM меморије, да би се добио 1kX8 модул. Интегрисана кола су повезана тако да раде као једна целина.

Истих 10 линија са адресне магистрале доводе се на адресни улаз ка сваком интегрисаном колу. Сигнали за селектовање интегрисаних кола

4. Структура и начин рада микрорачунара

¹ *Basic Input/Output System*

² *Evaluation Board*

Слика 4.13 Адресни простор организован по страницама

међусобно су повезани за селектовање модула. Улази $R/\overline{W}$ интегрисаних кола међусобно су повезани као заједнички $R/\overline{W}$ улаз модула. Улазно/излазне линије за податке нису међусобно повезане већ заједно формирају 8-битну реч за магистралу података.

Слика 4.146 показује како су два 256X4 RAM меморијска интегрисана кола повезана тако да формирају 256X8 модул. Линије CS, $R/\overline{W}$ и адресни улази сваког интегрисаног кола међусобно су повезани. Четири улазно/излазне линије за податке из сваког интегрисаног кола формирају 8-битну реч.

Сва интегрисана кола у оквиру RAM меморијских модулу адресирају се истовремено, селекују истовремено и примају истовремено READ и WRITE сигнале. На тај начин скуп интегрисаних кола понаша се као само једно интегрисано коло RAM меморије траженог капацитета.

Пример 4.6 *Колико страница меморије је смештено у сваком интегрисаном колу меморијског модула на слици 4.14?*

Решење:

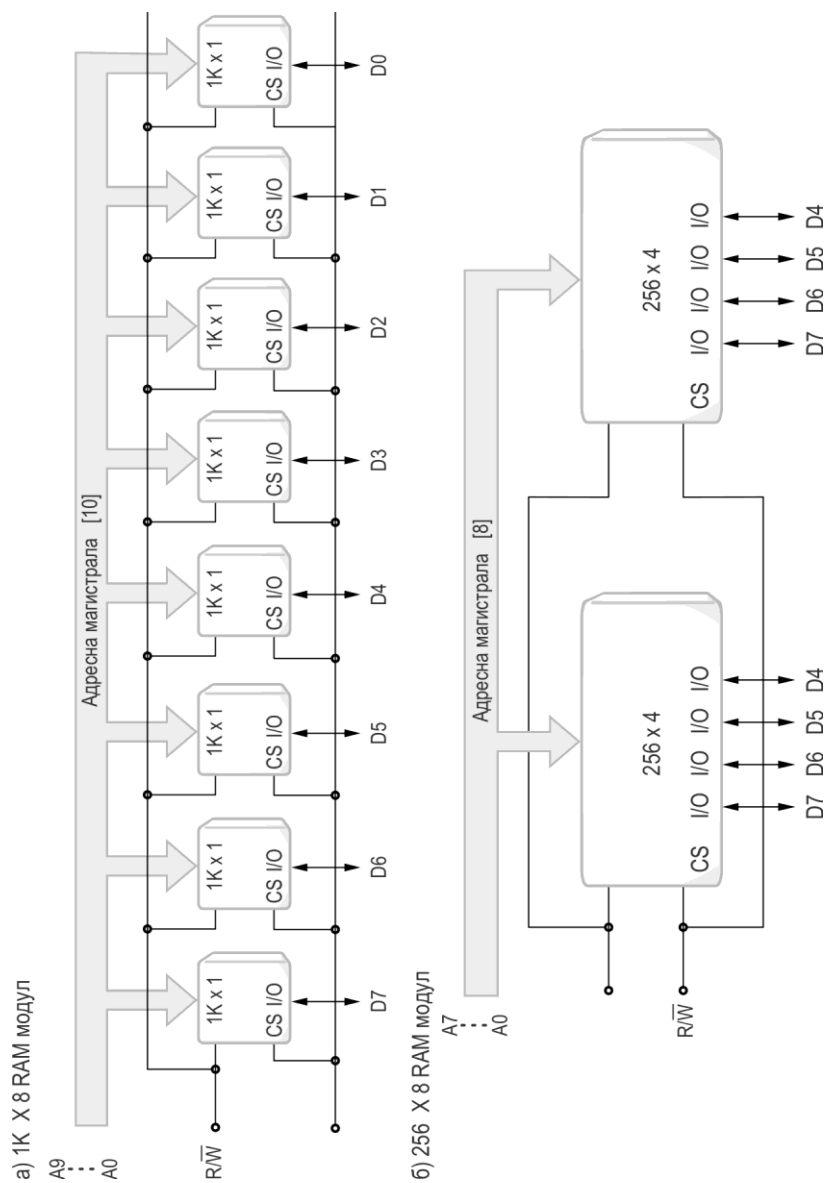
Меморијски модул 1kX8 (слика 4.14.а) садржи четири странице, пошто је једна страница величине 256X8. Меморијски модул 256X8 на слици 4.146 садржи једну страницу.

Пример 4.7 *Да ли бисте употребили RAM меморијско интегрисано коло величине 256X4 или 1kX1 уколико је потребна меморија капацитета две странице?*

Решење: Две странице су величине 512X8. Четири RAM меморијских интегрисаних кола величине 256X4 могу се повезати тако да се добију два 256X8 модула. RAM меморијско интегрисано коло величине 1kX1 не може се повезати тако да смести само 512 речи, пошто свако RAM меморијско интегрисано коло има на располагању 1024 адресе.

Да би се добио капацитет од 1kX8 потребно је искористити осам RAM меморијских интегрисаних кола, као што је приказано на слици 4.14а. Интегрисана кола не би била добро искоришћена пошто је потребно само пола од овог капацитета. Зато је комбинација четири

4. Структура и начин рада микрорачунара



Слика 4.14 Пример комбиновања интегрисаних кола RAM меморије 256X4 RAM меморије и интегрисаних кола 8 битних регистара.

Пример 4.8 *Описати организацију 16kX8 RAM меморијског модула која користи RAM меморијска интегрисана кола величине 4kX1.*

Решење:

Прво, осам меморијских интегрисаних кола величине 4kX1 комбиновано је тако да формирају меморијски модул 4kX8. Затим, четири модула величине 4kX8 повезују се тако да се добије потребан капацитет 16kX8.

Пример 4.9 *Који је највећи број 1kX8 модула који се могу користити са процесором MC68HC11?*

Решење:

Процесор MC68HC11 може да адресира $2^{16} = 65536$ адреса. Пошто је $1k = 1024$, онда је:

$$65536/1024 = 64$$

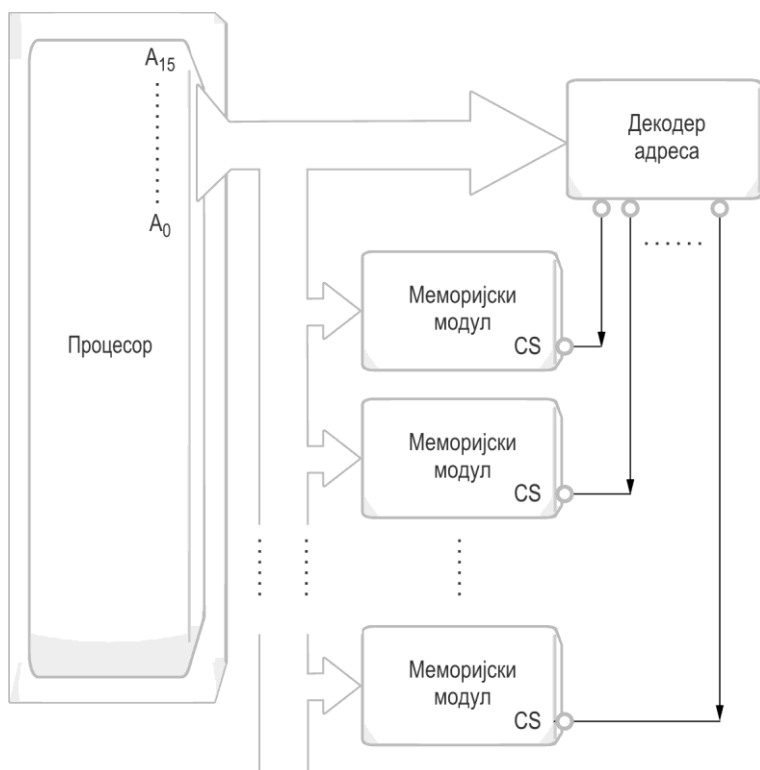
То значи да се 65536 адреса састоји од 64 блока величине 1k.

4.8 ДЕКОДОВАЊЕ АДРЕСЕ

Једном када пројектант микрорачунара одлучи какву ће комбинацију меморијских интегрисаних кола користити да би обезбедио жељени капацитет, следећи корак је да испројектује кола за декодовање адреса. Кола узимају адресу коју процесор поставља на адресну магистралу и конвертују у сигнале за селектовање интегрисаног кола, које меморијски модули и улазно/излазни склопови захтевају. Један и само један од ових сигнала за селектовање интегрисаног кола биће активиран за сваку од могућих адреса. Слика 4.15 показује општи принцип декодовања адресе.

Велики број адресних линија, које долазе из процесора, повезан је на улазе адресних декодера. Декодер одговара активирањем једног од својих излаза, који затим селекује један и само један меморијски модул¹. Генерално, користе се адресне линије веће тежине и број линија који се користе зависи од броја меморијских модула. На пример, уколико постоје четири меморијска модула, адресне линије A_{15} и A_{14} могу се користити као улази у декодер 1-од-4.

¹ Надаље ћемо користити термин меморијски модули не улазећи у то да ли су састављени од више интегрисаних кола или је то једно једино интегрисано коло.



Слика 4.15 Уобичајена организација декодовања адреса
($R/\overline{W}$ и магистрала података нису приказане)

Адресне линије мање тежине, које долазе из процесора, везане су на адресне улазе меморијског модула. Ове адресне линије декодује коло за декодовање у меморијским модулима, да би се селектовала једна меморијска локација којој се може приступити преко операција читавања (READ) и уписивања (WRITE).

Број адресних линија које се користе за ову намену зависи од капацитета меморијског модула. На пример, уколико су меморијски модули капацитета $4k \times 8$, они захтевају 12 адресних улаза пошто је $4096 = 2^{12}$. Тако, адресне линије A_0 - A_{11} треба да буду повезане са RAM меморијским модулима.

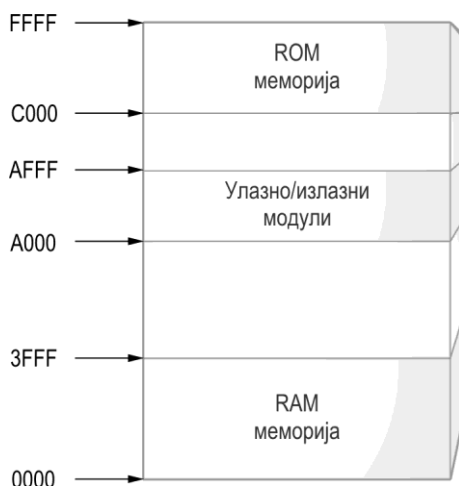
4. Структура и начин рада микрорачунара

На основу досадашње анализе види се да се у процесу декодовања адреса, адресне линије веће тежине користе да би се изабрао један меморијски модул, а адресне линије мање тежине да би се селектовала жељена локација из изабраног модула.

4.9 КОМПЛЕТАН ПРИМЕР ДЕКОДОВАЊА У МИКРОРАЧУНАРИМА

У овом одељку биће описана комплетна адресна декодујућа логика за микрорачунар реализован са процесором MC68HC11, чија је меморијска мапа дата на слици 4.16. Пошто постоји више интегрисаних кола за декодовање, за задату меморијску мапу постоји неколико могућих начина да се реализује логика декодовања.

Одабрану реализацију често користе пројектанти микрорачунарских система. Меморијска мапа на слици 4.16 показује да је дно адресног простора додељено RAM меморији. Овај принцип примењује се у свим системима који користе MC68HC11 микроконтролер, пошто његов инструкцијски скуп обезбеђује програмеру ефикасан начин да податке који се мењају, смести на нулту страницу меморије. Да би програмер то



4.16 Меморијска мапа за микрорачунарске системе са микроконтролером MC68HC11

искористио, страница \$00 (нула) треба да буде додељена RAM меморији. Врх адресног простора додељен је ROM меморији. Ово је карактеристика MC68HC11 система, пошто MC68HC11 захтева да неки важни подаци морају да буду стално смештени на адресама \$FFF8-\$FFFF.

Мапа показује да су адресе \$8000-\$8FFF додељене улазно/излазним склоповима. Избор улазно/излазних адреса обично се заснива на различитим критеријумима као што су једноставна логика декодовања, додељивање простора за проширење RAM и ROM меморијског простора, итд...

4.9.1 Логика за декодовање RAM меморије

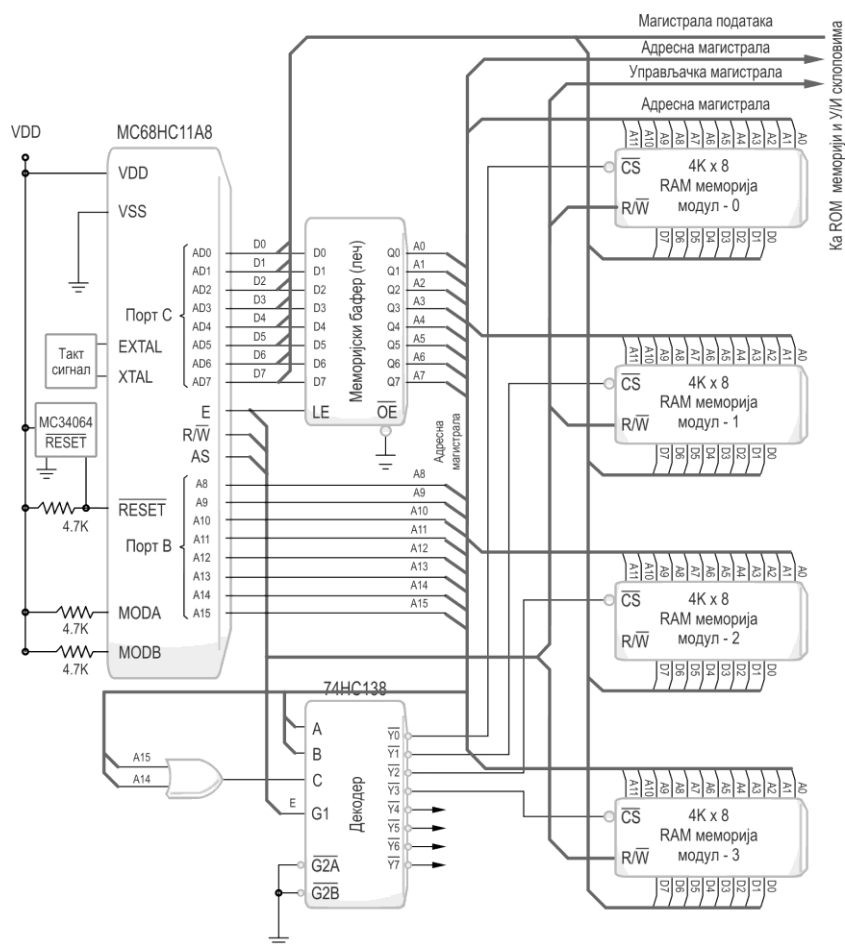
RAM меморија заузима адресе од \$0000 до \$3FFF, укупно 16384 адреса. Тако систем користи RAM меморијски капацитет од 16kX8. Користићемо четири 4kX8 RAM модула, чије је повезивање на магистралу процесора приказано на слици 4.17.

Као што смо већ нагласили сваки модул може бити комбинација од неколико RAM меморијских интегрисаних кола као што је приказано на слици 4.14.

Посматрајмо слику 4.17 и уочимо следеће:

- Сваки модул има своје улазно/излазне линије за податке, везане на магистралу података (порт C), али ће само изабрани (селектовани) модул поставити податке на ову магистралу. Линија процесора $R/\overline{W}$ везана је за сваки модул. Само ће селектовани модул одговорити на управљачке сигнале.
- Када је контролни адресни импулс (AS) на напонском нивоу логичке јединице биће селектован 74HC373 бафер. Нижи бајт адресе (A_0-A_7) са порта C биће запамћен у баферу када AS импулс пређе на напонски ниво логичке нуле.
- 74HC138 је декодер 1-од-8. Генерише сигнал $\overline{Y0}, \overline{Y1}, \overline{Y2}$ и $\overline{Y3}$ који селекује 4k меморијске модуле 0, 1, и 3, респективно. Ови сигнали су излазни сигнали декодера чији су улази адресне линије $A_{12}-A_{15}$ и Е-такт сигнал.
- Сваки модул има своје адресне улазе везане за линије A_0-A_{11} адресне магистрале. Ове адресне линије селекују специфичну локацију у оквиру селектованог модула.

4. Структура и начин рада микрорачунара



Слика 4.17 Логика за декодовање RAM меморије

Сваки модул садржи 4096 8-битних речи. Одредићемо адресе података који се смештају у сваки од меморијских модула и услов под којим се активира улаз за селектовање меморијског модула.

Да бисмо ово илустровали модул 3 биће селектован сваки пут кад је $\overline{Y3}$ на напонском нивоу логичке нуле. То се дешава када је декодерски улазни кôд: C=0, B=1, A=1 и када је његов улаз за селектовање (G1) на напонском нивоу логичке јединице. Декодерски улаз C биће на напонском

нивоу логичке нуле једино када су ИЛИ улази A_{15} , A_{14} такође на напонском нивоу логичке нуле. Улаз В биће на напонском нивоу логичке јединице (висок ново) када је адресна линија A_{13} на високом нивоу а улаз А биће на високом нивоу када је A_{12} на високом нивоу. Тако ће модул 3 бити селектован када је следећа адреса постављена на адресну магистралу:

	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
Селектовање модула 3 →	0	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X

Ознака „X“ испод A_0 - A_{11} указује да ове адресне линије не користи декодер за селектовање модула 3. A_{11} - A_0 може да буде било каква комбинација од 0000000000_2 до 1111111111_2 , зависно од тога којој речи треба приступити у модулу 3. Тако се комплетан опсег адреса за модул 3 одређује користећи све 0, а затим све 1 уместо ознаке „X“.

	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
$3000_{16} =$	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
$3FFF_{16} =$	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Видимо да је опсег хексадецималних адреса смештених у модулу 3 \$3000-\$3FFF. Када процесор поставља било коју адресу из овог опсега на адресну магистралу, модул 3 биће селектован или за операцију читавања или за операцију уписивања, у зависности од стања линије $R/\overline{W}$.

Сличном анализом одређује се опсег адреса и за остале RAM меморијске модуле и добија се:

Модул 0: \$0000-\$0FFF

Модул 1: \$1000-\$1FFF

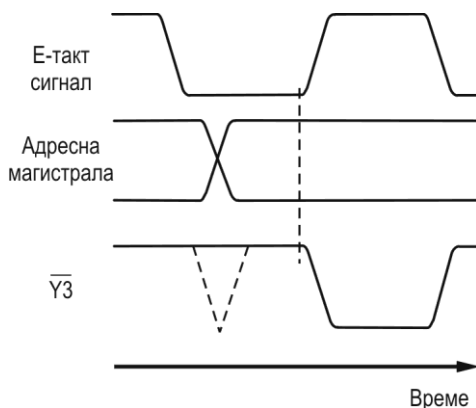
Модул 2: \$2000-\$2FFF

Модул 3: \$3000-\$3FFF

Веза Е-такт сигнала и декодера

Улаз декодера за селектовање ($G1$), који је активан када је на напонском нивоу логичке јединице, повезан је са Е-такт сигналом. Овим је обезбеђено селектовање декодера само за време дела такт циклуса када је $E=1$, тј. када је стабилна адреса на адресној магистрали. На овај начин

4. Структура и начин рада микрорачунара



Слика 4.18 Излаз декодера (за селектовање RAM3 меморијског модула) активан је само за време док је E=1

спречава се појава нежељених импулса (гличева¹) на декодерском излазу за време транзиција адресних линија док је сигнал E=0. Ови гличеви, уколико се појаве, могу да селекују RAM меморију у погрешном тренутку и проузрокују уписивање погрешног податка.

Слика 4.18 показује временски дијаграм сигнала за време уписа у RAM (WRITE операција) $\overline{Y3}$ са било које адресе у опсегу \$0C00-\$0FFF. Уочимо да $\overline{Y3}$ пада на низак ниво када је E-такт сигнал на високом напонском нивоу, без обзира што процесор поставља адресу за модул 3 тек за време AS сигнала. Уколико би декодер био активан за време промене на адресној магистрала испрекидана линија на слици за $\overline{Y3}$ сигнал указује где би могао да се појави глич и доведе до нежељеног уписа у меморију на некој случајној адреси. Због тога је неопходно у декодерску логику увести и E-такт сигнал који спречава појаву оваквих грешака које се тешко откривају зато што се дешавају стохастички.

Пример 4.10 *Шта ће се догодити RAM меморијским модулима на адресама које су више од \$3FFF?*

Решење:

За адресе које су више од \$3FFF једна или више адресних линија

¹ glitch - глич

биће висока. Ово ће произвести напонски ниво логичке јединице на С улазу декодера. Са $C=1$ декодерски излази $\overline{Y0}-\overline{Y3}$ не могу да буду на напонском нивоу логичке нуле. То значи да ниједан од RAM меморијских модула није селектован. Ово је оно што смо желели да постигнемо, пошто више адресе користе ROM меморијски модули и улазно/излазни склопови.

4.9.2 Непотпуно декодовање

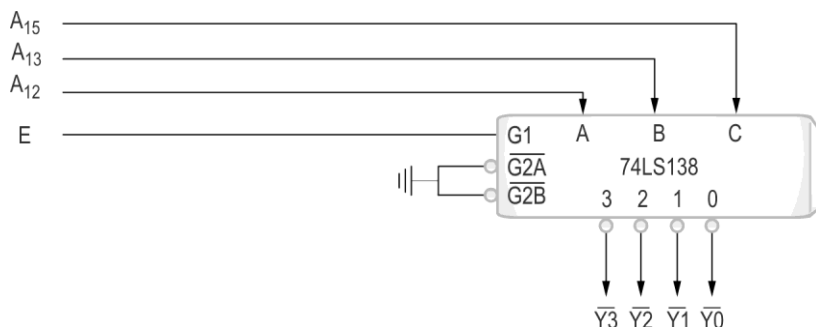
Да би селектовала меморијску локацију декодујућа шема на слици 4.17 користи све адресне линије. Овај принцип (техника) назива се потпуно декодовање. Примењује се у системима где се свака адреса користи (или ће бити искоришћена) за RAM меморије, ROM меморије или улазно/излазне модуле. Да би се минимизирао број декодујућих кола, а за микрорачунаре који немају потребе да користе комплетан адресни простор, пројектант може да користи принцип непотпуног декодовања. Код непотпуног декодовања не користе се све адресне линије.

Искористићемо шему непотпуног декодовања за микрорачунар на слици 4.13 пошто његова меморијска мапа (слика 4.12) показује да се не користи сав адресни простор. На пример, ИЛИ коло са два улаза може се уклонити и заменити директном везом A_{15} на декодер (његов С улаз), као што је приказано на слици 4.15. Са овом променом не користи се адресна линија A_{14} , пошто нема никаквог утицаја на декодер. Да би се селектовао RAM модул 3, тј. генерисао низак сигнал на $\overline{Y3}$, потребни су следећи услови:

	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
Селектовање модула 3 →	0	X	1	1	X	X	X	X	X	X	X	X	X	X	X	X

Адресна линија A_{14} сада нема утицаја тако да она може да буде на било ком логичком нивоу. Тако, постоје 2 комбинације које се могу појавити на овим адресним линијама, односно исти модул налази се на два места у меморијској мапи. За сваку од комбинација и за различите вредности $A_{11} - A_0$ добијају се други адресни опсези. На пример, за $A_{14}=0$ опсег би био \$3000-\$3FFF; а за $A_{14}=1$ добија се опсег \$7000-\$7FFF. Процесор може да користи, за приступ модулу 3, било који од ових адресних опсега пошто се

4. Структура и начин рада микрорачунара



Слика 4.19 Измена шеме са слике 4.17 да би се постигло непотпуно декодовање

ниједан не преклапа са адресама додељеним ROM меморији или улазно/излазним модулима. Цена коју смо платили за упрошћавање хардвера је губитак корисног меморијског простора. У простор од 32kB можемо да сместимо највише 16kB RAM меморијског простора.

Као што се на слици 4.19 види, непотпуно декодовање смањује декодујућу логику, а када се приступа истом модулу производи више од једног адресног опсега (ехо RAM меморије). У наведеном случају то није никакав проблем пошто ниједан од ових опсега не укључује адресе које су додељене другој меморији или улазно/излазном модулу.

Пример 4.11 Да ли може на слици 4.19, уместо адресне линије A_{15} , да се употреби адресна линија A_{14} ?

Решење:

Не, пошто се тада ехо RAM меморије појављује у подручју које је предвиђено за ROM меморију.

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X

Селектовање модула 3 налази се у адресном подручју \$3000-\$3FFF.

Ехо подручје

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
1	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X

Селектовање модула 3 налази се у адресном подручју \$B000-\$BFFF

4.9.3 Логика за декодовање ROM меморије

Сада је на реду додавање ROM меморије микрорачунару који анализирамо. Погледајмо поново меморијску мапу са слике 4.16. Видимо да ROM меморија заузима адресе од \$C000-\$FFFF. Ово је укупни простор од 16384 8-битних речи. Да би се то обезбедило потребан је ROM меморијског капацитета од 16kX8. Употребићемо четири меморијска интегрисана кола 27C32 UVEPROM капацитета 4kX8. На слици 4.20 приказано је на који начин су ови меморијски модули везани за магистрале процесора. Претпоставићемо да су линије адреса, података и управљачке линије исте као на слици 4.18.

Посматрајући слику 4.20 уочавамо следеће:

1. Сваки ROM меморијски модул има излазе везане за магистралу података, али само одабрани (селектовани) ROM поставиће податке на магистралу.
2. Сваки ROM меморијски модул има два управљачка улаза који су активни кад су на напонском нивоу логичке нуле. То су сигнал за селектовање интегрисаног кола ($\overline{CE}$) и сигнал за дозволу излаза ($\overline{OE}$). Улаз се користи да активира интегрисано коло. Када је $\overline{CE}$ на напонском нивоу логичке јединице ROM меморија је у стању „искљученог напајања”, и троши веома мало енергије из напонског извора +5V. Када је $\overline{CE}$ на напонском нивоу логичке нуле ROM меморија је у стању „укљученог напајања”, и реаговаће на адресне и $\overline{OE}$ улазе. Када је $\overline{CE}=0$, $\overline{OE}$ треба да буде на напонском нивоу логичке нуле да би селектовао излазне бафере ROM меморијског модула. Тако и $\overline{CE}$ и $\overline{OE}$ треба да буду на напонском нивоу логичке нуле да би обезбедили да се подаци појаве на излазу ROM меморијског модула.
3. Линија $R/\overline{W}$ процесора инвертована је и везана за $\overline{OE}$ улаз сваког од ROM меморијског модула. За време операције читавања (READ) напонски ниво логичке јединице на $R/\overline{W}$ линији активира улаз ROM меморијског модула на чијем $\overline{CE}$ улазу је напонски ниво логичке нуле.
4. Декодерско коло 74HC138 генерише $\overline{CE}$ сигнале. Ови сигнали су означени са $\overline{Cxxx}$, $\overline{Dxxx}$, $\overline{Exxx}$ и $\overline{Fxxx}$ што указује на опсег адреса који

4. Структура и начин рада микрорачунара

је селектован. На пример $\overline{C_{xxx}}$ сигнал селектује ROM меморијски модул чији је адресни опсег адреса \$C000-\$CFFF. Декодер генерише ове сигнале на бази адресних линија A_{15} - A_{12} и Е-такт сигнала.

- Адресни улаз у сваки ROM меморијски модул јесу адресне линије A_{11} - A_0 . Ове адресне линије селектују одређену реч податка унутар селектованог ROM меморијског модула.

У сваки ROM меморијски модул смештено је 4096 8-битних речи. Можемо да одредимо адресни опсег за сваки ROM меморијски модул одређивањем услова који активирају $\overline{CE}$ улаз. Да бисмо ово илустровали посматрајмо ROM-2 меморијски модул. Биће селектован када је декодерски излаз $\overline{E_{xxx}}$ на напонском нивоу логичке нуле. Ово ће се десити само када су декодерски улази у следећим стањима: $A_{15}=1$, $A_{14}=1$, $A_{13}=1$, $A_{12}=0$, $E=1$. Последњи услови користе се из истог разлога који је и раније описан. Меморијски модул ROM-2 биће селектован када је адреса:

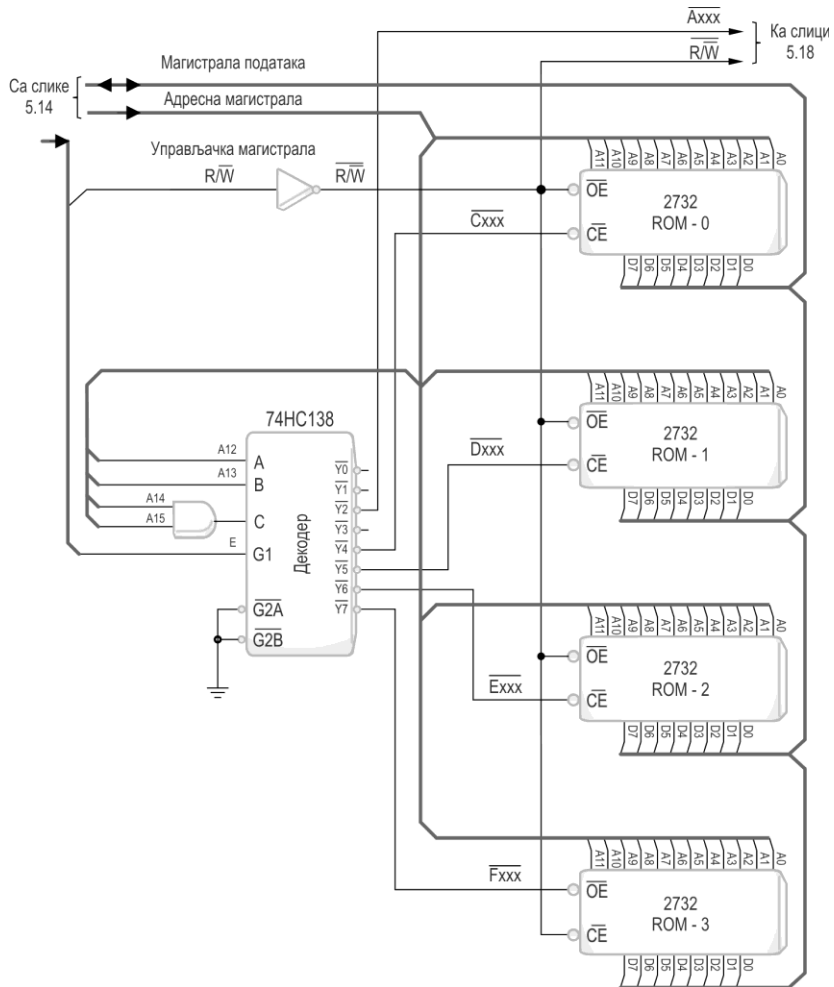
$$\overline{E_{xxx}}_{16} = \begin{matrix} A_{15} & A_{14} & A_{13} & A_{12} & A_{11} & A_{10} & A_9 & A_8 & A_7 & A_6 & A_5 & A_4 & A_3 & A_2 & A_1 & A_0 \\ 1 & 1 & 1 & 0 & X & X & X & X & X & X & X & X & X & X & X & X \end{matrix}$$

Знак „X” означава да адресна линија нема утицаја, пошто за селектовање ROM-2 меморијског модула декодер не користи адресне линије A_{11} - A_0 . Да би селектовао интерне меморијске локације ROM-2 меморијски модул користи адресне линије A_{11} - A_0 . Због тога је \$E000-\$EFFF потпуни опсег хексадецималних адреса за ROM-2 меморијски модул.

Сличном анализом могу се установити адресни опсези за сваки ROM меморијски модул. Резултат је:

ROM-0: \$C000-\$CFFF
 ROM-1: \$D000-\$DFFF
 ROM-2: \$E000-\$EFFF
 ROM-3: \$F000-\$FFFF

Пример 4.12 Да ли може још један ROM меморијски модул да се веже на декодерско коло приказано на слици 4.20, са улазом везаним за декодерски улаз најмање тежине?



Слика 4.20 Логику за декодовање ROM меморијских модула

Решење:

Не, овај декодерски излаз је $\overline{A_{xxx}}$ и реагује на адресни опсег \$A000-\$AFFF. Овај адресни опсег већ је додељен улазно/излазним модулима и не може се користити за ROM меморијски модул јер би дошло до преклапања адреса регистра периферије и ROM меморије.

Пример 4.13 У ROM меморију не може се уписивати. Описати шта се дешава када процесор, због неке програмске грешке, извршава инструкцију: STAA\$D776.

Решење:

Када процесор постави адресу \$D776 на адресну магистралу, декодерски излаз $\overline{Dxxx}$ постаће низак да би селектовао ROM-1 модул. Пошто извршава операцију уписивања процесор ће поставити $R/\overline{W}=0$. Овим се поставља висок ниво на $\overline{OE}$ улаз да би онемогућио излазе ROM меморије. Тако, ROM-1 меморијски модул не може да постави своје податке на магистралу података, а самим тим они се не могу сукобити са подацима које поставља процесор. Наравно, подаци процесора не могу се уписати ни у један ROM меморијски модул.

4.9.4 Улазно/излазна декодерска логика

Последњи део декодерских кола користи се да би се селектовали улазни или излазни модули који учествују у преносу података са процесором. Меморијска мапа за микрорачунарски систем који анализирамо (представљен на слици 4.16) указује да су адресе \$A000-\$AFFF додељене улазно/излазним модулима. Ово је опсег од 4096 адреса, али то не значи да ће микрорачунар имати толико повезаних улазно/излазних модула. У ствари, микрорачунар обично има три до шест улазно/излазних модула. Неки микрорачунари имају само два комуникациона модула, што представља апсолутни минимум за комуникацију микрорачунара са спољним светом. Додељивање толиког броја адреса за улазно/излазне склопове урађено је да би се искористило непотпуно декодовање. Наравно, на тај начин се губи доста адресног простора, и не користи се када је потребан максималан RAM и ROM меморијски капацитет.

У нашем примеру микрорачунара употребили смо два једноставна улазно/излазна модула, које називамо портови. Слика 4.21 показује декодујућу логику за прикључивање на магистралу процесора једног улазног и једног излазног порта који треба да обезбеде читавање тастатуре 4x4. На основу дијаграма уочавамо следеће:

1. Улазни порт састоји се од 74HC244 интегрисаног кола, које садржи осам бафера са три стања. Излази бафера везују се на

магистралу микрорачунара. Улази бафера служе као стварни улазни портови преко којих могу да се доведу спољашњи подаци. Излази бафера селектиовани су када се два $\overline{OE}$ улаза држе на напонском нивоу логичке нуле. Овим се на магистралу података постављају спољашњи подаци, тако да их процесор може читати.

2. Излазни порт састоји се од 74НС377 интегрисаног кола које садржи осам ивицом окиданих D флип-флопова са заједничким улазним сигналом за селектовање ($\overline{IE}$) и заједничким такт улазом (CP). Осам флип-флопова формира 8-битни излазни регистар. D улази везани су на магистралу података тако да подаци са магистрале могу да се упишу у регистар кад дође до позитивне транзиције на CP улазу и када је $\overline{IE}=0$. Q излази служе као стварни излазни порт преко кога се подаци из процесора могу послати ка спољашњим модулима.
3. Декодујућа логика састоји се од два НИ кола и два инвертора. Улази у ову логику долазе од сигнала који се генеришу у логици за декодовање ROM меморијских модула (слика 4.20). Једина адресна линија која се користи у улазно/излазној декодујућој логици је A_2 . Ова линија селекује улазни или излазни порт.

Очитавање са улазног порта

Када процесор извршава наредбу којом се жели очитавање података са улазног порта, мора да генерише одговарајуће логичке сигнале да би се добио низак ниво на излазу кола НИ2. Ово захтева висок ниво на сваком од НИ2 улаза па према томе имамо:

- $R/\overline{W}=1$, што указује на операцију READ;
- $\overline{A_{xx}}=0$, што значи да су нивои на четири адресне линије вишег реда 1, 0, 1, и 0 респективно;
- $A_2=1$.

Значи, процесор ће читавати податке са улазног порта кад год обавља READ операцију са било које адресе која је облика:

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
1	0	1	0	X	X	X	X	X	X	X	X	X	1	X	X

4. Структура и начин рада микрорачунара

Јасно је да, пошто многе адресне линије немају утицаја, постоји много различитих хексадецималних адреса које ће селектовати улазни порт. Због једноставности треба поставити сваки „X” на 0. Овим добијамо адресу $1010\ 0100_2 = A004_{16}$, коју ћемо сматрати адресом улазног порта.

Пример 4.14 *Која ће асемблерска инструкција податке, који се налазе на улазном порт, пренети у акумулатор?*

Решење:

LDAА \$A004, која се преводи у тробајтни машински кођ: \$B6, \$A0, \$04. Подсетимо се да је \$B6 операциони кођ за LDAА операцију, и да је \$A004 адреса операнда.

Упис на излазни порт

Када процесор треба да упише податке у излазни регистар, два услова морају бити испуњена: излаз HI1 кола треба да буде низак, и Е-такт сигнал треба да има позитивну транзицију. Низак излаз на HI1 јавиће се када је сваки улаз висок, тако да је:

- $R/\overline{W}=0$ за операцију WRITE ,
- $\overline{Axxx}=0$ и
- $A_2=0$.

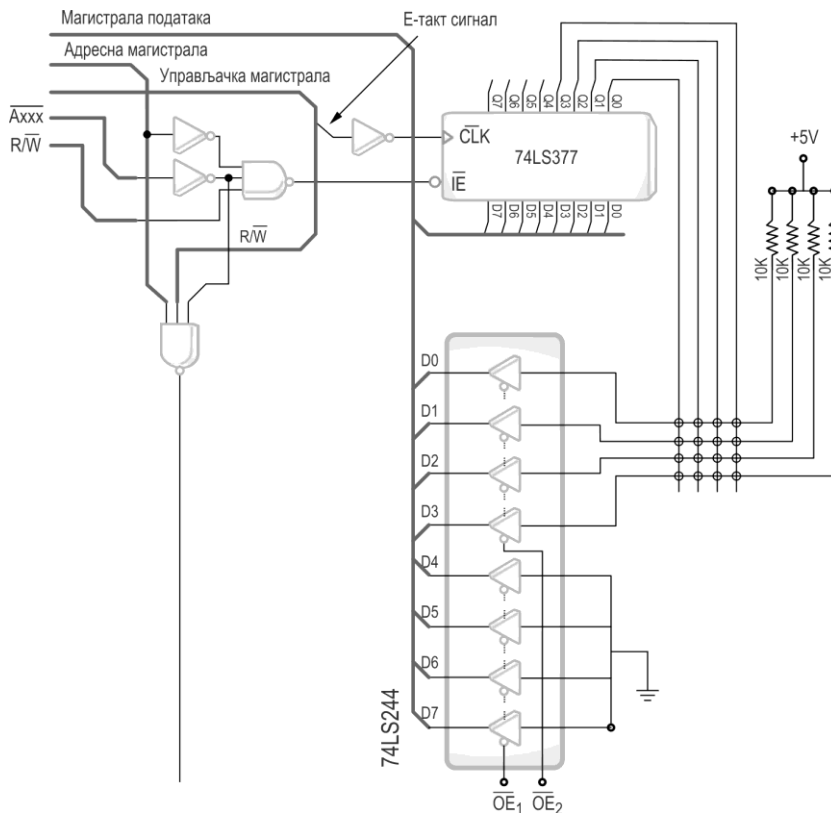
Тако процесор може да пошаље податке ка излазном порт у извршавањем операције WRITE на било коју адресу која је облика:

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
1	0	1	0	X	X	X	X	X	X	X	X	X	0	X	X

Још једном, због једноставности, поставићемо свако X на 0. На тај начин добијамо адресу $10100000_2 = A000_{16}$, коју ћемо сматрати излазном адресом.

Када процесор уписује податке у излазни регистар подаци ће бити у излазном порт у до следећег уписа.

На слици 4.21. приказана је шема повезивања сканирајуће тастатуре са процесором коришћењем једноставних логичких кола (бафер са три стања и лач). Ова два логичка кола морају да се налазе на различитим адресама у меморијској мапи. Да би се то остварило простор од 4096 локација који селекује излаз адресног декодера ($Axxx$) треба поделити на подопсеге,



Слика 4.21 Веза процесора са тастатуром

То се постиже увођењем нижих адресних линија у декодерску логику. Претпоставимо да леч треба да се налази на адреси \$A000 а бафер са три стања на адреси \$A004. Адресна линија која дефинише адресе 0 или 4 је \$A2, а подручје од \$A000 до \$AFFF дефинише линије са адресног декодера. Значи потребно је реализовати додатну декодерску логику чији ће улазни сигнали бити \$A2 и селект линија са адресног декодера. У примеру који анализирамо у декодерску логику уведена је и $R/\overline{W}$ линија. Улога $R/\overline{W}$ линије у декодерској логици је да спречи упис у бафер са три стања и читање са леча. Посебно је важно забранити упис у бафер са три стања јер би у том случају дошло до конфликта

4. Структура и начин рада микрорачунара

на магистралу података. Наиме, процесор би на магистралу података избацио податак а бафер са три стања на исту магистралу избацио би садржај са својих улаза, односно били би повезани излази бафера са три стања на излазе бафера процесорске магистрале података, што није дозвољено. Додатна декодерска логика реализована је са два троулазна НИ кола. Начин рада овим типом тастатуре детаљно је описан у 8. поглављу.

Пример 4.15 *Која ће асемблерска инструкција податке из акумулатора пребацити на излазни порт?*

Решење:

STA \$A000, која се преводи у тробајтни машински код: \$B7, \$A0, \$00. Подсетимо се да је \$B7 оперциони код за STA, и да је \$A000 адреса операнда.

4.10 ПОВЕЗИВАЊЕ МАГИСТРАЛА ПРОЦЕСОРА

Кола са слика 4.17, 4.20 и 4.21 формирају комплетан микрорачунар. Процесори, микроконтролери, RAM, ROM и VLSI интегрисана кола генерално се праве у CMOS технологији. Декодери, логичка кола, инвертори, бафери са три стања и излазни регистри су у LSTTL или HC технологији¹, да би минимизирали захтеве за потрошњу и оптерећење.

Уколико детаљније погледамо шеме ових кола уочићемо да се линије адресне магистрале, магистрале података и управљачке магистрале везују на неколико модула (оптерећења).

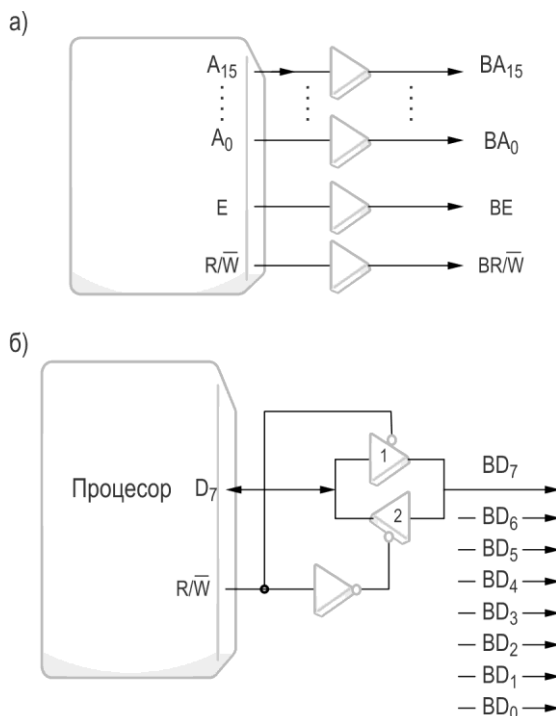
На пример, адресна линија A_{15} везана је за улаз ИЛИ кола и декодера. Адресна линија A_2 везана је за четири RAM меморијска модула, четири ROM меморијска модула, инвертор и НИ коло.

Свака линија магистрале података везана је за четири RAM меморијска модула, четири ROM меморијска модула, улазни порт и излазни порт.

Управљачки сигнал $R/\overline{W}$ повезан је на четири RAM меморијска модула, инвертор и НИ коло. Пошто се процесор прави у CMOS технологији, његове

¹ HC (High speed CMOS) модул пин/пин компатибилан са LSTTL (Low power Schottky TTL).

излазне линије немају довољан струјни капацитет за напајање више улаза других модула. Обично не могу да напајају више од једног стандардног TTL улаза без деградације логичких нивоа, и не могу да напајају капацитивна оптерећења, везана за магистралу података, без значајне деградације брзине промене логичких нивоа сигнала. Капацитивности магистрала су присутне као последица већег броја модула који су везани на магистралу података. Сваки прикључени модул повећава капацитивно оптерећење за 5 до 10pF. У многим, добро пројектованим системима, магистрала процесора се за друге модуле везује преко кола која се називају бафери¹, линијски драјвери² или бас драјвери³, што је илустровано на слици 4.22.



Слика 4.22 а) баферована адресна и управљачка магистрала;
б) двосмерни бафери магистрале података

¹ Buffers

² Line drivers

³ Bus drivers

4. Структура и начин рада микрорачунара

Слика 4.22а показује да су адресна и управљачка магистрала процесора повезане за неинвертујуће бафере. Излази бафера имају довољан струјни капацитет да могу да напајају 5 до 10 TTL кола. Бафери могу да подрже и капацитивно оптерећење које се јавља на линијама адресне магистрале које су везане на неколико RAM и ROM модула. 74LS244 чип се обично користи за ову намену. Садржи осам бафера у чипу. Ови бафери су бафери са три стања и обично су повезани тако да су стално селектовани.

Пошто баферски излази имају исте логичке нивое као и улази, излази су обично означени као улази, осим додатног слова „B”. На пример, излаз A_0 бафера означен је са BA_0 и назива се баферовани A_0 сигнал. Линије магистрале података процесора обично су бафероване. Двосмерне линије захтевају комплекснији начин повезивања (баферовања). Слика 4.22б показује како линије података могу да буду бафероване коришћењем бафера са три стања. $R/\overline{W}$ управљачки сигнали одређују који бафер је селектован. За време WRITE операције низак ниво на $R/\overline{W}$ селекује бафер 1 да дозволи да се подаци са D_7 појаве на магистрали података. За операцију READ висок ниво на $R/\overline{W}$ селекује бафер 2 да дозволи подацима са магистрале података да се појаве на улазу процесора. Уочимо да је само један бафер активан у датом тренутку. Исти принцип и начин повезивања примењени су и за остале линије података.

Постоји више интегрисаних кола, као што је 74LS245, која садрже парове бафера повезане, што је приказано на слици 4.22б. Ова интегрисана кола обично се називају двосмерни бафери или трансивери¹.

4.11 МЕМОРИЈСКИ МАПИРАНИ И ИЗОЛОВАНИ УЛАЗНО/ИЗЛАЗНИ МОДУЛИ

Многи процесори, као што је процесор микроконтролера MC68HC11, комуницирају са улазно/излазним модулима и меморијом на исти начин. Улазно/излазни модул адресира се на исти начин као и меморија. Процесор користи $R/\overline{W}$ сигнал да би одредио смер преноса

¹ Трансивери *bidirectional buffers, transceivers*

података. Улазно/излазни модул и меморија деле меморијски простор, с тим што су неке адресе резервисане за улазно/излазне модуле, а друге за меморију. Процесор користи исте инструкције да би комуницирао са улазно/излазним модулима као и са RAM и ROM меморијама. Другим речима када улазно/излазни модули имају додељене адресе у меморијској мапи процесор не прави никакву разлику између меморије и улазно/излазних модула. Овај метод за приступ улазно/излазним модулима назива се меморијски мапирани улазно/излазни модули.

Као што је напоменуто раније, два основна начина на који процесор може да приступа улазно/излазним модулима јесу меморијско мапирани и изоловани улазно/излазни модули. Претходним анализама били су обухваћени само системи са меморијски мапираним улазно/излазним модулима, где су улазно/излазни модули третирали као било која меморијска локација. Значи адресе су као меморија и процесор користи R/W сигнал да би одредио смер преноса података.

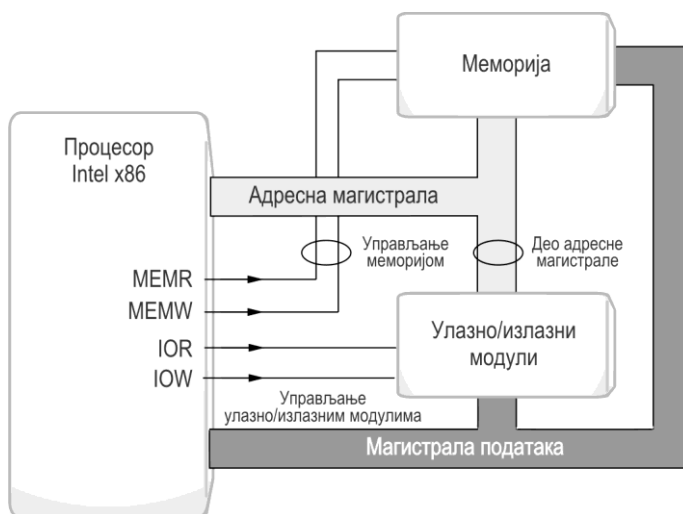
Неки процесори, као што је фамилија Intel x86 пројектовани су за изоловане улазно/излазне модуле где процесор третира меморију и улазно/излазне модуле независно, користећи различите управљачке сигнале за њих. Слика 4.23 показује како процесор Intel x86 комуницира са изолованим улазно/излазним модулима. Процесор Intel x86:

1. користи четири управљачка сигнала који одређују која операција се извршава: MEMR, MEMW, IOR и IOW;
2. MEMR је активан за време циклуса читавања меморије;
3. MEMW је активан за време циклуса уписивања у меморију;
4. IOR је активан за време циклуса читавања улазно/излазних модула;
5. IOW је активан за време циклуса уписивања у улазно/излазни модул;

На пример код IBM персоналног рачунара постоје 32 адресне линије од којих се само њих 11 користи за приступ улазно/излазним модулима. Ограничавање система само на 2048 улазно/излазних адреса је одлука пројектаната. Уместо коришћења регуларних инструкција за приступ меморији Intel x86 рачунари користе специјалне *in* и *out* инструкције за приступ улазно/излазним модулима.

Пошто се стање линије разликује зависно од тога да ли се ради о

4. Структура и начин рада микрорачунара



Слика 4.23 Изоловани улазно/излазни уређај који се примењује код процесора фамилије Intel x86

меморији или улазно/излазним модулима, меморија може да користи 65536 могућих адреса. Ово је предност за изоловани улазно/излазни модул над меморијски мапираним улазно/излазним модулима. Недостатак рада са изолованим улазно/излазним модулима је да се захтевају посебне инструкције за извођење операција са улазно/излазним модулима. Меморијски мапирани улазно/излазни модули и меморија користе исти скуп инструкција. Ово доприноси већој флексибилности у програмирању.

4.12 ПИТАЊА И ЗАДАЦИ

1. Описати функције следећих управљачких линија управљачке магистрале: $\overline{XIRQ}$, $\overline{IRQ}$, $R/\overline{W}$, AS , $\overline{RESET}$, Е-такт сигнал, $PH2$.
2. Каква је разлика између $\overline{XIRQ}$ и $\overline{IRQ}$ сигнала прекида?
3. Каква је разлика између Е-такт сигнала и $PH2$ сигнала?
4. Шта је то улазно/излазни порт?
5. Која од операција је READ а која WRITE:

- а) операциони код инструкције преноси се из меморије у инструкцијски регистар
 - б) садржај акумулатора А преноси се у RAM меморију
 - в) нижи бајт адресе операнда преноси се из програмског дела меморије у регистар адресе податка
6. Колико је операција READ, а колико WRITE када се извршава инструкција STAA \$7010?
 7. Одређени микрорачунар расподелио је меморијски простор на следећи начин: адресни простор од 0000 до \$4FFF за RAM меморију; адресни простор од \$6000 до \$6FFF за улазно/излазне модуле; адресни простор од \$D000 до \$FFFF за ROM меморију.
 - а) нацртати меморијску мапу овог микрорачунара
 - б) одредити капацитет RAM меморије
 - в) одредити колико се улазно/излазних модула може повезати са овим микрорачунаром
 - г) одредити капацитет ROM меморије
 8. На слици 4.20 употребљена су интегрисана кола ROM меморије капацитета 4kx8. Модификовати електричну шему тако да се користе интегрисана кола ROM меморије капацитета 4kx8. Адресни простор додељен ROM меморији је непромењен (\$C000 - \$FFFF).
 9. Зашто је често потребно „баферовати” адресу, податке и управљачке сигнале микрорачунара?
 10. Описати основне разлике између меморијски мапираних и изолованих улазно/излазних модула.
 11. Набројати све карактеристике IRQ сигнала прекида.
 12. Какву улогу има $R/\overline{W}$ сигнал?
 13. На шта указује висок ниво AS сигнала?
 14. Који податак се налази на магистрали података у другом циклусу извршавања инструкције STAA \$B8B9? Операциони код инструкције је \$B7.
 15. Чиме је одређена величина меморије којој процесор MC68HC11 може да приступи?
 16. Која врста адресирања се користи у инструкцији LDAA \$13?
 17. Описати како се генерише Е-такт сигнал код микроконтролера MC68HC11.

4. Структура и начин рада микрорачунара

18. Шта се дешава са такт сигнаlima микроконтролера MC68HC11 када се он налази у режиму „STOP“?
19. Каква је функција такт сигнала?
20. Набројати и описати намену такт сигнала микроконтролера MC68HC11.
21. Како микроконтролер MC68HC11 комуницира са улазно/излазним склоповима?
22. Које операције процесора се означавају као читавање (READ)?
23. Које операције процесора се означавају као уписивање (WRITE)?
24. Који сигнали учествују у операцији читавање (READ) процесора MC68HC11?
25. Нацртати и објаснити временску повезаност између различитих сигнала код операције читавања процесора MC68HC11.
26. Нацртати и објаснити временску повезаност између различитих сигнала код операције уписивања процесора MC68HC11.
27. Нацртати и објаснити активности на магистралама процесора MC68HC11 за време извршавања инструкције LDAA \$1234?
28. Шта је то меморијска мапа?
29. Нацртати и објаснити један пример меморијске мапе за микрорачунаре са микроконтролером MC68HC11 у проширеном начину рада.
30. Објаснити разлику између меморијски мапираних и изолованих улазно/излазних модула.

5. Процесор

Иако се разликују у архитектури, микрорачунари и микроконтролери имају један заједнички елеменат, а то је процесор. У суштини, процесор је централни део микрорачунара због тога што могућности процесора одређују могућности микрорачунара. Његова брзина одређује максималну брзину микрорачунара, његови прикључци¹ за адресе и за податке одређују капацитет меморије и величину речи података, а управљачки прикључци одређују начин повезивања са меморијом и периферијама.

Процесор извршава велики број операција, укључујући:

1. Обезбеђивање временских и управљачких сигнала за све елементе микрорачунара;
2. Узимање инструкција и података из меморије;
3. Пренос података ка меморији и из меморије;
4. Декодовање инструкција;
5. Извршавање аритметичких и логичких операција које инструкције захтевају;
6. Реаговање на управљачке сигнале као што су поновно укључивање $\overline{\text{RESET}}$ и прекид ($\overline{\text{IRQ}}$) које генеришу улазно/излазни склопови.

Процесор садржи сва потребна логичка кола за обављање ових операција, али велики део интерне логики није споља доступан. На пример, не може се на интегрисано коло процесора довести спољашњи сигнал који ће инкрементирати програмски бројач. Уместо тога, деловима процесора може се софтверски приступити. Значи, може се утицати на кола само програмом који се смешта у меморију и који процесор извршава. Ово чини процесоре разноликим и флексибилним - када се жели променити њихов начин рада једноставно се мења програм

¹ Pins-пинови

5. Процесор

(на пример променом ROM меморије у којој је смештен програм). Генерално, једноставније је мењати програм него реконфигурисати хардвер.

У овом поглављу анализираћемо рад проширене поједностављене верзије процесора MC68HC11 и у анализу укључити већину унутрашњих делова и спољашњих управљачких сигнала.

5.1 ПРОЦЕСОР MC68HC11

Слика 5.1 је целовит блок дијаграм интерне архитектуре процесора MC68HC11. Уколико упоредимо овај дијаграм са поједностављеном верзијом коју смо у претходним поглављима разматрали, проналазимо следеће разлике:

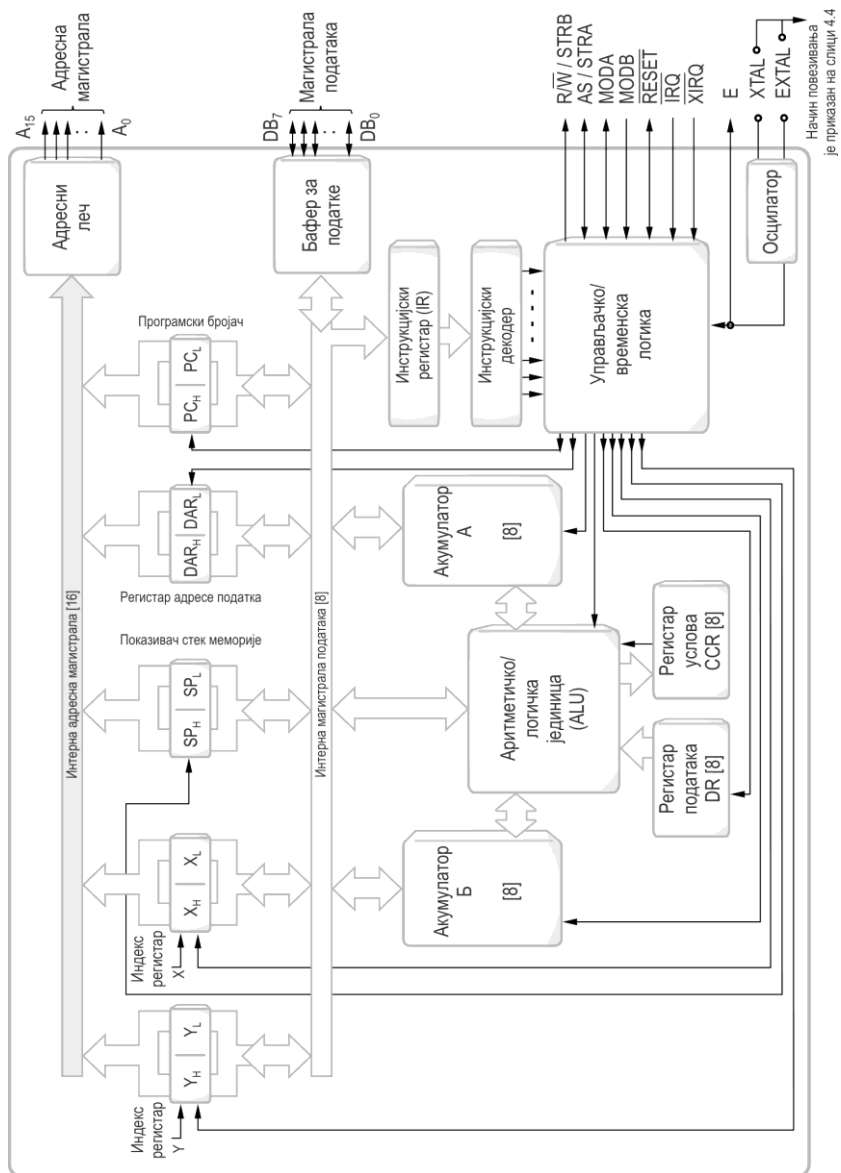
1. Постоји пет 16-битних регистара везаних на интерну адресну магистралу. Два индекс регистра (X, Y) и регистар - показивач стек меморије (SP¹) додати су уз програмски бројач (PC) и регистар адресе податка (DAR). Сваки од ових регистара подељен је на два дела.
2. Постоје два 8-битна акумулатора (A, B), везана на аритметичко/логичку јединицу и интерну магистралу. Акумулатори A и B могу се користити као посебни 8-битни регистри или као један 16-битни регистар, који се назива двоструки акумулатор (D).
3. Постоји 8-битни регистар услова (CCR²) повезан са аритметичко/логичком јединицом.
4. Приказани су сигнали управљачке магистрале које користи MC68HC11 у проширеном начину рада. Приказана су и екстерна кола која су потребна за генерисање системског Е-такт сигнала који је потребан процесору MC68HC11.

5.2 УПРАВЉАЧКО/ВРЕМЕНСКА ЛОГИКА

Управљачко/временском логиком нећемо се много бавити из два разлога. Прво, то је област процесора над којим немамо детаљан увид.

¹ Stack Pointer

² Condition Code Register



Начин повезивања
је приказан на слици 4.4

5.1 Блок дијаграм процесора MC68HC11

Друго, да би се развили кориснички програми није потребно знати детаљну структуру управљачко/временског дела.

Главна функција управљачко/временске логице је да из програмске меморије узима и декодује инструкције и генерише неопходне управљачке сигнале, који су потребни аритметичко/логичкој јединици и регистрима за извршавање инструкција.

5.2.1 Сигнали управљачке магистрале

Управљачки део генерише и спољашње сигнале који се шаљу другим деловима микрорачунара, као део управљачке магистрале система. О неким од ових управљачких сигнала нпр. $R/\overline{W}$, AS било је већ речи у 4. поглављу.

Поред генерисања излазних сигнала управљачке магистрале, управљачки део процесора реагује на улазне сигнале управљачке магистрале које шаљу други делови микрорачунара ка процесору. Примери улазних управљачких сигнала¹ су $\overline{RESET}$ и $\overline{IRQ}$.

Сваки процесор има свој јединствен скуп улазних и излазних управљачких сигнала, који су детаљно описани у произвођачким упутствима. Дефинисаћемо управљачке сигнале процесора MC68HC11.

AS сигнал

Овај излазни сигнал назива се контролни адресни импулс² и процесор MC68HC11 га користи када ради у нормалном проширеном начину рада³. Нижи бајт адресе уписује се у спољашњи 8-битни регистар када је AS сигнал на напонском нивоу логичке јединице, а преноси на адресну магистралу када је AS сигнал на напонском нивоу логичке нуле.

Поред улаза за прекид ($\overline{XIRQ}$ и $\overline{IRQ}$) још пет других улаза (прикључака) интегрисаног кола MC68HC11 (тачније MC68HC11A8) може да генерише захтев за прекид процесора. Прикључци су повезани (тј. додељени) са периферијама које се налазе у оквиру интегрисаног кола микроконтролера, као што је часовник (тајмер) или други улазно/излазни системи који

¹ *Interrupt Request*

² *Address Strobe*

³ *Normal expanded mode*

захтевају размену¹ управљачких сигнала. Прикључци су PA0/IC3, PA1/IC2, PA2/IC1, PA7/PAI²/OC1, и AS/STRA и могу се конфигурисати тако да детектују растућу или опадајућу ивицу улазног сигнала. STRA улаз постоји само када микроконтролер ради као једно интегрисано коло³, а у проширеном начину рада прикључак има улогу контролног адресног импулса (AS). Ових пет улаза имају предност у односу на XIRQ и IRQ пошто сваки од њих може независно да се маскира локалним управљачким битом и заједничким (глобалним) I битом у регистру услова (CCR). Сваки од ових пет прекида има индикацију свог стауса, која се може прочитати и захтев се може обрисати без да је био обрађен.

Нормални, проширени начин рада може се одабрати постављањем логичке јединице на оба прикључка MODB и MODA за време ресетовања. Овај начин рада омогућава да се приступи спољној меморији и периферијским уређајима преко временски мултиплексираних магистрала адреса/подаци.

Мултиплексирањем 8 бита мање тежине адресе са подацима на прикључцима порта C потребно је само 18 прикључака да би се обезбедила 8-битна магистрала података, 16-битна адресна магистрала и две управљачке линије.

Адресне линије мање тежине одвојене су од података са спољним, транспарентним лечом (као што је на пример 74HC373, на који се као такт сигнал доводи AS сигнал⁴).

Е-такт сигнал

Процесор MC68HC11 користи један јединствен такт сигнал и он се означава као Е-такт сигнал. Користи се интерно и екстерно, а генерисан је у самом интегрисаном колу. Учестаност Е-такт сигнала зависи од екстерног такт сигнала који се прикључује на улазе EXTAL и XTAL (слика 4.4).

¹ Handshake

² Импулсни акумулатор (*pulse accumulator*) је 8-битни бројач/часовник који се може конфигурисати на различите начине. У једном од начина вредност 8-битног бројача увећава се са сваком активном ивицом на PAI (*Pulse Accumulator Input*) прикључку.

³ Single-chip mode

⁴ Користи се (без обзира да ли је екстерни или интерни) учестаност Е-такт сигнала. Максимална учестаност магистрале за микроконтролер MC68HC11A8 је 2,1MHz што се може упоредити са најбржом спољном EPROM меморијом која је била доступна у време када се на тржишту појавио микроконтролер MC68HC11.

Поновно укључивање - ресет ($\overline{\text{RESET}}$)

Ресет и прекид често се анализирају заједно пошто користе исти принцип рада: са унапред дефинисане локације у меморији (ресет вектор¹ односно вектор прекида²) учитава се садржај у програмски бројач и на тај начин „принуди“ процесор да започне са извршавањем програма почевши од те адресе.

Три интерна извора и екстерни сигнали могу да изазову ресет микроконтролера MC68HC11. Било који извор (интерни или екстерни) поставља напонски ниво логичке нуле на $\overline{\text{RESET}}$ прикључак. Једина разлика између узрока ресета је њихов ресет вектор (табела 5.1) који се користи да укаже процесору на почетну адресу за извршавање програма.

Интерни узроци поновног укључивања су:

1. Основни ресет вектор користи се када је узрок интерни ресет после укључења напона напајања (POR³) или ако је спољња логика поставила напонски ниво логичке нуле на $\overline{\text{RESET}}$ прикључак.
2. Уколико осцилатор ради превише споро, или се заустави, коло за надгледање такт сигнала генерише ресет (уколико је та опција селектована).
3. Истицање времена на које је постављен интерни часовник генерише $\overline{\text{RESET}}$ сигнал. Реч је о колима за самонадгледање⁴ у интегрисаном колу које представља заштиту од системских грешака. Систем који обезбеђује⁵ правилан рад (COP⁶) рачунара је заштита од софтверских грешака. На пример кола која су задужена за откривање непостојећих операционих кодова изазивају немаскирајући прекид.

Када је ресет детектован интерни регистри и контролни битови постављају се у иницијална стања. Иницијална стања, с друге стране, управљају периферијама у интегрисаном колу и постављају их у дефинисана, почетна стања. После ресета, процесор у току прва три

¹ Reset vector

² Interrupt vector

³ Power-on reset

⁴ Self-monitoring

⁵ Watchdog system

⁶ Computer Operating Properly

Узрок ресета	Вектор у нормалном начину рада	Вектор код специјалног тестирања и <i>bootstrap</i>
POR или RESET	\$FFFE, \$FFFF	\$BFFE, \$BFFF
Clock Monitor Fail	\$FFFC, \$FFFD	\$BFFC, \$BFFD
COP Watchdog Time-Out	\$FFFA, \$FFFB	\$BFFA, \$BFFB

Табела 5.1 Ресет вектори и начини рада микроконтролера

цикла са читава ресет вектор са локација \$FFFE, \$FFFF (односно \$BFFE, \$BFFF у специјалном случају тестирања, или када је у режиму примања програма преко серијског порта) и почиње извршавање програма. Показивач стек меморије и други регистри процесора немају, у том тренутку, дефинисан садржај али X и I бит (регистра услова) за маскирање прекида постављају се тако да маскирају било који захтев за прекидом. Такође, S бит регистра услова (CCR) поставља се да забрани STOP врсту рада.

Линија R/W (STRB) је излазна управљачка линија која има две функције у зависности од начина рада. Када је микроконтролер MC68HC11 подешен за проширени начин рада ова управљачка линија обавештава екстерне модуле (меморију, улазно/излазне склопове) да ли се извршава операција читавања ($R/\overline{W}=1$) или уписивања ($R/\overline{W}=0$). Када микроконтролер MC68HC11 ради као једно интегрисано коло онда је то управљачка линија (STRB) која омогућава размену управљачких сигнала.

Маскирајући захтев за прекид ($\overline{IRQ}$)

Овај улаз микроконтролера користе улазно/излазни модули да би прекинули извршавање текућег програма и проузроковали да процесор почне да извршава специјалан програм за обраду (опслуживање) прекида¹. Адреса програма читава се у програмски бројач из прекидног вектора, који се налази у ROM меморији, на локацијама \$FFF2, \$FFF3. По завршетку програма за опслуживање прекида, процесор наставља са извршавањем програма на ком је радио када се прекид појавио. Улаз $\overline{IRQ}$ може се програмски онемогућити², тако да процесор неће на њега реаговати.

¹ Interrupt service routine

² Disabled

Немаскирајући захтев за прекид ($\overline{XIRQ}$)

Ово је други прекидни улаз који се разликује од $\overline{IRQ}$ по томе што се не може онемогућити у току рада. Само за време извршавања ресета X бит у регистру услова (CCR) поставља се и било који захтев за прекид биће маскиран док их софтвер не омогући. Прекидни вектор за $\overline{XIRQ}$ налази се у ROM меморији на адресама \$FFF4, \$FFF5.

5.3 РЕГИСТРИ

Уобичајена операција која се извршава унутар процесора јесте пренос бинарних информација из једног у други регистар. Број и тип регистара које процесор садржи кључни је део његове архитектуре и има значајног утицаја на програмирање одређене апликације. Структура регистара процесора разликује се значајно од произвођача до произвођача. Основна функција коју обављају различити регистри суштински је код свих иста. Користе се за смештање података, адреса, инструкцијских кодова и информација о статусу различитих операција процесора. У овом поглављу биће описани регистри процесора микроконтролера MC68HC11.

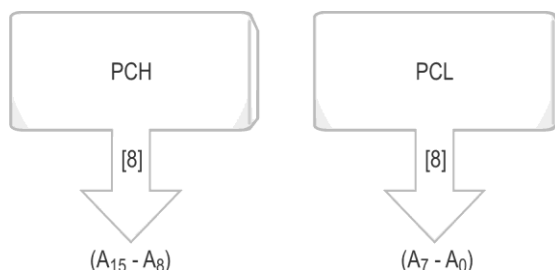
5.3.1 Инструкцијски регистар

Када процесор читава (узима) инструкцијску реч из меморије, он је уписује (шаље) у инструкцијски регистар. Ту је смештена док кола за декодовање не одреде која инструкција треба да се изврши. Процесор користи инструкцијски регистар у сваком циклусу инструкције, и програмер нема разлога, а ни могућности, да му приступа. Величина инструкцијског регистра је иста као и величина речи података. Микроконтролер MC68HC11 је 8-битни па је и инструкцијски регистар 8-битни.

5.3.2 Програмски бројач

Програмски бројач (PC) је 16-битни регистар у коме се увек налази адреса следеће инструкцијске речи у меморији коју процесор треба да прочита. Када је RESET улаз микроконтролера MC68HC11 активан, у програмски бројач се уписује адреса прве инструкције коју треба извршити.

Величина програмског бројача зависи од броја адресних битова са којима процесор ради. Код микроконтролера MC68HC11 процесор користи 16-битну адресу, па је програмски бројач 16-битни. Програмски бројач (и



Слика 5.2 Програмски бројач, 16- битни, раздвојен у два 8-битна дела (други адресни регистри) подељен је у два 8-битна регистра PCN и PCL, тако да сваки од њих садржи пола адресе.

Разлог за употребу мањих регистра је тај што је често потребно сместити садржај програмског бројача у меморију. Пошто се у меморију смештају 8-битне речи, 16-битни програмски бројач се морао раздвојити на два дела, и његов садржај се смешта у две сукцесивне меморијске локације.

Не постоје инструкције којима програмер може да мења садржај програмског бројача. То значи да не постоје инструкције којима програмер може садржај меморијских локација учитати и уписати у програмски бројач, нити инструкције којима може садржај програмског бројача уписати у неке меморијске локације. Када се извршавају инструкције гранања (BEQ) или безусловног скока (JMP) програмски бројач добија вредности ван нормалног редоследа (инкрементирања).

5.3.3 Регистар адресе података

Регистар адресе података (DAR¹) садржи адресу на коју се уписује или са које се чита податак у делу меморије за податке. На пример, када се извршава LDAA \$0201 инструкција, процесор смешта адресни део (\$0201) инструкције у регистар адресе података. Садржај регистра адресе података поставља се на адресну магистралу тако да процесор може да учита податак са те адресе².

¹ Data Address Register

² Детаљно је објашњено у 3. поглављу овог уџбеника.

5.3.4 Акумулатор

Акумулатор је регистар који учествује у највећем броју операција које извршава аритметичко логичка јединица. Он је, такође, регистар у који се смешта резултат многих операција које се извршавају у аритметичко логичкој јединици. Процесор MC68HC11 располаже са два 8-битна акумулатора А и В што убрзава извођење програма, јер дозвољава да два операнда буду истовремено у процесору. Као што је већ поменуто, ова два акумулатора могу се користити као један 16-битни акумулатор D.

Акумулатори А и В јесу регистри опште намене и користе се за држање операнда и резултата аритметичких операција и рада са подацима.

Највећи број операција може да користи и акумулатор А и акумулатор В. Постоји одређени број изузетака. Инструкције ABX и ABY додају садржај акумулатора В на садржаје индекса регистара X или Y, и не постоје еквивалентне инструкције за акумулатор А.

Инструкције TAP и TPA користе се за пребацивање садржаја из акумулатора А у регистар услова, или из регистра услова у акумулатор А. Еквивалентне инструкције за акумулатор В не постоје. Инструкција за децимално прилагођење акумулатора А (DAA), која се користи после аритметичких операција за децимално кодиране бројеве, не постоји и за акумулатор В. На крају, инструкције за додавање, одузимање и упоређивање садржаја акумулатора А и В (ABA, SBA и CBA) могу да раде само у једном смеру. Због тога је важно испланирати унапред да се одговарајући операнди налазе у одговарајућим акумулаторима.

5.3.5 Индекс регистри

Индекс регистри X и Y (16-битни) користе се за индексно адресирање. Код овог начина адресирања садржају 16-битног регистра додаје се 8-битно одступање које је део инструкције. На тај начин добија се ефективна адреса операнда. У многим случајевима инструкције које укључују индекс регистар Y могу да користе додатни бајт за објектни код и додатни циклус за време извршавања, у поређењу са еквивалентном инструкцијом са регистром X. Други индекс регистар је нарочито користан када се операнди налазе у две различите табеле које се користе за израчунавање.

У ранијим верзијама (M6800), програмер је морао да смешта индекс у неке привремене локације одакле се индекс учитавао у индекс регистар.

Инструкције ABX и ABY заједно са инструкцијама инкрементирања и декрементирања дозвољавају неке аритметичке операције над индекс регистрима. Инструкције за размену XGDY и XGDY, нуде једноставан начин да се напуни вредност индекса у 16-битни двоструки акумулатор D, који омогућава сложеније аритметичке операције од самих индекс регистара.

Уобичајено је да се напуни индекс регистар почетном адресом интерног меморијског простора (обично \$1000), који дозвољава да се индексно адресирање користи за приступ интерним улазно/излазним и управљачким (контролним) регистрима.

Још један важан аргумент за коришћење индексног адресирања је могућност рада (манипулације) на нивоу битова

Процесор MC68HC11 располаже са два 16-битна индекс регистра: X и Y. Као било који регистри опште намене и они се могу употребити и за смештање података и као бројачи. Још једно битно обележје овог регистра је да је могуће његово коришћење у програмима који користе табеле или низове података. Тада индекс регистар дефинише адресу податка коме се приступа.

Ова техника се назива индексно адресирање и посебна је врста адресирања коју програмер може да користи. Постоје различите врсте индексног адресирања. У овом поглављу анализираћемо индексно адресирање које користи процесор MC68HC11.

Основна идеја индексног адресирања је да се стварна, односно ефективна адреса, на коју се односи инструкција добија као сума одступања¹ и садржаја индекс регистра. Да бисмо ово илустровали, претпоставимо да је индекс регистар величине 16 битова и да је његов садржај \$0111. Претпоставимо да процесор извршава инструкцију пуњења акумулатора A (LDAA) користећи индексно адресирање. Симболички код за ову инструкцију биће LDAA Z, X. Величина одступања (означена са Z) је број који се додаје садржају индекс регистра и тако се добија адреса операнда. Код процесора MC68HC11 одступање је увек величине једног бајта. Инструкцију индексног адресирања можемо приказати на следећи начин:

¹ Offset - одступање, померај

5. Процесор

Адреса меморијске речи	Меморијска реч
A100	A6 ← кôд за LDAA који користи индексно адресирање
A201	0A ← одступање

$$\begin{aligned}\text{ефективна адреса} &= \text{одступање} + [X] \\ &= \$0A + \$0111 \\ &= \$011B\end{aligned}$$

Ефективна адреса је \$011B, тако да је то адреса са које се читава податак који се смешта у акумулатор А.

Овај пример показује да стварна адреса са које се подаци узимају зависи од текућег садржаја индекса регистра Х. Уколико се садржај Х регистра промени и инструкцијска секвенца понови, биће употребљена нова адреса. Ово обележје чини индексно адресирање посебно корисним за рад са табелама података које су смештене у меморији, што се може видети у примеру 5.1:

Пример 5.1 Табела од седам података смештена је на меморијске локације \$2061 до \$2067. Нацртати дијаграм тока за програм који ће сабрати седам података и сместити збир у меморијске локације на адресе \$2068 виши бајт и \$2069 нижи бајт. Користити индексно адресирање.

Решење:

Тражени дијаграм тока приказан је на слици 5.3. Испитати и уочити следеће важне тачке:

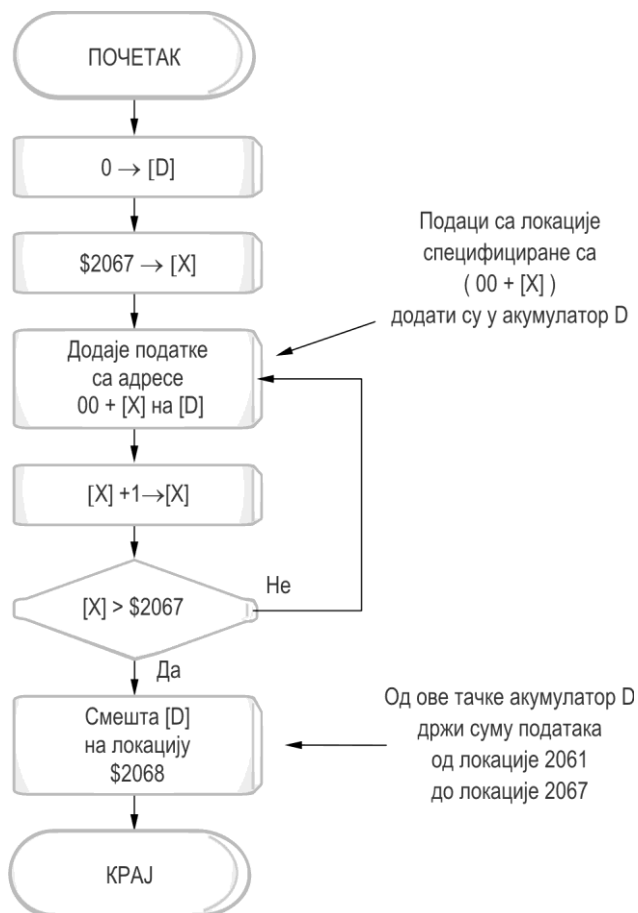
1. акумулатор D је обрисан (његов садржај је нула) пре него што се извршава инструкција ADD;
2. индекс регистар Х иницијално се пуни бројем \$2061;
3. инструкције у блоку 5-8 извршавају се укупно седам пута пре него што програм дође до блока 9;
4. ADD инструкција (блок 5) сваки пут се извршава узимањем података са друге локације, почевши од \$2061 и закључно са \$2067. На овај начин комплетна табела података додаје се на почетни садржај акумулатора D.

Асемблерски програм који одговара дијаграму тока на слици 5.3 је:

```

LDD  #$0
LDX  #$2061
PET  ADDD 0,X
INX
CPX  #$2068
BNE  PET
STD  $2068

```



Слика 5.3 Дијаграм тока програма за сабирање, индексним адресирањем, седам бројева у суседним меморијским локацијама

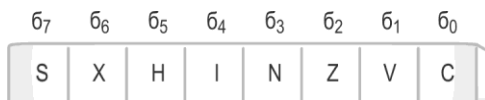
Регистар услова

Регистар услова садржи пет индикатора статуса, два бита за маскирање прекида и STOP бит¹. Регистар је добио име по пет битова услова (статуса) који су се највише користили у ранијим верзијама процесора (M6800 и M6801). У регистру нису постојали ни бит X за маскирање прекида ни бит STOP за заустављање рада.

Пет статусних ознака (битова) указују на резултате аритметичких и других операција процесора. То су:

- Бит преноса C (*Carry*);
- Бит премашења V (*oVerflow*);
- Нулти бит Z (*Zero*);
- Негативан бит N (*Negative*);
- Бит маскирања прекида I (*Interrupt Mask*);
- Бит половине преноса H (*Half carry*);
- Бит забране немаскирајућег прекида X (*Interrupt Mask*) и
- Бит који онемогућује STOP инструкцију S (*Stop Disable*).

Регистар услова (CCR²) често се назива и регистар статуса. Састоји се од битова са различитим значењем, које је дефинисао произвођач процесора. Ови битови називају се ознаке, заставице или флегови³. Сваки од битова користи се да укаже на резултат или део резултата претходно извршене операције процесора. Вредност појединих битова може се програмски испитивати. Дефинисаћемо и описати 8 битова који чине регистар услова за процесора MC68HC11.



Бит (флег) преноса C указује на статус аритметичких операција као што су сабирање и одузимање. На пример, ако процесор извршава инструкцију сабирања, и сабира две 8-битне речи, вредност преноса са места бита највеће тежине постаје вредност бита преноса C. Бит (C

¹ Disable bit

² Condition Code Register

³ Flag -заставица

флег) учествује у различитим инструкцијама које се односе на померања и ротацију и може се поставити на 1 или на 0, користећи инструкције SEC (*Set Carry*) и CLC (*Clear Carry*).

Уобичајено се С бит користи да укаже да је дошло до преноса код операције сабирања и да је дошло до позајмљивања код операције одузимања. Он се, такође, користи код операција множења и дељења. Операције померања и ротирања користе С бит да би потпомогле вишебајтне операције. Код процесора MC68HC11 садржај регистра услова аутоматски мењају (ажурирају) све инструкције. Тако, ретке су посебне инструкције чији је задатак само да мењају садржај регистра услова.

На пример инструкције напуни акумулатор А (LDAA), или смести акумулатор А (STAA) аутоматски постављају (на вредност 1) или бришу (поставља на вредност 0) N, Z, и V битове услова. То није случај са свим архитектурама, а код процесора Motorola ретке су инструкције које не мењају регистар услова. Најважније инструкције које не мењају садржај регистра услова су „гурање“¹, „повлачење“², додавање акумулатора В регистру X (ABX), додавање акумулатора В регистру Y (ABY), и 16-битни пренос и размена. Увек треба проверити које инструкције утичу, а које не на регистар услова.

Бит (флег) премашење V (*overflow*) користи се да укаже на премашење кад се бројеви са знаком сабирају или одузимају. Другим речима V се користи да укаже ако је премашење настало као резултат операција над бројевима у другом комплементу.

Да се подсетимо: бит највеће тежине се код бројева са знаком користи да означи да ако је 0 број је позитиван, ако је 1 број је негативан. На тај начин је само 7 битова остављено да одређују величину броја, па је опсег бројева који се може представити са 8 битова од -128_{10} до $+127_{10}$. Ако се код аритметичких операција као резултат добије број који је ван опсега онда се поставља бит премашења на вредност 1 да на то укаже. V бит се може програмски обрисати ($V=0$) или поставити ($V=1$) помоћу инструкција CLV и SEV.

Нулти бит (флег) Z (*Zero*) аутоматски се поставља на јединицу ($Z=1$) кад год је резултат аритметичких операција, рада са подацима или преноса

¹ *Pushes*

² *Pulls*

5. Процесор

података 8-битна реч која се састоји од свих нула. Z бит се аутоматски поставља на нулу ($Z=0$) за било који други резултат (различит од нуле).

Неколико операција (INX, DEX, INY, и DEY) утичу на Z бит, а не утичу на друге битове регистра услова. За ове операције корисник може да добије информацију само да ли је резултат операције једнак нули ($=$) или различит од нуле ($\neq$).

Негативан бит (флер) N (*Negative*) користи се да покаже знак било ког податка, резултата аритметичких операција, рада са подацима или преноса података. У ствари, N бит је увек једнак¹ 7. биту (бит знака) резултата код 8-битних операција и 15. биту код 16-битних операција.

Код бројева представљених у другом комплементу број је негативан када је бит највеће тежине постављен (MSB), а негативан када је обрисан.

Бит N се користи и у другим операцијама, а не само код бројева у другом комплементу. Бит регистра или меморијске локацији чија се вредност најчешће испитује поставља се као бит највеће тежине, а учитавањем у акумулатор његова вредност се пресликава у N бит.

Бит (флер) маскирања прекида I (*Interrupt mask*) општа је маска која дозвољава или забрањује прекид. Користи се да укаже да ли је утицај захтева за прекид са улаза $\overline{\text{IRQ}}$ дозвољен (омогућен²) или забрањен (ономогућен³). Када је $I=1$, утицај је ономогућен, када је $I=0$ утицај $\overline{\text{IRQ}}$ је дозвољен што значи да се $\overline{\text{IRQ}}$ улаз може користити да прекине регуларан рад процесора. Бит маскирања прекида може да се постави ($I=1$) или обрише ($I=0$) програмски помоћу SEI и CLI инструкције. Пошто је захтев за прекид „услужен“ и инструкција RTI извршена, враћа се садржај регистрима које су имали пре појаве прекида. Овај бит може и сам процесор да, по потреби у неким операцијама аутоматски поставља или брише, што ће бити анализирано у 6. поглављу.

Бит (флер) половине преноса H (*Half carry*) поставља се када је резултат операције сабирања пренос са позиције 3. бита на позицију 4. бита. Ово значи да ако се обавља BCD⁴ сабирање, резултат (збир)

¹ Треба имати на уму да је 1. бит b_0 , 7. бит b_6 , 15. бит b_{16} ...

² Enabled

³ Disabled

⁴ Binary Coded Digit

ће бити погрешан BCD код, и због тога га је потребно исправити. Када програмер користи сабирање BCD бројева, мора да дода инструкцију DAA (*Decimal Adjust Accumulator*- децимално прилагођење акумулатора). Ова инструкција указује процесору да провери вредност Н и С бита да би одредила корекциони фактор који треба додати резултату. Инструкција DAA ће садржај акумулатора А и С бита променити тако да представљају тачан бинарно кодирани децимални збир и тачно стање С бита.

Бит (флег) дозволе немаскирајућег прекида *X (Interrupt mask)* користи се да укаже да ли је утицај захтева за прекид са немаскирајућег улаза $\overline{\text{XIRQ}}$ микроконтролера MC68HC11 дозвољен (омогућен¹) или забрањен (ономогућен²). Када је $X=1$, утицај $\overline{\text{XIRQ}}$ је ономогућен, када је $X=0$ утицај $\overline{\text{XIRQ}}$ је дозвољен што значи да се $\overline{\text{XIRQ}}$ улаз може користити да измени регуларан ток извршења програма.

Не постоји инструкција којом се може вредност X бита поставити ($X=1$). Када се први пут укључи или после поновљеног укључивања (ресета) процесор MC68HC11 поставља аутоматски вредност X бита ($X=1$). После кратког времена постављају се иницијалне вредности интерних регистара процесора. Само тада се може обрисати вредност X бита ($X=0$) помоћу инструкције TAP (*Transfer Accumulator A to the Condition Code Register*). Једном када је $\overline{\text{XIRQ}}$ прекид дозвољен програмер га не може укинути.

Бит (флег) који ономогућује STOP инструкцију (*Stop Disable*) користи се да спречи извршавање STOP инструкције. Извршавање ове инструкције доводи до заустављања осцилатора и свих такт сигнала процесора MC68HC11.

Неки корисници сматрају да је употреба STOP инструкције опасна пошто доводи до заустављања осцилатора. Корисник може да употреби бит S у регистру услова (CCR) да дозволи или не дозволи STOP инструкцију. Уколико је STOP инструкцију процесор декодовао биће третирана као инструкција „нема операције“ (NOP³) и наставља се са извршавањем следеће инструкције.

¹ Enabled

² Disabled

³ No-operation

5.3.6 Условно гранање програма

Статусни битови N, Z, V, и C омогућавају гранање на основу резултата претходних операција. Могуће су различите врсте гранања (и бројева са знаком и без знака) под условима: $<$, $\leq$, $=$, $\neq$, $\geq$ или $>$. Пример уобичајеног коришћења битова укључује функције условног гранања које одређују инструкцију која треба да се изврши зависно од вредности битова регистра услова; на пример инструкција BEQ (*Branch If Equal to Zero* - гранај се ако је резултат операције нула). Када процесор извршава инструкцију BEQ, испитује вредност Z бита. Уколико је $Z=0$, показујући да је резултат извршења последње инструкције која је утицала на вредност Z бита различит од нуле, извршиће се следећа инструкција. Уколико је $Z=1$, што указује да је резултат последње инструкције нула, извршавање програма ће се наставити на новој адреси (програм се грана). Нова адреса (адреса гранања) добија се додавањем одступања, које је део инструкције гранања, на садржај програмског бројача.

5.3.7 Показивач стек меморије

Пре него што дефинишемо функције показивача стек меморије¹ потребно је да дефинишемо појам стека². Процесор MC68HC11 аутоматски подржава програмску стек (магацинску) меморију³. Она може да буде постављена било где у 64kB адресном простору и може да буде било које величине до величине меморијског простора који је доступан у систему.

Стек меморија је део RAM меморије резервисан за привремено смештање података, обично садржаја интерних регистара процесора.

Постоји неколико ситуација у којима се користи стек меморија. Процесор аутоматски памти повратну адресу и друге битне информације у стек меморији, за наставак извршавања главног програма. Програмер може да смести податке и информације на стек меморију у било које време користећи специјалне инструкције. На пример, садржај акумулатора A може да се смести на стек меморију коришћењем инструкције типа:

PSHA - упиши (*push*) садржај акумулатора A у стек меморију.

¹ *Stack Pointer*

² *Stack* - низ елемената постављених један иза другог.

³ *Stack*

Инструкција уписује садржај акумулатора А на локацију у стек меморији, која је дефинисана садржајем показивача стек меморије, а затим се декрементира садржај показивача стек меморије.

Супротна операција може да се извршава коришћењем инструкције:

PULA - прочитај (*pull*) податак са стек меморије и упиши у акумулатор А.

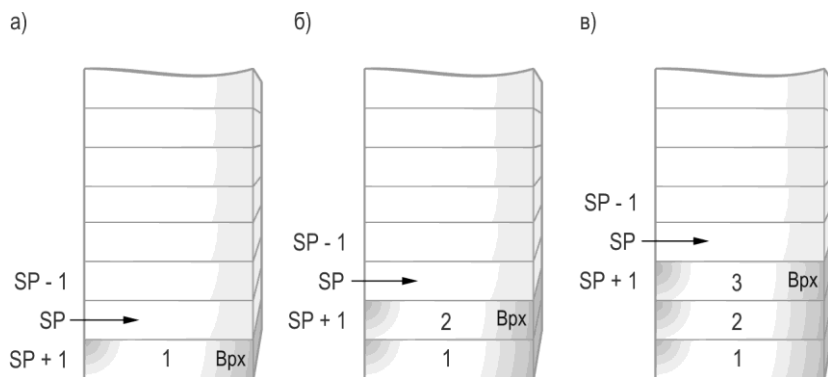
Инструкција инкрементира показивач стек меморије, а затим пуни акумулатора А податком са локације на коју показује показивач стек меморије. За операције упиши (PSHA) и прочитај (PSHA) програмер не треба да специфицира меморијску адресу. Адресу одређује показивач стек меморије. Програмер мора да води рачуна о иницијализацији показивача стек меморије и о томе да он не укаже на локације ван опсега који је резервисан за стек меморију. У супротном може се десити да се операција смештања на стек меморију обави преко дела меморије где су смештени програми или подаци. Код процесора MC68HC11 показивач стек меморије је 16-битни регистар, који дозвољава програмеру да стек меморију постави било где у адресном простору RAM меморије. Стек меморију дефинише програмер иницијализујући вредност показивача стек меморије помоћу инструкције LDS (*Load Stack Pointer* - напуни показивач стек меморије).

Рад са стек меморијом је такав да се податак који је последњи уписан први читава (LIFO¹ систем). Програм који следи уписује на стек меморију бројеве 1, 2, 3 тим редоследом (слика 5.4):

```
LDAA #1
PSHA      ; упиши број 1 на стек меморију
LDAA #2
PSHA      ; упиши број 2 на стек меморију
LDAA #3
PSHA      ; упиши број 3 на стек меморију
```

После извршења три инструкције уписивања („гурања“) на стек меморији ће се налазити три броја (1, 2, 3) са бројем 3 на врху стек меморије (слика 5.4в). Показивач стек меморије садржи адресу прве слободне локације која је за један мања од адресе последњег уписаног

¹ Last In First Out - последњи унутра први напоље.

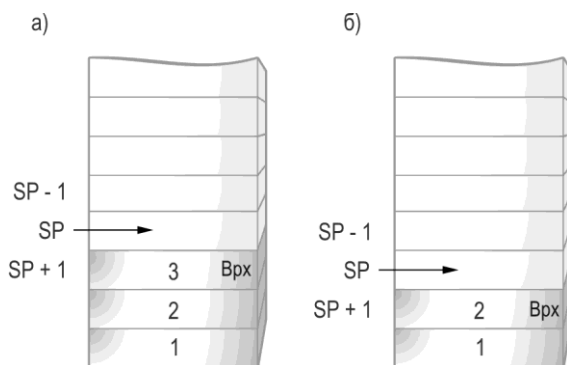


Слика 5.4 Уписивање података на стек меморију

броја¹. На слици 5.4 локације стек меморије у које су уписани бројеви су осенчене а неосенчене су слободне локације.

После извршавања инструкције PULA, број 3 ће бити враћен у акумулатор A, а број 2 ће бити на врху стек меморије (слика 5.5б).

Регистар – показивач стек меморије (SP) понаша се као адресни регистар за стек меморију. Кад је потребно податак сместити у стек меморију он се смешта на адресу коју садржи показивач стек меморије. Када податак треба да се прочита са стек меморије читава се са адресе која се налази



Слика 5.5 Подаци се читавају са стек меморије обрнутим редоследом од редоследа којим су уписани

¹ Код MC68HC12 показивач стек меморије садржи адресу на који је уписан последњи број.

у показивачу стек меморије која је увећана за један. Садржај показивача стек меморије иницијализује се после ресета. Показивач стек меморије аутоматски се декрементира по уписивању податка на стек меморију и инкрементира пре читавања податка са стек меморије. Инкрементирање и декрементирање аутоматски обавља управљачки део процесора.

Уобичајено, регистар показивач стек меморије иницијализује се неком од почетних инструкција у апликативном програму. Када се бајт упише у стек меморију показивач стек меморије се аутоматски декрементира. Читавање из стек меморије доводи до аутоматског инкрементирања показивача стек меморије.

У сваком тренутку показивач стек меморије садржи 16-битну адресу слободне локације на стек меморији. Стек меморија користи се у позивима потпрограма, прекидима и за привремено смештање података.

Када се позива потпрограм инструкцијама JSR¹ или BSR² адреса прве следеће инструкције иза инструкција JSR или BSR (текући садржај програмског бројача) аутоматски се смешта на стек меморију (прво нижи бајт). На крају потпрограма извршава се инструкција за повратак из потпрограма (RTS³). Инструкција RTS доводи до тога да се повратна адреса⁴ (сачувана у стек меморији) са стек меморије уписује у програмски бројач. Извршавање програма наставља се са те „повратне“ адресе.

Увек када се појави прекид (под претпоставком да није маскиран):

1. нормално се завршава инструкција чије је извршавање у току (текућа инструкција);
2. адреса следеће инструкције (текућа вредност програмског бројача) смешта се на стек меморију;
3. садржаји свих регистара процесора смештају се на стек меморију;
4. извршавање програма наставља се на адреси која је специфицирана адресним вектором највећег приоритета који је захтевао прекид.

¹ *Jump to subroutine*

² *BRanch to Subroutine*

³ *ReTurn from Subroutine*

⁴ Под повратном адресом подразумева се адреса са које би био узет инструкцијски код да се није „скочило“ на извршавање потпрограма. То значи да ће се извршавањем инструкције RTS у програмски бројач уписати (напунити) адреса инструкције која се налази после инструкције JSR (или BSR) .

По завршетку потпрограма за обраду прекида извршава се инструкција повратка из прекида (RTI¹). Инструкција RTI доводи до тога да се садржаји сачуваних регистара читавају са стек меморије обрнутим редоследом од оног којим су уписивани у стек меморију. Наставља се са извршавањем програма као да није ни био прекинут.

Други уобичајен пример коришћења стек меморије јесте привремено смештање садржаја регистара. Једноставан пример био би потпрограм који користи акумулатор А. Корисник ће сместити садржај акумулатора А на стек меморију када започиње извршавање потпрограма („улази у потпрограм“) и учитати га са стек меморије пре изласка из потпрограма. Овај метод је једноставан начин да се обезбеди да садржај регистара пре и после извршавања потпрограма буде исти.

Постоје и мање уобичајене примене стек меморије. На пример стек меморија се може користити за размену података између потпрограма који је уобичајен за преводиоце виших језика али га мање користе програмери на асемблеру.

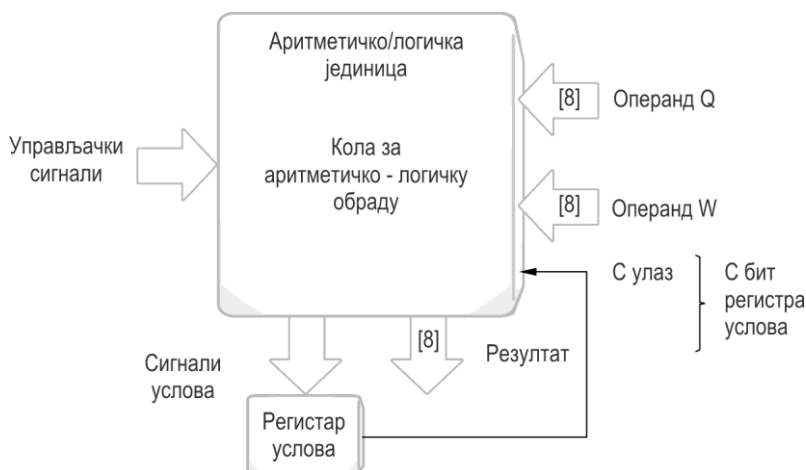
5.4 АРИТМЕТИЧКО/ЛОГИЧКА ЈЕДИНИЦА

Процесор садржи аритметичко/логичку јединицу која извршава велики број аритметичко/логичких операција. Ове операције могу да укључе два операнда као што су акумулатор А и податак из меморије, или акумулатор А и други интерни регистар. Неке операције укључују само један операнд. Поједностављен дијаграм аритметичко/логичке јединице приказан је на слици 5.6.

Аритметичко/логичку јединицу сачињавају сва логичка кола која се користе за аритметичке, логичке и друге операције које се извршавају са подацима. Ове операције могу се извршавати са два или једним операндом.

Два 8-битна операнда представљена су као улаз у аритметичко/логичку јединицу, без обзира што се често користи само један операнд. Улаз у аритметичко/логичку јединицу је и бит преноса С (из регистра услова), што ће детаљније бити објашњено у следећем поглављу. Операције које извршава аритметичко/логичка јединица одређене су управљачким сигнаlima из временско/управљачког дела процесора.

¹ *ReTurn from Interrupt*



Слика 5.6 Поједностављен дијаграм аритметичко/логичке јединице процесора MC68HC11

Аритметичко/логичка јединица има два излаза. Један је 8-битни излаз који представља резултат операције над операндима. Други излаз су статусни подаци који се уписују у регистар услова.

Улази, операнд Q и операнд W могу да дођу из неколико извора: акумулатора A и B, регистра података¹ (у који се учитава податак из меморије). Када аритметичко/логичка јединица извршава операције са два операнда један од њих је смештен у акумулатору, а други податак, који се узима из меморије, смешта се у регистар података. У неким операцијама други операнд може да буде садржај другог акумулатора. Резултат аритметичке операције која се обавља над два операнда, уписује се у акумулаторе. Ако су операнди садржаји акумулатора A и B резултат се увек уписује у акумулатор A.

Код операција са једним операндом он може да буде садржај акумулатора A, акумулатора B, регистра или податак из меморије. Резултат операције из аритметичко/логичке јединице шаље се назад, одакле је операнд узет. Када је операнд податак из меморије резултат се смешта прво у регистар података, а затим га процесор уписује у меморијску локацију са које је податак читан.

¹ Data buffer register

5.4.1 Операције са једним операндом

Неке од операција аритметичко/логичке јединице које се обављају над једним операндом су:

1. **Брисање**¹. Сви битови операнда се постављају на 0. Ако је операнд у акумулатору А, ову операцију можемо симболички да представимо са $0 \rightarrow A$.
2. **Комплементирање (инвертовање)**². Сви битови се мењају на супротан логички ниво. Уколико операнди долазе из акумулатора А, операција се представља са $\bar{A} \rightarrow A$.
3. **Инкрементирање**³. Операнд се увећава за 1. На пример, ако је операнд 11111010_2 у аритметичко/логичкој јединици се додаје 00000001_2 , и добија се резултат 11111011_2 . Ова операција омогућава да се регистри користе као бројачи. Симболично, представиће се као $[A]+1 \rightarrow A$.
4. **Декрементирање**⁴. Операнд је умањен за 1, тј. број 00000001_2 се одузима од операнда. Операција је погодна за програме у којима се регистри користе као бројачи уназад од неке иницијалне вредности. Симболички се може представити као $[A]-1 \rightarrow A$.
5. **Померање**⁵. Битови операнда померају се улево или удесно, на место „празног“ бита уписује се 0, а бит који је „избачен“ помера се у С бит (С флег) регистра услова. Анализираћемо операцију аритметичко померање улево⁶.

На слици 5.7 види се да је 0 уписана на место бита најмање тежине операнда а бит највеће тежине „избачен“ и померен (уписан) у С бит регистра услова (претходна вредност С бита је изгубљена).

На слици 5.8 приказана је операција логичког померања удесно⁷.

¹ Clear

² Complement - Invert

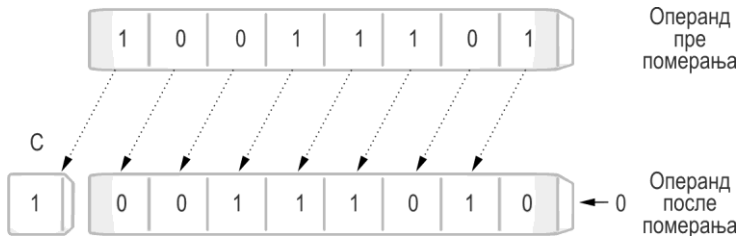
³ Increment

⁴ Decrement

⁵ Shift

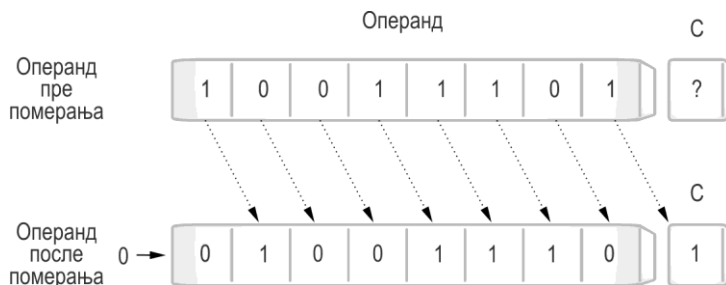
⁶ Arithmetic Shift Left

⁷ Logical Shift Right



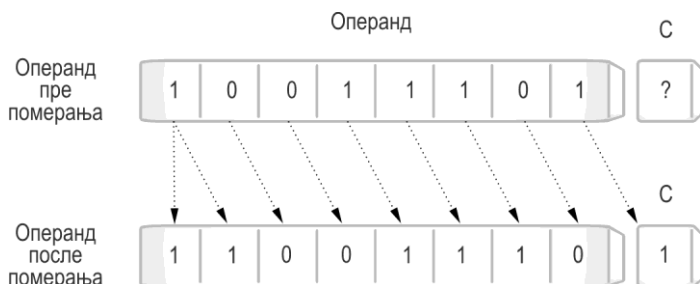
Слика 5.7 Операција аритметичког померања улево

Види се да је 0 померена у бит највеће тежине, а бит најмање тежине померен у C бит. Ову врсту операција користе програмери да би тестирали вредност одређеног бита операнда: врши се операција померања док се жељени бит не упише у C бит. Затим се користи инструкција условног гранања, која на основу вредности C бита одређује која се инструкција извршава као следећа. Постоји и операција аритметичко померање удесно¹. Резултат операције је померање сваког бита удесно, а бит најмање тежине помера се у C бит (слика 5.9). Бит највеће тежине помера се удесно и уписује на место бита највеће тежине. На тај начин сачуван је знак операнда: податак који је био позитиван (бит највеће тежине 0) остаје и даље позитиван, а податак који је био негативан (бит највеће тежине 1) остаје негативан. Резултат операције је еквивалентан дељењу операнда са два.



Слика 5.8 Операција логичког померања удесно

¹ Arithmetic Shift Right

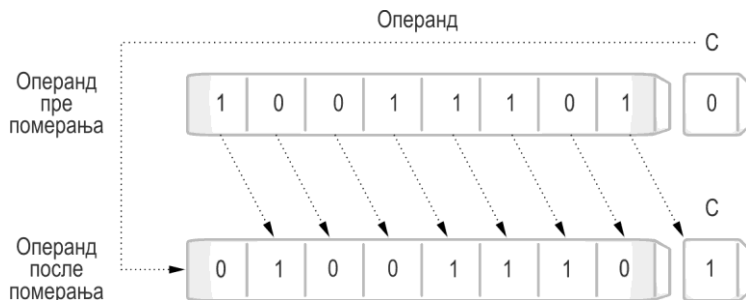


Слика 5.9 Операција аритметичког померања удесно

6. **Ротирање**¹. Ово је модификована операција померања у којој C бит постаје део циркуларног померајућег регистра заједно са операндом; тако се вредности избачене из операнда померају у C бит, а вредност из C бита помера се на испражњено место операнда.

Операција ротирања удесно² илустрована је на слици 5.10.

Уочимо да се C бит помера у бит највеће тежине (први бит са леве стране) операнда, а да се бит најмање тежине (последњи бит с десне стране) помера у C бит. Поступак ротирања улево је исти, само у супротном смеру.



Слика 5.10 Операција ротирања удесно

¹ Rotate

² Rotate Right

5.4.2 Аритметичко/логичке операције са два операнда

Склопови аритметичко/логичке јединице обављају операције сабирања и одузимања у бинарном бројном систему и увек третирају операнде као 8-битне бинарне бројеве. Анализираћемо неке аритметичко/логичке операција које се извршавају са два операнда.

1. **Сабирање**¹. Аритметичко/логичка јединица израчунава бинарни збир два операнда. У општем случају један операнд долази из једног од акумулатора, други из меморије, а резултат се шаље у акумулатор. Симболично ова операција се представља² са $[A_x] + [M] \rightarrow A_x$. Пошто MC68HC11 има два акумулатора (A и B), два операнда могу да буду из било ког од акумулатора. Резултат се шаље у акумулатор A, што се симболично може представити са $[A] + [B] \rightarrow A$. Ако је као резултат сабирање два операнда пренос бита највеће тежине, онда се поставља на 1 C бит преноса у регистру услова. У супротном C бит се поставља на 0. Другим речима C бит се користи као 9. бит резултата.
2. **Одузимање**³. Аритметичко/логичка јединица одузима један операнд (очитан из меморије) од другог операнда (из једног од акумулатора) и смешта резултат у одговарајући акумулатор. Симболично се може представити као $[A_x] - [M] \rightarrow A_x$. За операцију одузимања користи се принцип другог комплемента. Такође, може се садржај акумулатора B одузети од садржаја акумулатора A, као што је симболично приказано са $[A] - [B] \rightarrow A$. Бит преноса добијен овом операцијом смешта се у C бит.
3. **Множење**⁴. Операција множења извршава се са 8-битним бинарним бројевима без знака применом акумулатора A или B, а резултат се смешта у 16-битни (двоструки) акумулатор D. Операција се може симболично представити као $[A] \times [B] \rightarrow D$.

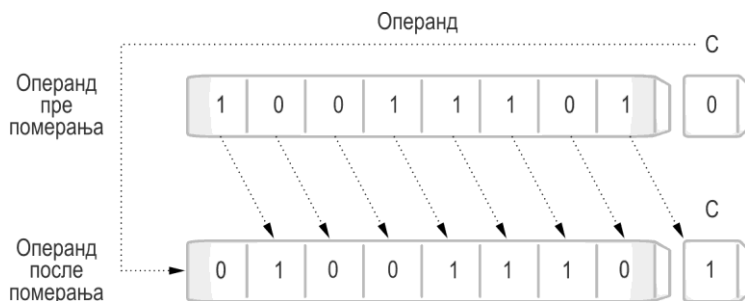
¹ Add

² Користи се ознака A_x да укаже да се релација односи и на акумулатор A и на акумулатор B

³ Subtract

⁴ Multiply

4. **Целобројно дељење**¹. Ова операција дели 16-битни операнд без знака, који се налази у акумулатору D, са 16-битним операндом који се налази у индекс регистру X, а резултат се смешта у акумулатор D. Симболично ову операцију можемо представити на следећи начин: $[D] / [X] \rightarrow D$
5. **Поређење**². Операција се извршава на исти начин као и одузимање али се резултат не смешта у акумулатор. Симболично написано је $[A_x] - [M]$ или $[A_x] - [A_y]$. Одузимање се обавља са циљем да се установи који је операнд већи и нема никаквог утицаја на садржај акумулатора. У зависности од тога да ли је резултат позитиван, негативан или нула, различити битови биће постављени у регистру услова. Значи, ова операција нема никаквог утицаја на операнде већ само на регистар услова.
6. **Логичко И**³. Операција логичког И обавља се над одговарајућим битовима два операнда. Симболично се представља као $[A_x] \cdot [M] \rightarrow A_x$. Један операнд је увек у једном од акумулатора, а други се очитава из меморије. Као пример претпоставимо да је $[A] = 10110101_2$ и да је $[M] = 01100001_2$. Операција логичког И представљена је на слици 5.11.
Уочимо да је сваки бит резултата добијен логичком И операцијом одговарајућих битова операнда. На пример 7. бит акумулатора



Слика 5.11 Логичка операција И

¹ Integer Divide

² Compare

³ Logical AND

је 1, а операнда из меморије 0, па је резултат $1 \cdot 0 = 0$. Слично 5. бит оба операнда је 1, тако да је 5. бит резултата $1 \cdot 1 = 1$.

7. **Логичко ИЛИ**¹. Одговарајући битови два операнда су у логичкој ИЛИ операцији, а резултат се налази у акумулатору. Симболично се операција може представити као $[A_x] \text{ ИЛИ } [M] \rightarrow A_x$. Уместо ознаке ИЛИ (OR) може се употребити знак сабирања („+“), али се тиме може изазвати забуна у односу на операцију бинарног сабирања. Користећи иста два операнда као у примеру са И операцијом, резултат ИЛИ операције биће 11110101_2 .
8. **Ексклузивно ИЛИ**². Операција се примењује на битове два операнда а резултат се смешта у акумулатор. Симболично се може представити са $[A_x] \oplus [M] \rightarrow A_x$. Користећи два иста операнда из претходног примера, резултат ЕИЛИ (EOR) операције је 11010100_2 . Програмеру су врло корисне анализиране логичке операције.

5.5 ПИТАЊА И ЗАДАЦИ

1. Описати све интерне 16-битне регистре процесора MC68HC11.
2. а) Коју функцију обавља RESET сигнал код процесора MC68HC11?
б) Шта одређује са које адресе ће процесор почети са извршавањем програма после појаве спољашњег RESET сигнала?
3. а) Објаснити разлику између прекидних сигнала XIRQ и IRQ процесора MC68HC11.
б) Набројати прекидне векторе процесора MC68HC11.
4. Стек меморија је:
а) део ROM меморије и користи се за смештање табела и програма
б) део RAM меморије и користи се за привремено смештање садржаја интерних регистара процесора
в) део RAM меморије а користи се за смештање програма који ће се користити у будућности
г) део ROM меморије и користи се за управљање улазно/излазним модулима
5. Када се извршава инструкција гранања испитује се садржај једног од

¹ Logical OR

² Exclusive OR

5. Процесор

интерних регистара. Који је то регистар:

- а) индекс регистар
- б) регистар услова
- в) инструкцијски регистар
- г) регистар адресе податка

6. Систем „први уписан, последњи прочитан (LIFO)¹” описује начин рада:

- а) RAM меморије
- б) ROM меморије
- в) аритметичко/логичке јединице
- г) стек меморије

7. Претпоставити да је садржај указивача стек меморије [SP]=\$010F. Уколико су извршене четири PUSH инструкције и две PULA инструкције који је садржај указивача стек меморије после извршавања ових инструкција?

8. Каква је разлика између SHIFT и ROTATE инструкција?

9. У чему је разлика између инструкције COMPARE и SUBTRACT?

10. а) Коју улогу има двоструки акумулатор D за време извршавања инструкције множења?
- б) Коју улогу има двоструки акумулатор D за време извршавања инструкције дељења?

¹ Last in First Out

6. Програмирање процесора MC68HC11

У овом поглављу анализираћемо рад процесора који је задужен за извршавање свих инструкција у програму. Процесор¹ MC68HC11 може да извршава инструкције микропроцесора MC6800 и MC6801 (компатибилност и изворног и објектног кода) и још 90 нових операционих кодова. Пошто постоји више од 256 инструкцијских кодова², користи вишестранично повезивање операционих кодова³, у коме су нове инструкције одређене вредношћу бајта (предбајт⁴) пре операционог кода који указује на страницу на којој се налази бајт операционог кода.

Архитектура процесора MC68HC11 таква је да се периферије и меморија третирају на идентичан начин у оквиру 64kB меморијског простора⁵. Значи, не постоје посебне инструкције за приступање улазно/излазним уређајима.

Процесор MC68HC11 нуди неколико нових могућности у поређењу са ранијим процесорима M6801 и M6800. Највећа разлика је додатни 16-битни индекс регистар (Y). Постоји скуп нових инструкција за операције са битовима било које меморијској локацији, 64kB адресног простора. Постоји и нова инструкција за дељења 16-битног броја са 16-битним бројем. Инструкције за размену садржаја омогућавају да се садржаји оба индекса регистра размењују са садржајем 16-битног, двоструког, акумулатора D. Такође, неколико инструкција је иновирано тако да омогућавају једноставније извођење 16-битних аритметичких операција.

¹ Правилно би било: „процесор микроконтролера MC68HC11“, али ћемо ми у овом уџбенику да користимо израз „процесор MC68HC11“ подразумевајући да је реч о процесору микроконтролера MC68HC11, а не процесору чија је ознака MC68HC11.

² Један бајт је резервисан за операциони код. То даје максимално 256 различитих операционих кодова.

³ *Multiple-page opcode map*

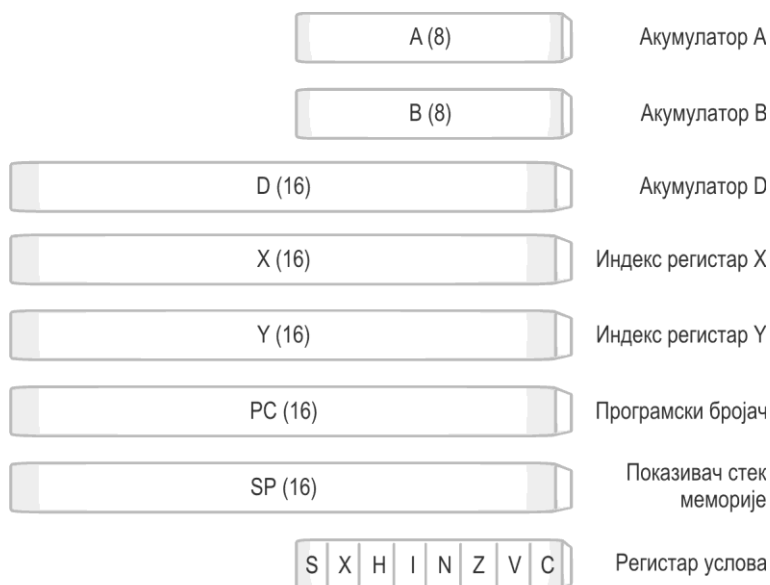
⁴ *Prebyte*

⁵ Детаљније објашњење може се наћи у 4. поглављу овог уџбеника.

6. Програмирање процесора MC68HC11

У даљем тексту биће анализирани сви детаљи архитектуре процесора, врсте адресирања¹ и инструкцијски скуп (по врстама инструкција). Примерима ће бити детаљније приказан начин на који се користе и инструкцијски скуп и специфичности архитектуре процесора. Детаљан опис сваке инструкције дат је у 11. поглављу.

Процесор MC68HC11 располаже са 108 различитих врста (типова) инструкција. Неке од њих већ смо анализирали у претходним поглављима. Сваки процесор има свој јединствен машински и асемблерски језик. Међутим, врсте инструкција и начин програмирања који користи процесор микроконтролера MC68HC11 слични су онима које користи већина 8-битних процесора. Тако, научивши програмирање процесора MC68HC11 релативно једноставно се учи и програмирање других 8-битних, 16-битних и 32-битних процесора.



C - пренос, V - премашење, H - пренос са четвртог бита, I - маскирање прекида,
N - негативни резултат, Z - резултат је нула, X - маскирање прекида,
S - спречавање извршења STOP инструкције

Слика 6.1 Програмски модел процесора MC68HC11

¹ Addressing modes- начин адресирања

6.1 ПРОГРАМСКИ МОДЕЛ ПРОЦЕСОРА MC68HC11

На слици 6.1 представљен је програмски модел процесора MC68HC11. Регистри су интегрални део процесора и не могу се адресирати као да су меморијске локације. Програмски модел садржи само оне интерне регистре процесора који су доступни програмеру. То су: два 8-битна акумулатора (A, B), један 16-битни, двоструки акумулатор (D)¹, два 16-битна индекс регистра (X, Y), један 16-битни показивач стек меморије (SP²), један 16-битни програмски бројач (PC) и регистар услова (CCR).

6.2 НАЧИН АДРЕСИРАЊА ПРОЦЕСОРА MC68HC11

Да би се указало на меморијске локације (референцирана меморија) код процесора MC68HC11 шест врста (начина) адресирања може се употребити: непосредно³, директно⁴, проширено⁵, индексирано⁶ (са било којим од два 16-битна индекс регистра и 8-битним одступањем), инхерентно⁷ и релативно⁸. Неке од инструкција захтевају додатни бајт⁹ пре операционог кода да би се обезбедило вишестранично повезивање операционих кодова¹⁰.

Сваки од начина адресирања (осим инхерентног) као резултат даје интерно генерисану двобајтну вредност¹¹ која се означава као ефективна адреса. Ова вредност, која је резултат дела инструкције са операндима, јесте вредност која се појављује на адресној магистрали за време приступања меморији. Начин адресирања је имплицитни део сваке од MC68HC11 инструкција.

¹ *Double Accumulator*

² *Stack Pointer*

³ *Immediate*

⁴ *Direct*

⁵ *Extended*

⁶ *Indexed*

⁷ *Inherent*

⁸ *Relative*

⁹ *Prebyte*

¹⁰ *Multiple-page opcode map*

¹¹ Објашњено је у 3. поглављу, одељак: Извршавање програма

Ефективна адреса користи се да укаже на меморијску локацију са које се узима или на коју се смешта операнд, или са које се наставља са извршавањем програма.

Инструкције могу бити класификоване и према томе како је специфициран део инструкције о адреси операнда. У досадшњој анализи о програмирању процесора користили смо две врсте инструкција. Једна врста (SUBA, STAA) су тробајтне инструкције које се састоје од операционог кода и двобајтне адресе операнда. Друга врста (CLRA, WAI) су једнобајтне инструкције, само операциони код, пошто адреса операнда није потребна.

Као што ће се видети, ниједна од инструкција не користи све начине адресирања. На пример инструкција за сабирање (LDAA) може да користи четири (од шест) начина адресирања.

6.2.1 Проширено адресирање

Код проширеног начина адресирања ефективна адреса операнда се налази иза операционог кода у следећа два бајта. Тако је дужина највећег броја инструкција које користе проширено адресирање три бајта. Формат за проширено адресирање је:

Бајт 1	Операциони код	двобајтна адреса операнда
Бајт 2	АД _В	
Бајт 3	АД _Н	

где је са АД_В означен адресни бајт веће тежине, а са АД_Н адресни бајт мање тежине.

Пример инструкције која користи проширено адресирање је:

	Машински код	Асемблерски језик
Операциони код →	B6	LDAA \$0E07
Адреса операнда →	0E	
	07	

При извршењу инструкције процесор узима податак са адресне локације \$0E07 и уписује га у акумулатор А. Операциони код \$B6 налаже

процесору да узме следећа два бајта да би одредио адресу операнда. Еквивалентан асемблерски исказ ове инструкције је LDAA \$0E07. Мнемоник указује да треба извршити операцију „напуни акумулатор А”. Део „\$0E07“ означава хексадецималну вредност адресе меморијске локације или улазног модула.

Последња два бајта инструкције садрже апсолутну адресу операнда. Ове инструкције могу бити тробајтне или четворобајтне (уколико је потребан предбајт): један за операциони кођ и два за адресу. Инструкције са друге, треће и четврте странице захтевају предбајт. Само четири инструкције са проширеним адресирањем, које користе индекс регистар Y, захтевају предбајт.

Машински кођ	Асемблерски језик
BB	ADDA \$0FC9
0F	
C9	

Пример 6.1 Описати шта се дешава приликом извршавања следеће инструкције?

Решење:

Приликом извршавања ове инструкције процесор ће узети податак са локације \$0FC9, и додати га на садржај акумулатора А. Операциони кођ \$BB указује процесору које операције треба да извршава (ADDA) и која врста адресирања се користи. Иза операционог кођа је двобајтна адреса.

6.2.2 Директно адресирање

Код директног адресирања бајт мање тежине ефективне адресе налази се у бајту иза операционог кођа. Формат директног адресирања приказан је на следећи начин:



6. Програмирање процесора MC68HC11

Први бајт је операциони кôд. Други бајт је адресни бајт мање тежине (AD_n). Није специфициран бајт веће тежине ефективне адресе. Подразумева се да је бајт веће тежине \$00 и не појављује се као део инструкције (штеди се меморијски простор и време потребно за извршавање инструкције).

Пример који користи директно адресирање је:

	Машински кôд	Асемблерски језик
Операциони кôд →	96	LDAA \$50
Адреса операнда →	50	

Ова инструкција наложиће процесору да узме податак са локације чија је адреса \$0050 и напуни њиме акумулатор А. То се може написати као LDAA \$50 (или LDAA \$0050). Када процесор интерпретира операциони кôд \$96, зна да се користи директно адресирање. Он затим узима нижи бајт адресе $AD_n = \$50$ и аутоматски додаје виши бајт адресе $AD_b = \$00$ да би се формирала ефективна (потпуна) адреса операнда \$0050.

Пример 6.2 *Описати шта се дешава приликом извршавања следеће инструкције?*

Машински кôд	Асемблерски језик
D0	SUBB \$3D
3D	

Решење:

Извршавање ове инструкције доводи до тога да ће процесор узети податак са локације \$003D, и одузети га од садржаја акумулатора В. Операциони кôд \$D0 указује процесору коју операцију треба да извршава (SUBB) и коју врсту адресирања да користи. Иза операционог кода је једнобајтна адреса.

Чињеница да је виши бајт адресе \$00 ограничава употребу директног адресирања на област меморије \$0000–\$00FF. Директно адресирање увек формира ефективну адресу операнда, тако да је на нултој страници. Због тога се овај начин адресирања често назива адресирање нулте странице.

Директно адресирање даје могућност корисницима да приступају области \$0000–\$00FF користећи инструкције које користе један бајт мање меморијског простора програма од одговарајуће инструкције са проширеним адресирањем. Елиминисањем додатног меморијског простора време за извршавање програма смањује се за један циклус. У случају великих програма уштеде су значајне. За највећи број апликација подразумевана меморијска мапа микроконтролера је таква да поставља интерну RAM меморију у \$0000–\$00FF област. То има својих предности пошто пројектанти могу да доделе те локације подацима којима се често приступа.

У неким апликацијама пожељно је да се лоцирају интерни регистри периферијских јединица у овај простор. Овај начин је пожељан када постоји потреба за интензивном комуникацијом са улазно/излазним модулима. Тако се постиже уштеда у меморијском простору и убрзава извршавање програм што је важно код временски захтевних апликација.

Код микроконтролера MC68HC11 софтверски се може конфигурисати меморијска мапа тако да интерна RAM меморија, и/или интерни регистри, или екстерни меморијски простор може да заузме ове адресе.

6.2.3 Непосредно адресирање

Код непосредног адресирања стварни аргумент (операнд) смештен је (налази се) у бајту (бајтовима) који је одмах иза операционог кода инструкције. Број бајтова одговара величини регистара. Ове инструкције могу да буду величине два, три или четири бајта (уколико се предбајт користи). Машински код (бајт) који следи иза операционог кода је вредност операнда (сам операнд), а не његова адреса.

Постоје многе ситуације где су подаци које треба уписати у регистре или акумулаторе познате, одређене величине. Нпр. може да постоји корак у програму где се жели напунити акумулатор А вредношћу \$24, или где је потребно додати \$02 на постојећи садржај акумулатора В. За ситуације као што је ова није неопходно да се специфицира адреса операнда; уместо тога операнд се налази непосредно иза операционог кода. Ово се зове непосредно адресирање и његов формат је:

6. Програмирање процесора MC68HC11



Први бајт је операциони кôд. Други бајт није адреса операнда него операнд са којим ће процесор радити за време извршавања инструкције.

Следи пример инструкције која користи непосредно адресирање:

	<i>Машински кôд</i>	<i>Асемблерски језик</i>
Операциони кôд →	86	LDAA #\$24
Операнд податак →	24	

Ова инструкција ће наложити процесору да напуни акумулатор А операндом (податком \$24) који се састоји од једног бајта. Податак \$24 није адреса операнда већ сам операнд. Вредност операнда је непосредно иза адресе операционог кода. Асемблерски запис ове инструкције је LDAA #\$24. Симбол „#” се користи да би означио непосредно адресирање; вредност која се налази иза симбола „#” увек је податак. Честа грешка програмера је да забораве да напишу „#” знак.

У примеру који следи дат је пример 6 инструкција са непосредним адресирањем. Вредност сваког операнда је у бајту непосредно иза операционог кода.

<i>Машински кôд</i>	<i>Операција</i>	<i>Операнд</i>	<i>Коментар</i>
86 16	LDAA	#22	$22_{10} \rightarrow A, [A]=16_{16}$
C8 34	EORB	#\$34	$EXOR(\$34, B) \rightarrow B$
81 24	CMPS	##100100	Упоредбавање [A] са 00100100_2
CC 12 34	LDD	#\$1234	$1234_{16} \rightarrow A:B, [A:B]=1234_{16}$
86 41	LDAA	# 'a	$ASCII\ A \rightarrow A, [A]=61_{16}=00110001_2$
CE 10 00	LDX	#\$1000	$1000_{16} \rightarrow X, [X]=1000_{16}$

Различити симболи и објашњења могу се користити и за знак „#”. Пошто не користе сви асемблери иста правила за синтаксу и специјалне карактере препоручује се одговарајућа документација за асемблер који се користи. Префикс знаци који су коришћени у претходном примеру су:

Префикс	Дефиниција
Без знака	Децимални број
\$	Хексадецимални број
@	Октални број
%	Бинарни број
'	Један ASCII карактер

Пример 6.3 Описати шта се дешава приликом извршавања следеће инструкције.

	Машински кôд	Асемблерски језик
Операциони кôд →	CE	LDX #\$01A9
16-битни операнд (податак) →	01 A9	

Решење:

Дата инструкција користи непосредно адресирање на шта указује симбол „#“. Пошто је индекс регистар X 16-битни, подаци који се читавају у тај регистар одређени су са два бајта који се налазе непосредно иза операционог кода. Ова тробајтна инструкција смешта (пуни) у индекс регистар X податак \$01A9.

Пример 6.4 Које операције се дешавају приликом следеће инструкцијске секвенце?

Адреса	Машински кôд	Асемблерски језик
0200	96	LDAA \$3A
0201	3A	
0202	8B	ADDA #\$16
0202	16	
0204	B7	STAA \$F000
0205	F0	
0206	00	

Решење:

Прва инструкција пуни акумулатор А податком са адресе \$003А, и користи директни начин адресирања. Подсетимо се да се адреса интерпретира као да је на страници 00 (виши бајт адресе АД_В = 00).

Друга инструкција додаје на садржај акумулатора А број \$16, што значи да користи непосредно адресирање.

Трећа инструкција смешта садржај акумулатора А на локацију \$F000, користи 16-битну адресу, тј. проширени начин адресирања.

6.2.4 Имплицитно адресирање

За имплицитно адресирање користе се једнобајтне инструкције код којих је информација о операнду садржана у операционом коду инструкције. У имплицитној врсти рада све што је потребно за извршавање инструкције процесор сазнаје из операционог кода. Операнди (уколико постоје) смештени су у регистрима процесора и не узимају се из меморије. Многе инструкције процесора MC68HC11 користе садржаје регистара као операнде. На пример АВА инструкција проузроковаће да процесор сабере садржаје акумулатора А и В и смести резултат у акумулатор А. Инструкција INCB проузроковаће да се садржај акумулатора В увећа за 1 (инкрементира). Слично INX инструкција доводи до тога да индекс регистар увећа свој садржај за 1. Навешћемо примере ових инструкција.

Машински кôд	Операција	Операнд	Коментар
1B	ABA		A+B → A
5C	INCB		B+1 → B
08	INX		X+1 → X

На пример, „обриши акумулатор А” има мнемоник CLRA и операциони кôд \$4F. Када процесор учита операциони кôд, он на основу њега зна да је операнд садржај акумулатора А. Процесор даље обезбеђује операције којима се постиже да садржај акумулатора А има вредност \$00 ([A]=\$00).

Други пример је инструкција „инкрементирај X регистар”. Мнемоник је INX, а операциони кôд је \$08. Овај операциони кôд указује процесору да је операнд садржај X регистра. Процесор инкрементира X (додаје му 1).

Све инструкције које бришу битове у регистрима услова користе имплицитно адресирање. Нпр. инструкција за „брисање С бита“ има мнемоник CLC и операциони код \$0D. Овај операциони код указује процесору да је операнд садржај регистра услова, и да процесор поставља С бит на 0 (С бит регистра услова).

Пример 6.5 *Колики ће бити садржај X регистра по завршетку следеће инструкцијске секвенце?*

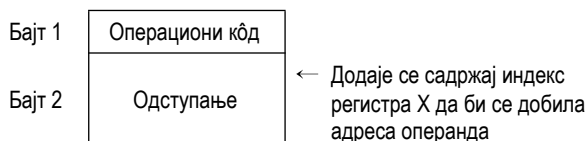
Машински код	Операција	Операнд	Коментар
CE 00 01	LDX	#\$01	01 → X, [X]=0001 ₁₆
09	DEX		[X]-1 → X, [X]=0000 ₁₆
09	DEX		[X]-1 → X, [X]=FFFF ₁₆

Решење:

Инструкција LDX #\$01 користи непосредно адресирање да би напунила X регистар са 16-битним податком 0001₁₆ = 0000 0000 0000 0001₂. Прва инструкција DEX ће декрементирати индекс регистра X на 0000₁₆, а следећа на FFFF₁₆.

6.2.5 Индексно адресирање

Код индексног адресирања индекс регистри X или Y користе се за израчунавање ефективне адресе. Ефективна адреса је променљива и зависи од текуће вредности индекс регистра (X и Y) и броја без знака (одступања) који је садржан у инструкцији. Овај начин адресирања може се користити за референцирање било које меморијске локације у 64KB адресном простору. Формат индексног адресирања је (без предбајта):



Овај начин адресирања изузетно је погодан при адресирању табела или блокова података у меморији. Први бајт увек је операциони код. Други бајт је одступање (офсет) адресе, и то је 8-битни број без знака.

Одступање се додаје на постојећи садржај индекса регистра X да би се израчунала ефективна адреса операнда. Садржај индекса регистра остаје неизмењен. Пошто је одступање једнобајтни број без знака, прихватљиве су само вредности у опсегу 0–255. Другим речима:

$$\text{ефективна адреса операнда} = [X] + \text{одступање}$$

Овим начином адреса операнда није дата експлицитно процесору; уместо тога она треба да израчуна адресу операнда додавањем бајта одступања на садржај индекса регистра X. Пример индексног адресирања је следећи:

	<i>Машински кôд</i>	<i>Асемблерски језик</i>
Операциони кôд →	A6	LDAA \$2A,X
Одступање →	2A	

Ова инструкција наложиће процесору да дода \$2A на садржај индекса регистра X да би се добила адреса операнда. Процесор ће затим напунити акумулатор A садржајем меморијске локације на овој адреси.

Асемблерска презентација ове инструкције је LDAA \$2A,X, што се интерпретира на следећи начин: напуни акумулатор A са податком са адресне локације добијене додавањем вредности \$2A на садржај X индекса регистра.

У микрорачунарским системима инструкције су обично смештене у ROM меморији. Због тога ће одступање бити тачно дефинисана вредност, која се одређује за време асемблирања програма а не за време извршавања програма. Употреба једнобајтног динамичког одступања омогућава се употребом инструкције - додај садржај акумулатора A на садржај индекса регистра X (ABX). Сложеније израчунавање адреса је потпомогнуто аритметичким могућностима 16-битног акумулатора D код инструкција XGDY и XGDY.

Уколико одступање није специфицирано машински кôд ће на месту бајта одступања имати вредност \$00. Различити начини употребе индексног адресирања дати су у следећој табели:

Машински кôд	Лабела	Операција	Операнд	Коментар
E3 00		ADDD	X	EA= [X]
E3 00		ADDD	,X	EA= [X]
E3 00		ADDD	0,X	EA= [X]
E3 04		ADDD	4,X	EA= [X]+4
	KST	EQU	7	Дефинише се вредност константе KST=7
E3 07		ADDD	KST,X	EA=[X]+7
E3 22		ADDD	\$22,X	EA=[X]+22
E3 22		ADDD	KST*8/2+6,X	EA=[X]+ KST*8/2+6

Инструкције за рад (операције) са битовима подржавају директни и индексни начин адресирања али не и проширени. Индексно адресирање је веома битно за ове инструкције пошто директни начин адресирање омогућава да се приступи само до првих 256 меморијских локација. С друге стране, индексно адресирање дозвољава приступ било којој меморијској локацији у 64kB меморијском простору.

Други индексни регистар (Y) побољшава ефикасност операција померања и операција које користе податке из више од једне табеле. Већина инструкција које користе Y регистар захтевају двобајтни операциони кôд. То значи: додатни бајт за смештање у програмском делу меморије и додатни циклус за извршавање инструкције, упоређујући са еквивалентним инструкцијама које користе индекс регистар X.

Пример 6.6 Које су то операције које се извршавају у следећој инструкцијској секвенци?

Машински кôд	Лабела	Операција	Операнд	Коментар
CE F0 12		LDX	#\$F012	\$F012 → X, [X]= F012 ₁₆
EB D2		ADDB	\$D2,X	EA=[X]+ \$D2
E7 00		STAB	X	EA=[X]

Решење:

Прва инструкција пуни индекс регистар X податком \$F012 користећи непосредни начин адресирања. Друга инструкција користи индексно

адресирање да би додала податке акумулатору В, са адресе која се добија збиром адреса $F012_{16} + 00D2_{16} = F0E4_{16}$. Трећа инструкција смешта добијени податак из акумулатора В на адресу $\$F012$ ($F012_{16} + 001_6 = F012_{16}$), користећи индексно адресирање.

6.2.6 Релативно адресирање

У кратком приказу инструкција у 3. поглављу анализирали смо и инструкције са условним гранањем, које користе релативни начин адресирања. Видели смо како инструкције са условним гранањем, када се употреби одговарајући услов, могу да промене редослед инструкција које извршава процесор. Као што је наглашено, ова класа инструкција је оно што даје рачунару његову способност одлучивања, на бази задатог алгорита.

Релативно адресирање користи се само код инструкција гранања. Као што смо видели у 3. поглављу, инструкција гранања (за разлику од других инструкција за манипулацију са битовима) генерише два бајта машинског кода: један за операциони код а други за релативно одступање. Пошто је пожељно гранање у оба правца, бајт одступања је други комплемент броја опсега од -128 до +127. Уколико је услов за гранање испуњен садржај 8-битног бајта (одступање), који се налази иза операционог кода, додаје се на садржај програмског бројача да би се добила ефективна адреса гранања; у супротном (услов није испуњен) наставља се са извршавањем следеће инструкције (иза инструкције гранања). Бајт одступања увек је последњи бајт инструкције гранања. Уколико је бајт одступања 0, извршиће се следећа инструкција иза инструкције гранања, независно од испуњености услова. Инструкција „гранај се увек“¹ А (BRA) са одступањем $\$FE$ доведиће до бесконачне петље. Директни или индексирани начин адресирања „гранај се уколико су битови обрисани“ (BRCLR²) и „гранај се уколико су битови постављени“ (BRSET³) јесу четворобајтне инструкције. Због тога ће

¹ Branch always

² Инструкција извршава логичку операцију И између садржаја меморијске локације М и маске која се добија из инструкције. Гранање се реализује уколико је резултат нула (само уколико су сви битови који се налазе на месту јединица у бајту маске испитиваног бајта нуле).

³ Гранање се реализује уколико је резултат нула (само уколико су сви битови који се налазе на месту јединица у бајту маске испитиваног бајта јединице).

одступање \$FC проузроковати да се инструкција извршава непрекидно док критеријум тестирања не буде задовољен. Инструкције BR-CLR и BRSET које за индексано адресирање користе Y регистар су петобајтне инструкције; тако ће бајт одступања \$FB довести до поновљеног извршавања инструкције све док критеријум тестирања не буде задовољен. Примери релативног начина адресирања су:

Машински код	Лабела	Операција	Операнд	Коментар
20 00	NAZAD	BRA	NAPRED	Гранање на NAPRED
22 FC	NAPRED	BRA	NAZAD	Гранање на NAZAD
24 04		BCC	SKOK	Гранање на SKOK
27 FE	NASS	BEQ	NASS	Гранање на NASS (самог себе)
27 FE		BEQ	*	„*“ значи овде
7E 10 00	SKOK	JMP	\$1000	
8D F7		BSR	NASS	

6.3 ИНСТРУКЦИЈСКИ СКУП

У овом одељку биће анализирана основна својства и организација инструкцијског скупа. Због тога ће инструкцијски скуп бити подељен у групе инструкција. Да би се проширио број инструкција које се користе код процесора MC68HC11 додат је механизам са предбајтом.

Већина инструкција која користи предбајт везана је за индекс регистар Y. Инструкције које не захтевају предбајт налазе се на првој страници мапе операционих кодова. Инструкције које захтевају предбајт налазе се на страницама 2, 3 и 4 мапе операционих кодова. Када је предбајт код \$18 односи се на 2. страницу, када је код \$1A на 3. страницу, а када је код \$CD на 4. страницу. Предбајт код примењује се само на операциони код који је одмах иза њега.

6.3.1 Опис инструкција

У овом делу представимо неке примере о начину добијања различитих типова информација из 11. поглавља.

Пример 6.7 *Пронаћи операциони код за асемблерску инструкцију LDAA #\$20 и LDAB #\$20.*

Решење:

Прво, у 11. поглављу, треба пронаћи опис за инструкцију LDA, који нам уједно даје информације о инструкцијама LDAA и LDAB. LDAA и LDAB инструкције имају сва четири начина адресирања. Нас интересује непосредан начин адресирања због ознаке „#“ у асемблерској инструкцији. Из табеле се уочава да је операциони код за LDAA (непосредно) \$86. Тако ће се инструкција LDAA #\$20 превести у два бајта 86 20. Затим се проналази да је операциони код за LDAB (непосредно) \$C6. Тако се LDAB #\$20 преводи у два бајта, C6 20.

Пример 6.8 *Претворити инструкцију STX \$F000 у код машинског језика, и одредити начин адресирања у инструкцији.*

Решење:

STX \$F000 инструкција користи проширено адресирање да би податак из индекс регистра X сместила на локацију \$F000. Видимо из табеле да је операциони код за мнемоник „STX“ и проширено адресирање \$FF. Тако се STX \$F000 преводи у FF F0 00.

Пример 6.9 *Превести инструкцију STAA \$3D у машински код.*

Решење:

Инструкција STAA\$3D смешта садржај акумулатора А на локацију \$003D. Може се уочити да се користи директно адресирање пошто је адреса одређена само са две цифре \$3D, а ефективна адреса операнда је \$003D. Погледајмо мнемоник „STA“ и пронађимо у табели STAA (директно). Операциони код је \$97. Тако се STAA \$3D преводи у 97 3D.

Пример 6.10 *Инструкције које користи процесор MC68HC11 могу да извршавају одређене операције над подацима који се налазе у меморији или над подацима који се налазе у неком од акумулатора. Операција брисања је једна од таквих. Она поставља све битове на 0 (тј. постојећи садржај меморије или акумулатора замењује са 0016). Погледајмо у 11. поглављу мнемоник CLR. Ово је мнемоник инструкције која брише садржај меморијске локације. Погледајмо затим мнемонике CLRA/CLRB. Ово су мнемоници за брисање*

акумулатора А и В, респективно.

Конвертовати следеће инструкције у машински код: CLR \$F000 и CLRB.

Решење:

У 11. поглављу се проналази да је код за CLR (проширено адресирање) \$7F. Тако да се инструкција CLR \$F000 преводи у 7F F0 00. Операциони код за CLRB (имплицитно) је \$5F. Тако се CLRB претвара у \$5F. Инструкција CLRB је једнобајтна.

Пример 6.11 Упоредити време потребно за извршавање инструкције DECB (декрементирање акумулатора В) са временом потребним за извршавање инструкције DEX (декрементирање индекс регистра X).

Решење:

Мнемоник за декрементирање садржаја акумулатора В је „DECB“. Потребна су два такт циклуса за извршавање ове инструкције. Мнемоник за декрементирање садржаја индекс регистра X је „DEX“. Потребна су четири такт циклуса за извршавање ове инструкције. Више такт циклуса је потребно за декрементирање индекс регистра X него за декрементирање акумулатора В. Разлог је тај што је индекс регистар 16-битни, а процесор ради са 8-битним подацим, па су му потребна два корака више за декрементирање индекс регистра X.

Пример 6.12 Коју операцију извршава инструкција LDAB #\$D3, и на које битове у регистру услова утиче извршавање ове инструкције.

Решење:

Инструкција LDAB#\$D3 учитава податак \$D3 у акумулатор В користећи непосредно адресирање. У табели за инструкцију LDAB (11. поглавље) види се да смештање података у акумулатор утиче на N и Z битове - означено је са „Δ“ испод колона N и Z. У овом случају подаци су $D3_{16} = 11010011_2$. Пошто податак није нула: Z бит ће бити постављен на 0; N бит ће бити постављен на 1 јер је бит знака податка (бит највеће тежине) 1.

У табели се такође види да ће V бит бити обрисан (постављен на 0) кад год се инструкција изводи, без обзира на податке.

6.4 НАЧИН ПРЕДСТАВЉАЊА ПРОГРАМА - ЛИСТИНГ

Листинг програма, један од резултата процеса асемблирања представљен је на следећи начин:

Машински кôд	Лабела	Операција	Операнд	Коментар
0100	START	86	LDAA #\$D6	; Напуни акумулатор А подацима
0101		D6		
0102		B7	STAA \$0200	; Смести податке на адресу \$ 0200
0103		02		
0104		00		
0105	KRAJ	3E	WAI	; Заустави извршавање

Прва колона приказује хексадецималну меморијску адресу где су бајтови програма смештени.

Друга колона се користи да се дају имена или ознаке (лабеле) неким меморијским локацијама. На пример, локацији \$0100 дата је лабела START која указује да је то прва адреса у програму, а адреси \$0105 дата је лабела KPAJ која указује да је то последња инструкција у програму. Лабеле су део програма и могу се користити уместо стварних нумеричких адреса.

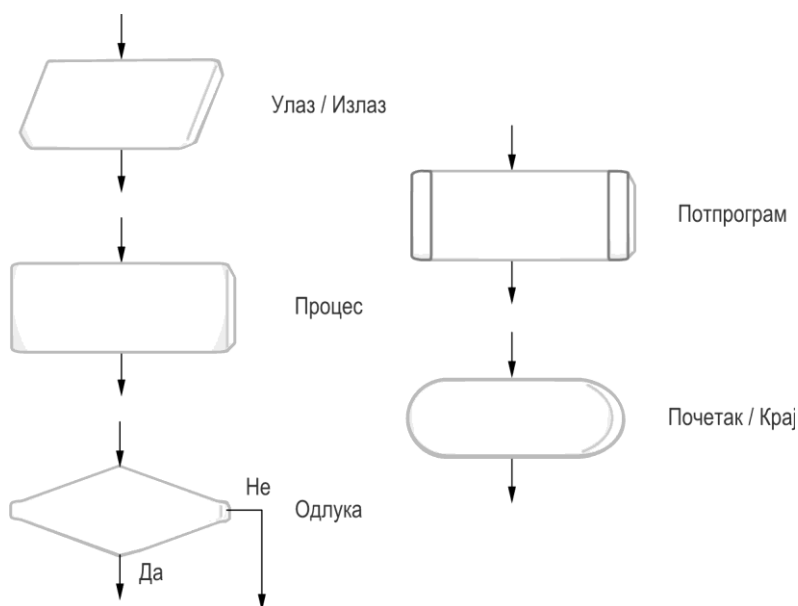
У трећој колони налази се хексадецимални инструкцијски кôд. Ови инструкцијски кодови представљају бинарни низ који се смешта у програмски део меморије (тј. програм у машинском језику).

У четвртој колони налазе се мнемоници (симболичне ознаке) заједно са адресама операнада или самим операндима (непосредна врста адресирања).

Пета колона садржи коментаре који објашњавају ток програма. Коментари немају само задатак да помогну при праћењу туђих програма, већ да олакшају програмеру да прати логику програма за време писања програма.

6.5 ДИЈАГРАМИ ТОКА И СТРУКТУРНО ПРОГРАМИРАЊЕ

У овом поглављу биће описана синтакса дијаграма тока која ће се користити у овом уџбенику. Програми се пишу у линеарном,



Слика 6.2 Символи дијаграма тока

једнодимензионалном начину. Писање програма на овај начин природан је процес пошто и сам рачунар извршава програм на такав начин, секвенционално, од врха до дна¹. Овај једнодимензионални облик добар је за једноставне програме али условна гранања и позиви потпрограма могу да усложне програме тако да се не могу једноставно посматрати у линеарном начину. Дијаграм тока је начин да се софтвер представи у дводимензионалном формату користећи погодан механизам за визуелизацију условних гранање и позива потпрограма. Није неопходно цртати дијаграм тока за сваки програм.

Ипак, дијаграми тока су корисни у иницијалној фази пројектовања софтверског система за дефинисање сложених алгоритама. Дијаграм тока се, такође, може користити у завршној фази документације, када систем већ проради, да би се омогућило његово коришћење и измене. На слици 6.2 представљени су симболи дијаграма тока.

¹ Top to bottom

Улазно/излазни модул се користи да би представио операцију учитавања података из улазних модула или операцију слања података ка излазном модулу. На пример, овај симбол се користи када процесор шаље податке на екран.

Модул за процесирање користи се да би представио било коју операцију коју обавља процесор над подацима (осим улазно/излазних операција). Примери операција које могу да буду укључене у модул за процесирање су аритметичке операције, логичке операције, померање и пребацивање података.

Модул облика ромбоида користи се сваки пут кад процесор треба да донесе неку одлуку. Уобичајена ситуација је када процесор изводи инструкцију условног скока. Блок за одлучивање има две излазне стрелице које указују на два могућа пута која процесор може да одабере.

Потпрогра¹ је скуп инструкција које одрађују специјалан задатак који је програмеру потребан да се изврши неколико пута за време извршавања главног програма.

Симбол за почетак/крај користи се за представљање почетних и завршних тачака у програму или потпрограму. У оквиру овог модула користе се речи као што су START, BEGIN (почетак) и END и STOP (завршетак), и RETURN (повратак).

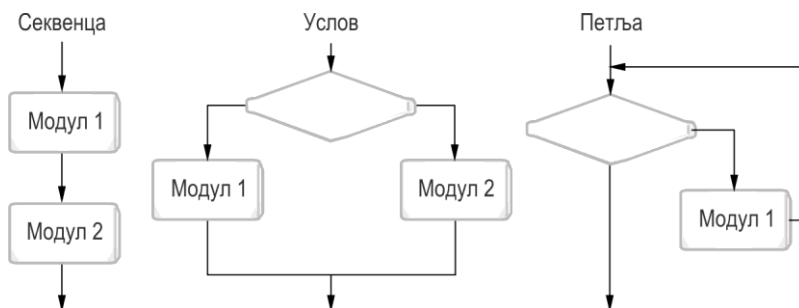
Један од кључних задатака када се развија софтвер, поготову у асемблерском језику, је одржавање конзистентне структуре. Један од начина на који се то може обезбедити је структурно програмирање. Већина виших програмских језика (са изузетком GOTO) намеће програмерима структурно програмирање.

Структурни програми сачињени су од три основна градивна блока: секвенца, услов² и петља³. Најнижи ниво модула за обраду садржи команде. Улазно/излазни модул је такође градивни модул ниског нивоа. Структурно програмирање укључује постојеће модуле у много сложеније структуре, као што је приказано на слици 6.3.

¹ Subroutine

² Conditional

³ While loop



Слика 6.3 Повезивање модула код структурног програмирања

6.6 ИНСТРУКЦИЈЕ АКУМУЛАТОР/МЕМОРИЈА

Већина инструкција користи два операнда. Један операнд је садржај акумулатора или индекс регистра; други операнд се обично добија из меморије користећи један од описаних начина адресирања. Инструкције акумулатор/меморија могу се поделити у шест група:

1. узимање (пуњење¹), смештање² и пренос³;
2. основне аритметичке операције;
3. множење и дељење;
4. логичке операције;
5. испитивање вредности и рад битовима;
6. померање и ротирање.

6.6.1 Узимање, смештање и пренос

Скоро све активности процесора укључују пренос података из меморије и периферија у процесор или пренос резултата из процесора у меморију или улазно/излазне модуле.

Постоје и инструкције за узимање, смештање и пренос које су везане за индекс регистре и показивач стек меморије. Инструкције узимања, смештања и преноса приказане су у табели 6.1.

¹ Load

² Store

³ Transfer

6. Програмирање процесора MC68HC11

Функција	Мнемоник	Непосредно	Директно	Проширено	Индексирано (X регистар)	Индексирано (Y регистар)	Имплицитно
Обрисати бајт меморије (<i>Clear Memory Byte</i>)			X	X	X		
Обрисати акумулатор A (<i>Clear Accumulator A</i>)						X	
Обрисати акумулатор B (<i>Clear Accumulator B</i>)						X	
Напунити акумулатор A (<i>Load Accumulator A</i>)	X	X	X	X	X		
Напунити акумулатор B (<i>Load Accumulator B</i>)	X	X	X	X	X		
Напунити акумулатор D (<i>Load Double Accumulator D</i>)	X	X	X	X	X		
Узми – повуци са стек меморије и смести у акумулатор A (<i>Pull A from Stack</i>)						X	
Узми – повуци са стек меморије и смести у акумулатор B (<i>Pull B from Stack</i>)						X	
Смести-гурни садржај акумулатора A на стек меморију (<i>Push A onto Stack</i>)						X	
Смести-гурни садржај акумулатора B на стек меморију (<i>Push B onto Stack</i>)						X	
Смести садржај акумулатора A (<i>Store Accumulator A</i>)	X	X	X	X	X		
Смести садржај акумулатора B (<i>Store Accumulator B</i>)	X	X	X	X	X		
Смести садржај акумулатора D (<i>Store Double Accumulator D</i>)	X	X	X	X	X		
Пренеси садржај акумулатора A у B (<i>Transfer A to B</i>)						X	
Пренеси садржај акумулатора A у регистар услова (<i>Transfer A to CCR</i>)						X	
Пренеси садржај акумулатора B у A (<i>Transfer B to A</i>)						X	
Пренеси садржај регистра услова у акумулатор A (<i>Transfer CCR to A</i>)						X	
Замени садржаје акумулатора D и индекс регистра X (<i>Exchange D with X</i>)						X	
Замени садржаје акумулатора D и индекс регистра Y (<i>Exchange D with Y</i>)						X	

Табела 6.1 Инструкције узимања, смештања и преноса

Пример 6.13 У следећем кратком програму одредите функције сваке од инструкција, садржај акумулатора А и В и битова N и Z по завршетку програма.

START	LDAA #\$3E	; Напуни А са подацима
	STAA \$F0	; Смести В у меморију
	LDAB \$F0	; Напуни В из меморије
KRAJ	WAI	; Заустави извршавање

Решење:

Инструкција LDAA #\$3E пуни акумулатор А са бројем \$3E користећи непосредан начин адресирања. Друга инструкција, STAA \$F0, користи директан начин адресирања да би сместила садржај акумулатора А на меморијску локацију \$00F0. Инструкција LDAB \$F0 пуни акумулатор В са подацима са адресе \$00F0 користећи директни начин адресирања. Последња инструкција, WAI, суштински зауставља процесор. Табела у даљем тексту сумира статус регистара процесора, битове регистра услова и садржаје меморијских локација после извршења сваке инструкције.

Инструкција	Статус после извршења
LDAA #\$3E	[ACCA] = $3E_{16} = 00111110_2$; N=0, Z=0
STAA \$F0	[\$00F0] = $3E_{16} = 00111110_2$; N=0, Z=0
LDAB \$F0	[ACCB] = $3E_{16} = 00111110_2$; N=0, Z=0
WAI	; Нема промена

Пример 6.14 Програм који следи смешта садржаје акумулатора А и В негде у меморију. Одредити адресе у које се смештају садржаји акумулатора.

START	LDS #\$FE00	; \$FE00 $\Rightarrow$ SP, [SP]= \$FE00
	PSHA	; Смести садржај А на стек меморију
	PSHB	; Смести садржај В на стек меморију
KRAJ	WAI	; Заустави извршавање

Решење:

LDS #\$FE00 инструкција пуни указивач стек меморије са \$FE00 користећи непосредно адресирање. Указивач стек меморије је 16-битни регистар, тако да се морају навести два бајта иза оперативног кода који одређују његову позицију. Инструкција PSHA смешта садржај акумулатора А у меморијску локацију одређену садржајем указивача стек меморије \$FE00 и затим декрементира указивач стек меморије на \$FDFF. Инструкција PSHB смешта садржај акумулатора А на меморијску локацију одређену са [SP]=\$FDFF и затим декрементира [SP] на вредност \$FDFF.

На крају: садржај акумулатора А смештен је на локацију \$FE00, а садржај акумулатора В на локацију \$FDFF.

Пример 6.15 У примеру 6.13 за реализацију функције копирања садржаја акумулатора А у акумулатор В употребљено је више инструкција него што је неопходно. Употребити одговарајуће инструкције да би се направила ефикаснија верзија програма.

Решење:

У примеру 6.13 коришћена је меморија као посредник између акумулатора А и В. Ово се може избећи коришћењем TAB инструкције као што је приказано у овом решењу:

START	LDAA #\$3E	; \$3E → A, [A]=\$3E
TAB		; [A] → B, [B]= [A]
KRAJ	WAI	; Заустави извршавање

Пример 6.16 Написати програм који узима реч податка с врха стека и смешта је у акумулатор А, без промене садржаја показивача стек меморије (SP).

Решење:

Прво на шта помислимо код учитавања података с врха стек меморије и стављања у акумулатор је коришћење PULA инструкције. PULA инструкција у овом примеру не може да се користи јер при свом извршавању инкрементира показивач стек меморије пре него што скине податке са стек меморије. Као што је познато, показивач стек меморије декрементирао су када се подаци стављени на стек меморију.

Други начин да се превазиђе претходно описан поступак приказан је следећим програмом.

START	TSX	; [SP] → X
	LDA \$00,X	; Напуни А са адресе EA=[X]+00
KRAJ	WAI	; Заустави извршавање

Да бисмо анализирали програм претпоставимо да је $[SP] = 036B_{16}$. Показивач стек меморије указује на адресу $036B$, која је следећа слободна локација у стек меморији; адреса $036C$ је врх стек меморије, где је смештен последњи податак који је послат на стек меморију. Анализирајући програмску секвенцу:

TSX инструкција преноси садржај показивача стек меморије (SP) у индекс регистар X а затим инкрементира индекс регистар. По извршењу ове инструкције $[X] = 036B + 1 = 036C$. Инструкција LDA \$00,X користи индексно адресирање да би напунила акумулатор А са ефективне адресе дате са $(EA=[X]+00)$. Пошто је $[X] = 036C$, подаци узети са ове адресе се смештају у акумулатор А. На тај начин узети су подаци са врха стек меморије ($036C$) без икаквог утицаја на садржај показивача стек меморије.

		Стек меморија	
[SP] →	036B	????	← Врх стек меморије
	036C	подаци	
	036D	подаци	
	036E	подаци	
	036F	подаци	

Операције брисања

Ова операција брише сваки бит операнда (поставља на 0). Три типа инструкција за брисање су:

- CLR - брисање бајта у меморији,
- CLRA - брисање акумулатора А,
- CLRB - брисање акумулатора В.

CLR инструкција користи и проширено и индексно адресирање. Ове инструкције обично се користе да би се поставила меморијска локација или

акумулатор на 0, а пре него што се он користи као бројач. У програмирању бројач се користи да бисмо пратили колико пута се одређени догађај десио, или колико се пута нека програмска петља или операција извршила.

6.6.2 Аритметичке инструкције

Ова група инструкција омогућава аритметичке операције са различитим операндима. Могуће су 8-битне и 16-битне аритметичке операције, а једноставно се могу проширити и на аритметичке операције у којима су операнди вишебајтни. Могу се реализовати аритметичке операције са означеним и неозначеним бројевима¹. BCD аритметика може се реализовати употребом инструкције DAA која обезбеђује BCD формат.

На пример инструкција поређења обезбеђује одузимање операнда али ова операција има утицаја само на регистар услова, а не и на операнде. Тест инструкција ретко се користе пошто остале аритметичке операције аутоматски утичу на регистар услова.

Функција	Мнемоник	Непосредно	Директно	Проширено	Индексирано (X регистар)	Индексирано (Y регистар)	Имплицитно
Сабери садржаје акумулатора A и B (Add Accumulators)	ABA						X
Додај садржај акумулатора B на садржај индекс регистра X ² (Add Accumulator B to X)	ABX						X
Додај садржај акумулатора B на садржај индекс регистра Y (Add Accumulator B to Y)	ABY						X
Додај пренос C на садржај акумулатора A (Add with Carry to A)	ADCA	X	X	X	X	X	
Додај пренос C на садржај акумулатора B (Add with Carry to B)	ADCB	X	X	X	X	X	

¹ Означени бројеви су позитивни или негативни бројеви (бројеви са предзнаком). За рад са означеним бројевима користи се други комплемент. Код означених бројева бит највеће тежине представља предзнак. Код неозначених бројева сви битови одређују нумеричку вредност.

² Правилно је да се за сваку функцију у табели напише: садржај акумулатора A (B, D), садржај регистра X (Y), садржај меморијске локације... Због прегледности то се некада изоставља.

Функција	Мнемоник	Непосредно	Директно	Проширено	Индексирано (X регистар)	Индексирано (Y регистар)	Имплицитно
Додај садржај меморијске локације на садржај акумулатора A (<i>Add Memory to A</i>)	ADDA	X	X	X	X	X	
Додај садржај меморијске локације на садржај акумулатора B (<i>Add Memory to B</i>)	ADDB	X	X	X	X	X	
Додај садржај меморијске локације на садржај акумулатора D (<i>Add Memory to D - 16 Bit</i>)	ADDD	X	X	X	X	X	
Упореди садржаје акумулатора A и B (<i>Compare A to B</i>)	CBA						X
Упореди садржај акумулатора A са садржајем меморијске локације (<i>Compare A to Memory</i>)	CMPA	X	X	X	X	X	
Упореди садржај акумулатора B са садржајем меморијске локације (<i>Compare B to Memory</i>)	CMPB	X	X	X	X	X	
Упореди D са меморијом (<i>Compare D to Memory</i>) (16 Bit)	CPD	X	X	X	X	X	
Децимално прилагођење A (<i>Decimal Adjust A</i>) (for BCD)	DAA						X
Декрементирање меморијског бајта (<i>Decrement Memory Byte</i>)	DEC			X	X	X	
Декрементирање акумулатора A (<i>Decrement Accumulator A</i>)	DECA						X
Декрементирање акумулатора B (<i>Decrement Accumulator B</i>)	DECB						X
Инкрементирање меморијског бајта (<i>Increment Memory Byte</i>)	INC			X	X	X	
Инкрементирање акумулатора A (<i>Increment Accumulator A</i>)	INCA						X
Инкрементирање акумулатора B (<i>Increment Accumulator B</i>)	INCB						X
Други комплемент меморијског бајта (<i>Two's Complement Memory Byte</i>)	NEG			X	X	X	
Други комплемент акумулатора A (<i>Two's Complement Accumulator A</i>)	NEGA						
Други комплемент акумулатора B (<i>Two's Complement Accumulator B</i>)	NEGB						

6. Програмирање процесора MC68HC11

Функција	Мнемоник	Непосредно	Директно	Проширено	Индексирано (X регистар)	Индексирано (Y регистар)	Имплицитно
Одузети од A са преносом (<i>Subtract with Carry from A</i>)	SBCA	X	X	X	X	X	
Одузети од B са преносом (<i>Subtract with Carry from B</i>)	SBCB	X	X	X	X	X	
Одузети меморију од A (<i>Subtract Memory from A</i>)	SUBA	X	X	X	X	X	
Одузети меморију од B (<i>Subtract Memory from B</i>)	SUBB	X	X	X	X	X	
Одузети меморију од D (<i>Subtract Memory from D</i>) (16 Bit)	SUBD	X	X	X	X	X	
Тестирање да ли је нула или негативан (<i>Test for Zero or Minus</i>)	TST			X	X	X	
Тестирање да ли је нула или негативан A (<i>Test for Zero or Minus A</i>)	TSTA						X
Тестирање да ли је нула или негативан B (<i>Test for Zero or Minus B</i>)	TSTB						X

Табела 6.2 Аритметичке инструкције

У тексту који следи навешћемо пример коришћења инструкција из табеле 6.2.

Додавање садржаја акумулатора B на акумулатор A (ABA)

Ова инструкција додаје садржај акумулатора B на A и смешта резултат у акумулатор A. Резултат ће утицати на N, Y, C и V битове регистра услова. Подсетимо се да процесор не зна да ли су два операнда бројеви са или без знака; само програмер то зна. Процесор додаје два 8-битна броја као да су бројеви без знака.

Пример 6.17 Сабрати садржаје акумулатора A и B где је $[A]=01000101_2=69_{10}$ а $[B]=00110010_2=50_{10}$

Решење:

Процесор ће сабрати ова два броја као бинарне бројеве на следећи начин:

$$\begin{array}{rcl}
 & 01000101 & \text{акумулатор A} \\
 + & 00110010 & \text{акумулатор B} \\
 \hline
 & 01110111 & \\
 \end{array}$$

Нема преноса, па је C=0

Резултат је смештен у акумулатор A

Сабирање не производи пренос са бита највеће тежине па процесор брише C бит (C=0). Резултат је позитиван број (различит од нуле): процесор брише Z бит (Z=0) и N бит (N=0). N бит узима вредност исту као и бит знака резултата (и ако се бројеви са знаком не користе). Бит премашења V биће обрисан (V =0), пошто је знак резултата исти као битови знака оба операнда.

Подсетимо се да V бит служи да покаже премашење на позицији бита знака у случају сабирања и одузимања бројева са знаком.

Додај садржај меморијске локације акумулатору (ADDA/ADDB)

Ове инструкције се користе за додавање бајта податка из меморије једном од акумулатора, а резултат остаје у једном од акумулатора. Резултат утиче на N, Z, C и V битове. Свака од ових инструкција може се користити са неколико врста адресирања, као што је дато у 11. поглављу.

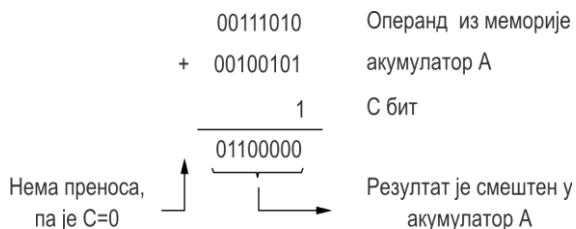
Додавање садржаја меморије садржају акумулатора са преносом (ADCA/ADCB)

Ове инструкције сличне су ADDA и ADDB инструкцијама с том разликом што C бит овде учествује у операцији сабирања. На пример, ADCA инструкција додаје операнд из меморије и C бит на садржај акумулатора A. Илустроваћемо два различита случаја.

Пример 6.18 Претпоставићемо да је C=1 пре операције ADCA.

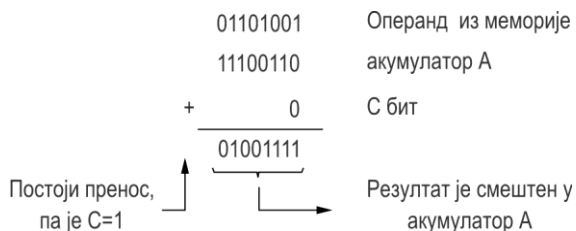
Решење:

Уочимо да се иницијалне вредности C бита додају при сабирању два операнда. Нема преноса са позиције бита највеће тежине; процесор ће обрисати C бит (C=0) после завршетка операције сабирања.



Пример 6.19 Посматрајмо други пример где је $C=0$ пре операције ADCA.

Решење:



У овом примеру резултат производи пренос са позиције бита највеће тежине, тако да ће процесор поставити С бит ($C=1$) по завршетку операције сабирања.

Поставља се питање зашто је неопходно додавати С бит операндима. Разлог је сабирање бројева који се састоје од више бајтова. Сабирање са више бајтова користи се за сабирање бројева који су дужи од једног бајта. Пренос који се формира додавањем бајтова мање тежине додаје се у сабирању бајтова веће тежине.

Одузми акумулатор В од акумулатора А (SBA)

Ова инструкција одузима садржај акумулатора В од садржаја акумулатора А и резултат смешта у акумулатор А. Аритметичко/логичка јединица за одузимање користи метод другог комплемента; то значи да се други комплемент садржаја акумулатора В додаје на садржај акумулатора А. Резултат утиче на N, Z и V битове на исти начин као и претходно описана операција сабирања. На С бит утиче на други начин:

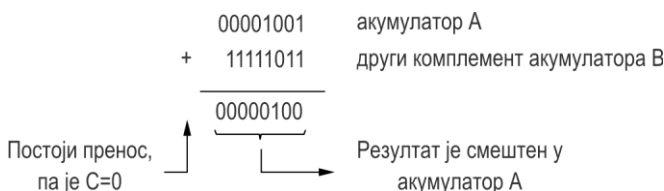
После операције одузимања С бит се комплентира чиме указује да је дошло до позајмљивања.

Када се већи број одузима од мањег броја постојаће позајмљивање, тако да ће процесор поставити С бит ($C=1$). Када се мањи број одузима од већег или једнаког броја, нема позајмљивања, тако да је С бит обрисан ($C=0$). У следећем примеру то ћемо илустровати.

Пример 6.20 Претпоставимо да је $[A] = 00001001_2 = 9_{10}$ и да је $[B] = 00000101_2 = 5_{10}$. Објаснити како ће се SBA операција извршити?

Решење:

SBA операција ће изгледати овако:

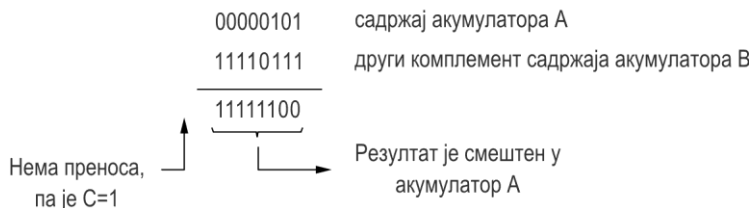


Додавање другог комплемента садржаја акумулатора В на садржај акумулатора А производи бит преноса са позиције бита највеће тежине. Ово се увек дешава када се од већег броја одузима број који је мањи, или једнак броју од кога се одузима. Процесор комплентира (инвертује) С бит како би указао на статус позајмљивања.

Пример 6.21 Претпоставимо да је $[A] = 00000101_2 = 5_{10}$ и да је $[B] = 00001001_2 = 9_{10}$. Објаснити како ће се SBA операција извршити.

Решење:

Сада ће SBA операција изгледати овако:



У овом случају, нема преноса са позиције 8. бита. Ово се увек дешава када је број који се одузима мањи од броја од кога се одузима. Процесор ће поставити C=1 да би указао да је извршено позајмљивање. Позајмљивање се користи код вишебајтног одузимања.

Одузимање садржаја меморије од садржаја акумулатора (SUBA/SUBB)

Ове инструкције се користе да би се одузео податак из меморије од податка из акумулатора. Утицај на N, Z, C и V битове исти је као и код SBA инструкције. Свака од ових инструкција користи неколико начина адресирања, као што је описано у 11. поглављу.

Пример 6.22 *Слика 6.4 је дијаграм тока програма који извршава неколико операција сабирања и одузимања. Написати програм за овај дијаграм тока.*

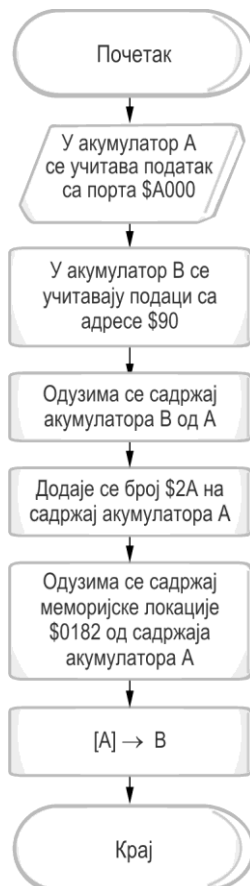
Решење:

Адреса	Лабела	Инструкцијски код	Мнемоник
2000	START	B6 A0 C0	LDAA \$A000
2003		D6 90	LDAB \$90
2005		10	SBA
2006		8B 2A	ADDA #\$2A
2008		B0 01 82	SUBA \$0182
200B		16	TAB
200C	KRAJ	3E	WAI

Испитати пажљиво овај програм и уочити на који начин свакој инструкцији одговара један од блокова у дијаграму тока. Такође проверити врсте адресирања и операционе кодове за сваку инструкцију.

Одузимање садржаја меморијске локације од акумулатора са преносом (SBCA/SBCB)

Ова инструкција одузеће бајт меморије и C бит од једног од акумулатора. SBCB инструкција се користи једино када се извршава вишебајтно одузимање, и C бит представља позајмљивање које генерише одузимање бајтова мање тежине.



Слика 6.4 Дијаграм тока за пример 6.22

Пример 6.23 Објаснити се извршава инструкција `SBCB $0250` ако је садржај акумулатора В $\$28$ ($[B] = 28_{16}$), податак $\$12_{16}$ се налази на локацији $\$0250$ и *С* бит преноса је постављен ($C=1$).

Решење:

Када процесор извршава инструкцију `SBCB $0250`, он ће одузети податак са $\$0250$ и *С* бит од садржаја акумулатора В. Све вредности су хексадецималне.

Децимално прилагођење акумулатора (DAA)

Ова инструкција се користи само после сабирања два BCD кодирана операнда. Резултат се смешта у акумулатор А. Аритметичко/логичка јединица сабира операнде као да су то бинарно кодирани бројеви, и бинарни резултат се смешта у акумулатор А. Инструкција DAA онда се користи за конвертовање резултата из акумулатора А у одговарајућу BCD форму. Инструкција DAA не може се користити са акумулатором В, и не може се користити за одузимање BCD кодираних бројева.

Пример 6.24

Да бисмо илустровали DAA инструкцију претпоставимо:
 $[A]=01000110_2$, BCD кодирано: $[A]_{BCD}=46_{10}$ и $[B]=00100111_2$, BCD кодирано: $[B]_{BCD}=27_{10}$.

Решење:

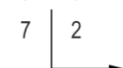
$$\begin{array}{rcl}
 01000110 & \text{акумулатор А} \\
 + 00100111 & \text{акумулатор В} \\
 \hline
 01101101 & \\
 \hline
 \end{array}$$


 Резултат је смештен у акумулатор А

Инструкција ABA се извршава на следећи начин:

Резултат није у BCD формату. Може се конвертовати у BCD формат извршавањем DAA инструкције. Додавањем 0110 на садржај акумулатора А инструкција DAA ће извршити корекцију на следећи начин:

$$\begin{array}{rcl}
 01101101 & \text{садржај акумулатора А после АБА инструкције} \\
 + 0110 & \\
 \hline
 01110100 & \\
 \hline
 \end{array}$$


 Исправан BCD формат смештен у акумулатор А

Пример 6.25 Инструкција DAA се користи после инструкција сабирања. Колико начина постоји и који су они који омогућавају сабирање BCD кодираних операнда?

Решење:

Постоје три могуће инструкцијске секвенце које се могу користити при сабирању операнда који су BCD кодирани:

1. ABA, DAA
2. ADDA, DAA
3. ADCA, DAA

Ови парови инструкција обављају операције додавања BCD кодираних бројева и као резултат дају BCD кодирани број.

Операција негације

Овом операцијом се формира други комплемент. Она се назива *негација* зато што се у случају бројева са знаком од позитивних праве негативни и обрнуто. Постоје три врсте операција негације:

- NEG - други комплемент податка из меморије;
- NEGA - други комплемент садржаја акумулатора A;
- NEGB - други комплемент садржаја акумулатора B.

Пример 6.26

Пошто не постоји инструкција која одузима садржај акумулатора A од садржаја акумулатора B написати инструкцијску секвенцу која би реализовала ову функцију: одузети садржај акумулатора A од садржаја акумулатора B, а резултат сместити у акумулатор B.

Решење:

NEGA ; Други комплемент садржаја акумулатора A
 ABA ; Додати садржају акумулатора B садржај
 акумулатора A
 TAB ; Пренети садржај акумулатора A у акумулатор B

Пошто резултат ABA инструкције остаје у акумулатору A, потребна је TAB инструкција за преношење резултата у акумулатор B.

Операција инкрементирања (увећавање за 1)

Ова операција додаје 1 на операнд. Постоје три типа инструкција инкрементирања чији операнд је 8-битни:

- INC - инкрементирање податка у меморији;
- INCA - инкрементирање садржаја акумулатора A;

- INCB - инкрементирање садржаја акумулатора B.
INC инструкција користи проширено и индексно адресирање.

Постоје три инструкције за инкрементирање 16-битних регистара X и SP:

- INX - инкрементира садржај X регистра;
- INY - инкрементира садржај Y регистра;
- INS - инкрементира садржај показивача стек меморије (SP) .

Инструкције за инкрементирање обично се користе за увећавање садржаја меморијских локација или регистара који се користе као бројачи.

Пример 6.27 *Податак \$FE смештен је на адресу \$F000. Ако се над садржајем ове локације изврше две узастопне инструкције INC \$F000 колики ће бити садржај те локације?*

Решење:

Прва инструкција INC \$F000 додаће 1 на садржај локације \$F000, па се добија \$FF. Следећа INC \$F000 инструкција додаће 1 на \$FF и добиће се \$00. Процесор не води рачуна о преносу који је проузрокован инструкцијом за инкрементирање (погледати 11. поглавље).

Операција декрементирања (смањивања за 1)

Ова операција одузима 1 од операнда. Постоји пет различитих типова инструкција за декрементирање:

- DEC - декрементирање податка у меморији;
- DECA - декрементирање садржаја акумулатора A;
- DECB - декрементирање садржаја акумулатора B;
- DEX - декрементирање садржаја индекс регистра X;
- DEY - декрементирање садржаја индекс регистра Y;
- DES - декрементирање садржаја указивача стек меморије (SP).

Инструкције за декрементирање обично се користе за декрементирање меморијске локације или регистра који користи бројач уназад.

Пример 6.28 *Одредити садржај локације \$F000 после извршавања инструкција CLR \$F000, DEC \$F000.*

Решење:

Резултат CLR \$F000 инструкције је [\$F000]=00000000₂, DEC \$F000

инструкција одузима 1 од [\$F000] и то додавањем другог комплемента броја 00000001:

$$\begin{array}{rcl}
 00000000 & \text{меморијска локација \$F000} \\
 + 11111111 & \text{други комплемент од 00000001} \\
 \hline
 11111111 \\
 \text{FF} & \text{Резултат је смештен у акумулатор B}
 \end{array}$$

Уочимо да би операција одузимања довела до позајмице и постављања С бита (иако нема преноса). За инструкцију декрементирања процесор занемарује позајмљивање, и не мења С бит.

Пример 6.29 Замислимо да имамо програм који надгледа улазни порт да би проверио статус појединих битова. Инструкција INX се користи да би пребројала колико пута посматрани бит има вредност 1. Програм читава податак са улазног порта, проналази да је специфицирани бит 1, и извршава операцију INX. Уколико је на почетку програма $[X]=0000_{16}$, а на крају програма $03C1_{16}$, колико пута је улазни бит имао вредност 1?

Решење:

$$03C1_{16} = 3 \times 16^2 + C \times 16^1 + 1 \times 16^0 = 768 + 192 + 1 = 961_{10}$$

За време извршавања програма индекс регистар X се инкрементира 961_{10} пута.

Инструкције упоређивања

Ова класа инструкција обавља операције одузимања два операнда, али резултат не смешта у неки од регистара. Резултат се користи само да би утицао на регистар услова (CCR). Битови се могу користити да би се оценио релативни однос операнда (тј. $A = B$, $A < B$, $A > B$).

Упореди садржаје акумулатора (CBA)

Ова инструкција одузима садржаје акумулатора B од A, а резултат не смешта у акумулатор A. Резултат утиче само на N, Z, C и V битове регистра услова.

Пример 6.30 Претпоставимо да се у оба акумулатора налазе бројеви без знака. Написати инструкцијску секвенцу која почиње на адреси \$E000 и грана се до \$E06F уколико је садржај акумулатора A већи од садржаја акумулатора B; у супротном програм се зауставља.

Решење:

Адреса	Лабела	Инстр. код	Мнемоник	Коментар
E000	START	11	CBA	; Упоредити акумулаторе
E001		22	BHI \$ E0CF	; Гранај се на \$E06F ако
E002		6C		; је [A] > [B]
E003		3E	WAI	; У супротном заустави се

CBA инструкција одузима садржај акумулатора B од садржаја акумулатора A, али резултат не смешта у акумулатор A; садржаји акумулатора A и B се не мењају. Инструкција BHI проверава да ли је резултат већи од нуле пошто то указује да је [A]>[B]. Уколико је овај услов испуњен, програм се грана на \$E06F.

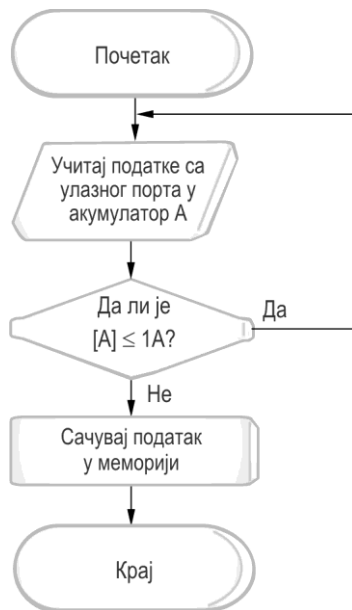
Упоређивање садржаја акумулатора A/B са садржајем меморијске локације (CMPA/CMPB)

Ове инструкције одузимају садржаје акумулатора B или A и садржај меморијске локације. Резултат утиче само на N, Z, C и V битове регистра услова.

Пример 6.31 Напиши инструкцијску секвенцу која континуално тестира величину података на улазном порту. Када податак достигне вредност већу од $26_{10} = 1A_{16}$, смешта се на локацију \$01A2 и програм се зауставља. Улазни порт има адресу \$A00A и подаци су бинарни са знаком. Дијаграм тока је приказан на слици 6.5:

Решење:

START	LDAA \$A00A	; Учитај податке са порта \$A00A у акумулатор A
	CMPA #\$1A	; Упореди садржај акумулатора A са $26_{10} = 1A_{16}$



Слика 6.5 Пример 6.31

BLE START	; Уколико $[A] \leq 1A_{16}$ иди натраг и понови учитавање
STAA \$01A2	; Ако је $[A] > 1A_{16}$ смести податак у меморију
KRAJ	WAI ; Заустављање

Треба уочити како CMPA и BLE инструкције извршавају операцију представљену блоком за одлучивање. Инструкција BLE се користи зато што су подаци бинарни са знаком. CMPA инструкција не мења садржај акумулатора А тако да он још увек садржи податке са улазног порта.

Пример 6.32 *Тастатура је интерфејс ка рачунарском улазном порту, тако да су подаци на улазном порту у било ком тренутку ASCII кодирани за тастер који је притиснут. Када ниједан тастер није притиснут податак је \$FF. Треба написати инструкцијску секвенцу да ради следеће:*

- а) читава податке са тастатуре
- б) уколико ниједан тастер није притиснут, понавља а)
- в) уколико је притиснут RETURN тастер зауставља програм
- г) уколико је било који други тастер притиснут, иде натраг на а) и поново читава тастатуру.

Слика 6.6 је дијаграм тока за задатак који треба обавити. Подаци са тастатуре су учитани у акумулатор В. Први блок за одлучивање тестира податке да ли су једнаки \$FF. Уколико јесу, значи да ниједан од тастера није притиснут.

Програм се грана тако да може поново да учитава податак са тастатуре. У супротном, програм наставља са блоком за одлучивање где тестира да ли су подаци различити од \$0D (ASCII код за RETURN тастер). Уколико јесу, грана се натраг да би поново прочитао податке. У супротном програм наставља са извршавањем следеће инструкције и зауставља се.

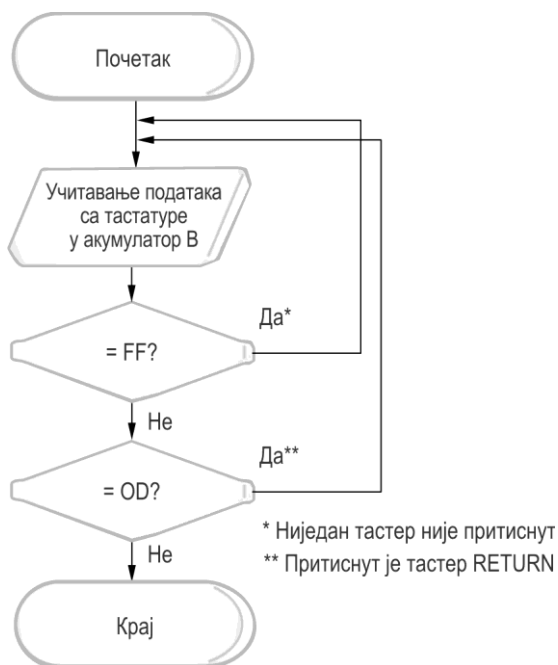
Решење:

START	LDAB \$C090	; Учитај податке у акумулатор В
	CMP #FF	; Упореди са FF (није ниједан тастер притиснут)
	BEQ START	; Уколико је = гранај
	CMPB #\$0D	; Упоредити са 0D, код за RETURN тастер
	BNE START	; Уколико је ≠ гранај се натраг
KRAJ	WAI	; У супротном заустави се

Анализирати пажљиво програм. Уочити како се инструкције поређења и гранања користе за реализацију блокова за одлучивање.

Упоређивање садржаја акумулатора D са садржајем меморијских локација (CPD)

Ова инструкција извршава се на исти начин као CMPA и CMPB, осим што се упоређују два 16-битна броја пошто је акумулатор D 16-битни регистар. Упоређивање се реализује 16-битним одузимањем од садржаја акумулатора D податка који се налази на двама узастопним меморијским локацијама. На пример инструкција CPD \$205E значи да



Слика 6.6 Пример 6.32

ће се упоређивати садржај акумулатора D и податак чији је виши бајт на адреси \$205E а нижи на адреси \$205F.

Тестирање садржаја меморије (TST)

Ова инструкција тестира податак на одређеној меморијској локацији и према томе прилагођава вредност N и Z битова. Посматрајмо следећу инструкцијску секвенцу:

TST \$F700	; Тестира податке на адреси \$F700
BEQ NULA	; Ако је податак 0 гранај се на NULA
BMI NEG	; Ако је податак негативан, гранај се на NEG
NULA LDAA \$F700	
.....	
NEG LDAB \$F701	

Тест инструкција испитује податке на адреси \$F700 (меморија или улазни порт), поставља бит N на вредност бита највеће тежине података и поставља Z=1, ако је податак једнак нули. BEQ инструкција ће произвести гранање једино ако је податак 0 (Z=1). BMI инструкција ће довести до гранања само уколико су подаци негативни (N=1)

Тестирање садржаја акумулатора A и B (TSTA/TSTB)

Ове инструкције обављају TST операцију над подацима у назначеном акумулатору.

Пример 6.33 *Ако је $[A]=D1_{16}$, одредити шта се дешава приликом извршавања инструкцијске секвенце.*

```
TSTB
BMI NEG
BNE NENULA
.....
NEG    STAB $F300
WAI
```

Решење:

$D1_{16}=11010001_2$, TSTB ће поставити N=1 и Z=0. Због тога ће BMI инструкција довести до гранања на програмску целину, која започиње лабелом NEG због чега се неће извршити инструкција BNE.

6.6.3 Инструкције множења и дељења

У инструкцијском скупу MC68HC11 постоји једна инструкција множења и две инструкције дељења. Множење два 8-битна броја као резултат даје 16-битни број.

Целобројно дељење (IDIV) реализује дељење два 16-битна броја, а као резултат добија се 16-битни број и 16-битни остатак. Нецелобројно дељење реализује дељење (FDIV) 16-битног дељеника са 16-битним делиоцем, који је већи број од дељеника. Као резултат добија се 16-битни број (у опсегу 0 до 0,99998) и 16-битни остатак.

Операција	Мнемоник	Имплицитно адресирање
Множење $(A \cdot B) \rightarrow D$	MUL	X
Нецелобројно дељење $(D/X \rightarrow X; r \rightarrow D)$	FDIV	X
Целобројно дељење $(D/X \rightarrow X; r \rightarrow D)$	IDIV	X

6.6.4 Инструкције које реализују логичке операције

Ова група инструкција користи се за реализацију логичких операција Булове алгебре: И (AND), ИЛИ (OR) и ексклузивно ИЛИ (EXOR) - ЕИЛИ и први комплемент.

Функција	Мнемоник	Непосредно	Директно	Проширено	Индексирано (X регистар)	Индексирано (Y регистар)	Имплицитно
$[A] \bullet [M]$	ANDA	X	X	X	X	X	
$[B] \bullet [M]$	ANDB	X	X	X	X	X	
Упоређивање битова A и меморије (<i>Bit(s) Test A with Memory</i>)	BITA	X	X	X	X	X	
Упоређивање битова акумулатора B и податка са меморијске адресе (<i>Bit(s) Test B with Memory</i>)	BITB	X	X	X	X	X	
Први комплемент податка са меморијске адресе (<i>One's Complement Memory Byte</i>)	COM			X	X	X	
Први комплемент садржаја акумулатора A (<i>One's Complement A</i>)	COMA						X
Први комплемент садржаја акумулатора B (<i>One's Complement B</i>)	COMB						X
Ексклузивно ИЛИ садржаја акумулатора A и податка са меморијске адресе (<i>OR A with Memory - Exclusive</i>)	EORA	X	X	X	X	X	
Ексклузивно ИЛИ садржаја акумулатора B и податка са меморијске адресе (<i>OR B with Memory - Exclusive</i>)	EORB	X	X	X	X	X	
ИЛИ садржаја акумулатора A и податка са меморијске адресе (<i>OR A with Memory- Inclusive</i>)	ORAA	X	X	X	X	X	
ИЛИ садржаја акумулатора B и податка са меморијске адресе (<i>OR B with Memory-Inclusive</i>)	ORAB	X	X	X	X	X	

Табела 6.3

6. Програмирање процесора MC68HC11

У овом одељку биће детаљно анализирана свака од ових инструкција.

1	0	1	0	1	0	1	0	Акумулатор А
1	1	0	1	0	0	1	1	Операнд (маска)
1	0	0	0	0	0	1	0	Резултат И операције
1	1	1	1	1	0	1	1	Резултат ИЛИ операције
0	1	1	1	1	0	0	1	Резултат ЕИЛИ операције

Резултат ИЛИ операције } смештено у
Резултат ЕИЛИ операције } акумулатор А

Важно је уочити да се логичке операције извршавају посебно над сваком битском позицијом и независно од осталих битских позиција. На пример, 8. бит¹ (b_7) акумулатора је у И операцији са 8. битом операнда и даје 8. бит резултата; такође 7. бит (b_6) акумулатора А је у И операцији са 7. битом операнда и добија се 7. бит резултата, итд. за све остале битове.

Извршавање логичких инструкција има утицаја на Z и N битове. Уколико је резултат 00000000, биће постављен Z бит; у супротном он ће бити обрисан. Уколико је јединица на позицији 8. бита (b_7) резултата, биће постављен N бит ($N=1$), у супротном ће бити обрисан ($N=0$).

И маскирање

Често се И (AND) инструкције користе за селективно брисање одређених битова податка, при чему преостали битови у податку остају непромењени. Ово се може урадити преко И операције податка са бројем који се назива маска. Маске програмер користи да би одабрао битове које жели да обрише. Ово је илустровано на следећи начин:

Битови података b_7 до b_0 , могу да буду било која комбинација јединица и нула. Бајт маске има 1 на свакој битској позицији осим 8. и 3. позиције (b_7 и b_2) на којој је 0. Резултат који се добија И операцијом податка и маске има нулу на 8. и 3. позицији (b_7 и b_2), док остали битови

¹ Треба имати на уму: бит највеће тежине (8. бит) обележава се са b_7 ; бит најмање тежине (1. бит) обележава се са b_0 .

б ₇	б ₆	б ₅	б ₄	б ₃	б ₂	б ₁	б ₀	Податак
0	1	1	1	1	0	1	1	Маска за брисање битова б ₂ и б ₇
0	б ₆	б ₅	б ₄	б ₃	0	б ₁	б ₀	Резултат И операције

имају исту вредност као и одговарајући битови бајта податка¹. Другим речима: обрисани су други и седми бит бајта податка, а остали битови су непромењени. Значи да се било који бит податка може обрисати када се постави 0 на одговарајућу позицију у бајту маске. Овај процес обично се назива И-маскирање. Техника маскирања обично се користи за изоловање вредности једног бита у речи податка. Реч податка може бити статусна реч учитана са улазног уређаја, а одговарајући бит статусне речи може бити употребљен да укаже процесору коју инструкцијску секвенцу треба да извршава. Ово је илустровано на следећи начин:

б ₇	б ₆	б ₅	б ₄	б ₃	б ₂	б ₁	б ₀	Статусна реч из улазног модула
0	0	0	1	0	0	0	0	Маска за изоловање бита б ₄
0	0	0	б ₄	0	0	0	0	Резултат И операције

У бајту маске бит б₄ је 1, а сви остали битови су 0. Као резултат И операције добија се број чији су сви битови 0 осим бита б₄. Уколико је вредност бита б₄ једнака 0, резултат ће бити 0, а Z бит у регистру услова ће бити постављен (Z=1). Уколико је вредност бита б₄ једнака 1, резултат ће бити различит од 0, а Z бит у регистру услова ће бити обрисан (Z=0). Тестирањем Z бита програм може да одреди статус б₄ и промени редослед извршавања инструкција.

ИЛИ маскирање

Инструкција ORA може се користити за селективно постављање одређених битова бајта податка, без утицаја на остале битове. Користи

¹ Особина И операције: шта год је у И операцији са 0 је 0, и шта год је у И операцији са 1 задржава своју вредност.

6. Програмирање процесора MC68HC11

се маска од једног бајта који има 1 само на оним позицијама које треба да буду постављене на 1. Ово је илустровано на следећи начин:

б ₇	б ₆	б ₅	б ₄	б ₃	б ₂	б ₁	б ₀	Податак
0	1	1	1	0	0	0	0	Маска за постављање битова б ₄ , б ₅ и б ₆
б ₇	1	1	1	б ₃	б ₂	б ₁	б ₀	Резултат ИЛИ операције

Маска, коју користимо у овом примеру, садржи 1 на месту битова б₄, б₅ и б₆. Резултат ИЛИ операције између маске и бајта податка је бајт са 1 на местима битова б₄, б₅ и б₆ независно од вредности ових битова у податку. На остале битове податка маска нема утицаја пошто су остали битови на 0 и немају утицаја¹.

ЕИЛИ-селективно инвертовање

Инструкција ЕИЛИ обично се користи за селективно инвертовање специфичних битова података. Ова операција нема утицаја на остале битове а може се илустровати на следећи начин:

б ₇	б ₆	б ₅	б ₄	б ₃	б ₂	б ₁	б ₀	Податак
0	0	1	0	0	0	0	0	ЕИЛИ маска за инвертовање б ₅
б ₇	б ₆	б ₅	б ₄	б ₃	б ₂	б ₁	б ₀	Резултат ЕИЛИ операције

Резултат показује да је бит б₅ податка инвертован (1 на б₅ позицији маске), док су други битови бајта податка непромењени (0 на осталим позицијама маске). Таблица истинитости за ЕИЛИ (EOR) логичку операцију је:

¹ Особина ИЛИ операције: било шта у операцији ИЛИ са 1 је 1; било шта у операцији ИЛИ са 0 је непромењено.

Бит податка (D)	Бит маске (M)	Резултат $D \oplus M$
0	0	0
1	0	1
0	1	1
1	1	0

Једна од примена ЕИЛИ операције је када микрорачунар чита са улазног порта инвертоване податке. Проблем се може решити постављањем инвертора са три стања између излаза уређаја и магистрале података. Мање скуп начин је да се учитају подаци у рачунар и затим употреби ЕИЛИ инструкција којом се инвертују подаци. Ово се једноставно постиже коришћењем следеће програмске секвенце:

LDAA \$FD00 ; Учитавање података са адресе \$FD00
 EORA #\$FF ; Инвертовање садржаја акумулатора

Маска (величине једног бајта) $FF_{16}=11111111_2$ користи се да инвертује сваки бит акумулатора.

Пример 6.34 Претпоставимо да је $[A]=5516$ и $[B]=3916$. За сваку од следећих инструкција одредити резултат и вредности N и Z битова:

- ANDAA #\$C8
- ORAA #\$AA
- EORB #\$39

Решење:

- Инструкција извршава операцију И између бајта података \$C8 и $[A]=55_{16}$. Као резултат добија се $[A]=40_{16}=01000000_2$. Бит Z ће бити 0 пошто резултат није 0; N бит ће бити 0 пошто је 8. бит (6.) резултата 0.
- Инструкција извршава операцију ИЛИ између бајта података \$AA и $[A]=55_{16}$. Као резултат се добија $[A]=FF_{16}=11111111_2$. Бит Z је 0 а N бит је 1.
- Инструкција извршава операцију ЕИЛИ између бајта података \$39 и $[B]=39_{16}$. Као резултат се добија $[B]=00_{16}$. Бит Z је постављен на 1 пошто је резултат 0; N бит је постављен на 0.

Пример 6.35 Написати део програма који узима бајт из меморијске локације \$F900, брише бит најмање тежине (LSB), поставља бит највеће тежине (MSB), и враћа у меморију без утицаја на друге битове.

START	LDA \$F900	; Упиши податке са адресе \$F900 у акумулатор А
	ANDA #\$FE	; И маска за бит најмање тежине (LSB)
	ORAA #\$80	; ИЛИ маска за постављање бита највеће тежине (MSB)
	STAA \$F900	; Смести податке у меморију
KRAJ	WAI	; Заустављање

Решење:

Прво треба уписати податак из меморије у један од акумулатора (у овом случају А), затим се инструкцијом ANDA #\$FE изврши И операција - маскирање акумулатора са 11111110₂ да би се обрисао бит најмање тежине без утицаја на друге битове. Трећа инструкција врши ИЛИ операцију - маскирање акумулатора са 10000000₂ да би се поставио бит највеће тежине; и четврта инструкција смешта резултат у меморију.

Операција комплементирања (COM)

Операција ће комплементирати бајт података (тј. комплементира, инвертује сваки бит). Три типа инструкција за комплементирање су:

- COM - комплементирање бајта меморије;
- COMA - комплементирање садржаја акумулатора А;
- COMB - комплементирање садржаја акумулатора В.

COM инструкција користи и проширени и индексни начин адресирања.

Пример 6.36 Садржај акумулатора А је \$AA. Одредити његов садржај пошто се изврше следеће инструкције: ADDA #\$55, COMA.

Решење:

После извршавања инструкције ADDA #\$55 садржај акумулатора А

је $[A] = FF_{16} = 11111111_2$. Инструкција СОМА инвертује сваки бит. Као резултат се добија $[A] = 00000000_2 = 0$

6.6.5 Испитивање вредности¹ и рад са битовима²

Ова група инструкција користи се за рад са операндима који су величине једног бита, али ове инструкције могу да раде са било којом комбинацијом бита у 8-битној локацији у 64kB меморијском простору. Инструкције за испитивање бита (BITA и BITB) обављају И (AND) операцију. Вредност операнда се не мења, а резултат операције утиче само на садржај регистра услова. Инструкције BSET и BCLR учитавају операнд, раде са одабраним битовима операнда и враћају резултат натраг на адресу операнда. Када се инструкције читај-модификуј-упиши³ као што су BSET и BCLR користе за рад са улазно/излазним уређајима и управљачким регистрима физичка локација са које се читава није увек иста као локација у коју се уписује.

Функција	Мнемоник	Непосредно	Директно	Проширено	Индексирано (X регистар)	Индексирано (Y регистар)
Испитивање бита (бита) акумулатора А и броја са меморијске адресе (<i>Bit(s) Test A with Memory</i>)	BITA	X	X	X	X	X
Испитивање бита (бита) акумулатора В и броја са меморијске адресе (<i>Bit(s) Test B with Memory</i>)	BITB	X	X	X	X	X
Обрисати бит (бита) на меморијској адреси (<i>Clear Bit(s) in Memory</i>)	BCLR		X		X	X
Поставити битове на меморијској адреси (<i>Set Bit(s) in Memory</i>)	BSET		X		X	X
Гранање уколико је (су) бит (битови) обрисани (<i>Branch if Bit(s) Clear</i>)	BRCLR		X		X	X
Гранање уколико је (су) бит (битови) обрисани (<i>Branch if Bit(s) Set</i>)	BRSET		X		X	X

Табела 6.4

¹ Data Testing

² Bit Manipulation

³ Read-modify-write

Ово су специјалне инструкције за тестирање података које могу да тестирају један или више битова операнда без утицаја на сам операнд.

Испитивање битова BITA/BITB

Ова инструкција извршава бит по бит И операцију специфицираног бајта податка са одређеним акумулатором, али не мења садржај акумулатора. Резултат једино утиче на N и Z битове. Тако је могуће да се користи И маскирање за изоловање једног или више битова без промене садржаја акумулатора. На пример посматрајмо следећу инструкцијску секвенцу:

BITA#\$08	; Изолије се бит
BEQ\$12C0	; Уколико је 0 гранај се на \$12C0
BITA#\$04	; Изолије се бит b_2
BNE\$12D0	; Уколико је 1 гранај се на \$12D0

Прва инструкција реализује И операцију садржаја акумулатора А са 00001000_2 . Резултат ће бити 00000000_2 једино ако је 4. бит (b_3) акумулатора А једнак 0. Друга инструкција доводи до грања на адресу \$12C0 само ако је 4. бит једнак 0. Трећа и четврта инструкција ће тестирати 3. бит (b_2) акумулатора А па уколико је његова вредност 1 гранаће се на \$12D0. Уочимо да садржај акумулатора А остаје непромењен после прве BITA инструкције, тако да друга BITA инструкција оперише истим подацима.

6.6.6 Инструкције померања и ротирања

Инструкције померања и ротирања процесора MC68HC11 укључују бит преноса (C) регистра услова (CCR) као додаток 8-битном или 16-битном операнду инструкције. Ова карактеристика омогућава једноставно проширивање употребе ових инструкција и на операнда који се састоји од више речи¹. Постављање и брисања бита преноса пре извршавања инструкција померања и ротирања програмеру омогућава да једноставно контролише шта ће бити убачено у операнд.

Инструкција аритметичког померања удесно (ASR) води рачуна о вредности бита највеће тежине. Тиме инструкција води рачуна о

¹ Multiple-word operands

оригиналној вредности бита највеће тежине што помаже у раду са бројевима представљеним у другом комплементу (бројевима са знаком).

Функција	Мнемоник	Непосредно	Директно	Проширено	Индексирано (X регистар)	Индексирано (Y регистар)	Имплицитно
Аритметичко померање улево податка са меморијске адресе (<i>Arithmetic Shift Left Memory</i>)	ASL			X	X	X	
Аритметичко померање улево садржаја акумулатора A (<i>Arithmetic Shift Left A</i>)	ASLA						X
Аритметичко померање улево садржаја акумулатора B (<i>Arithmetic Shift Left B</i>)	ASLB						X
Аритметичко померање улево садржаја акумулатора D (<i>Arithmetic Shift Left Double</i>)	ASLD						X
Аритметичко померање удесно податка са меморијске адресе (<i>Arithmetic Shift Right Memory</i>)	ASR			X	X	X	
Аритметичко померање удесно садржаја акумулатора A (<i>Arithmetic Shift Right A</i>)	ASRA						X
Аритметичко померање удесно садржаја акумулатора B (<i>Arithmetic Shift Right B</i>)	ASRB						X
Логичко померање податка са меморијске адресе улево (<i>Logical Shift Left Memory</i>)	LSL			X	X	X	
Логичко померање акумулатора A улево (<i>Logical Shift Left A</i>)	LSLA						X
Логичко померање садржаја акумулатора B улево (<i>Logical Shift Left B</i>)	LSLB						X
Логичко померање садржаја акумулатора D улево (<i>Logical Shift Left Double</i>)	LSLD						X
Логичко померање податка са меморијске адресе удесно (<i>Logical Shift Right Memory</i>)	LSR			X	X	X	
Логичко померање садржаја акумулатора A удесно (<i>Logical Shift Right A</i>)	LSRA						X
Логичко померање садржаја акумулатора B удесно (<i>Logical Shift Right B</i>)	LSRB						X
Логичко померање садржаја акумулатора D удесно (<i>Logical Shift Right D</i>)	LSRD						X
Ротирање податка са меморијске адресе улево (<i>Rotate Left Memory</i>)	ROL			X	X	X	

Функција	Мнемоник	Непосредно	Директно	Проширено	Индексирано (X регистар)	Индексирано (Y регистар)	Имплицитно
Ротирање садржаја акумулатора А улево (<i>Rotate Left A</i>)	ROLA						X
Ротирање садржаја акумулатора В улево (<i>Rotate Left B</i>)	ROLB						X
Ротирање податка са меморијске адресе удесно (<i>Rotate Right Memory</i>)	ROR			X	X	X	
Ротирање садржаја акумулатора А удесно (<i>Rotate Right A</i>)	RORA						X
Ротирање садржаја акумулатора А удесно (<i>Rotate Right B</i>)	RORB						X

Табела 6.5

Инструкције за логичко померање улево стављене су у заграду пошто нема разлике између аритметичког и логичког померања удесно. Оба мнемоника асемблер посматра на исти начин: нема разлике између логичког и аритметичког померања улево. Међутим, сматра се да је за програмере једноставније да постоје две врсте мнемоника.

Пример 6.37 Ако је $[A]=AA_{16}$, $[B]=F9_{16}$, $[\$F000]=51_{16}$ и $C=1$. За сваку од датих инструкција одредити операнд и C бит после извршења инструкције.

- а) ASL $\$F000$
- б) ASRA
- в) LSRA
- г) ROL $\$F000$
- д) RORB

Решење:

- а) ASL $\$F000$ извршава операцију аритметичког померања улево садржаја меморијске локације $\$F000$. Операција се одвија на следећи начин:

Уочимо да је сваки бит операнда померен улево, са битом b_7 помереним у C бит и 0 убаченом у бит b_7 . Тако су $[\$F000]=$

C	б ₇	б ₆	б ₅	б ₄	б ₃	б ₂	б ₁	б ₀	
1	0	1	0	1	0	0	0	1	[\$F000] пре померања
0	1	0	1	0	0	0	1	0	[\$F000] после померања

$10100010_2 = A2_{16}$ и $C=0$ после операције померања улево. У извршавању инструкције процесор учитава операнд у аритметичко логичку јединицу, извршава операцију померања операнда, и затим га смешта натраг у меморију.

- б) ASRA оперише над садржајем акумулатора А. Операција се извршава овако:

Уочимо да се сваки бит операнда помера десно. Бит 0 се помера у С бит, а бит б₇ остаје непромењен. Тако су после извршавања инструкције $[A] = 11010101_2 = D5_{16}$ и $C=0$.

	б ₇	б ₆	б ₅	б ₄	б ₃	б ₂	б ₁	б ₀	C
[A] пре померања	1	0	1	0	1	0	1	0	1
[A] после померања	1	1	0	1	0	1	0	1	0

- в) LSRA се извршава над акумулатором А. Инструкција се извршава на следећи начин:

	б ₇	б ₆	б ₅	б ₄	б ₃	б ₂	б ₁	б ₀	C
[A] пре померања	1	0	1	0	1	0	1	0	1
[A] после померања	0	1	0	1	1	0	1	1	0

Уочимо да се сваки бит помера удесно. Бит б₀ се помера у С бит а 0 се убацује на место бита б₇. Тако је по извршавању инструкције $[A] = 01010101_2 = 55_{16}$ и $C=0$.

- г) ROL \$F000 се извршава над [\$F000].

С	б ₇	б ₆	б ₅	б ₄	б ₃	б ₂	б ₁	б ₀	
1	0	1	0	1	0	0	0	1	[\$F000] пре ротације
0	1	0	1	0	0	0	1	1	[\$F000] после ротације

Уочимо да се сваки бит помера улево, са С битом који се ротира у б₀ бит и б₇ битом који се помера у С бит. Тако је после операције ротирања $[\$F000] = 10100011_2 = A3_{16}$.

д) RORB се извршава над акумулатором В.

Сваки бит се помера удесно, С бит се ротира у бит б₇ и бит б₀ се помера у С бит. Тако је после операције $[B] = 11111100_2 = FC_{16}$ и $C=1$.

б ₇	б ₆	б ₅	б ₄	б ₃	б ₂	б ₁	б ₀	Податак
0	0	1	0	0	0	0	0	ЕИЛИ маска за инвертовање б ₅
б ₇	б ₆	$\bar{b}_5$	б ₄	б ₃	б ₂	б ₁	б ₀	Резултат ЕИЛИ операције

Пример 6.38 Конверзија BCD кодираног податка у ASCII кодирану податак.

Решење:

Рачунар често шаље податке ка излазном уређају који их штампа или приказује. Уобичајени излазни уређаји су видео монитори и штампачи. Периферни уређаји захтевају да су подаци представљени у ASCII коду. Уколико рачунар треба да пошаље ка штампачу неки знак (број, слово, симбол итд.) то је обично ASCII код тог знака. Посматрајмо типичну примену; податке из рачунарске меморије треба послати ка штампачу. Сваки бајт из меморије састоји се од две BCD децимално кодиране цифре. На пример децимални број 64 (64_{10}) смештен је као 01100100_2 , што је децимални код тог броја. Два децимална броја смештена у једну меморијску реч називају се паковани BCD.

У табели 6.6 види се да се сваки 8-битни ASCII код децималне цифре састоји од битова 0011 на месту четири бита највеће тежине, иза којих је BCD кодирана цифра.

Децимална цифра	ASCII код децималне цифре
0	0011 0000
1	0011 0001
2	0011 0010
3	0011 0011
4	0011 0100
5	0011 0101
6	0011 0110
7	0011 0111
8	0011 1000
9	0011 1001

Табела 6.6

Да би послао ASCII знак ка штампачу, рачунар прво треба да распакује и конвертује BCD код сваке цифре у њен 8-битни ASCII код. На пример BCD „спакована“ реч је $01100100_2 = 64_{10}$ и рачунар треба да направи два бајта: један 00110110_2 (ASCII код за 6) и други 00110100_2 (ASCII код за 4). Слика 6.7 је дијаграм тока програма који извршава ову операцију. У даљем тексту корак по корак биће представљене све операције које је потребно обавити за BCD спаковану реч 01100100_2 .

1. Напуни акумулатор А спакованим BCD податком

$$[A] = 01100100_2$$

2. Извршити четири померања удесно акумулатора А, са нулом убаченом у акумулатор с леве стране. Ово се може извршити са четири узастопне LSRA инструкције. Резултат је:

$$[A] = 00000110_2$$

3. BCD код за 6 (цифру веће тежине децималног броја), сада је на правој позицији ASCII кода.
4. У акумулатору А су битови b_4 и b_5 операцијом ИЛИ са маском 00110000_2 постављени на 1. Резултат ове операције је:



Слика 6.7 Дијаграм тока за конвертовање спакованог BCD податка у ASCII кодирани податак

$$[A] = 00110110_2$$

што је ASCII код за број 6₁₀.

5. Извршавањем STAA операције процесор шаље садржај акумулатора А ка излазном порту за који је везан штампач.
6. Поново се учитава спакован BCD податак у акумулатор А да би се направио ASCII код друге цифре:

$$[A] = 01100100_2$$

7. Да би се обрисали битови највеће тежине извршава се И операција садржаја акумулатора А ($[A]=01100100_2$) и маске 00001111₂, тако да се добија:

$$[A] = 01100100_2$$

8. Извршава се ИЛИ операција садржаја акумулатора А са маском 00110000₂, и добија се:

$$[A] = 001100100_2$$

што представља ASCII код за број 4₁₀.

9. Процесор шаље садржај акумулатора ка излазном порту за који је везан штампач.

START	LDAA \$0300
	LSRA
	LSRA
	LSRA
	LSRA
	ORA #\$30
	STAA \$F800
	LDAA \$0300
	ANDA #\$0F
	ORA #\$30
	STAA \$F800
KRAJ	WAI

6.7 ИНСТРУКЦИЈЕ ЗА РАД СА РЕГИСТРОМ УСЛОВА

Ове инструкције омогућавају програмеру рад (манипулацију) са битовима регистра услова.

Операција	Мнемоник	Имплицитно адресирање
Обриши С бит (<i>Clear Carry Bit</i>)	CLC	X
Обриши бит за маскирање прекида (<i>Clear Interrupt Mask Bit</i>)	CLI	X
Обриши бит премашења (<i>Clear Overflow Bit</i>)	CLV	X
Постави С бит (<i>Set Carry Bit</i>)	SEC	X
Постави бит за маскирање прекида (<i>Set Interrupt Mask Bit</i>)	SEI	X
Постави бит премашења (<i>Set Overflow Bit</i>)	SEV	X
Премештање садржаја акумулатора А у регистар услова (<i>Transfer A to CCR</i>)	TAP	X
Премештање садржаја регистра услова у акумулатор А (<i>Transfer CCR to A</i>)	TPA	X

Табела 6.7

На први поглед могло би се очекивати да постоје инструкције за постављање и брисање сваког од осам битова регистра услова. У ствари постоје инструкције само за три од осам битова (C, I и V). Детаљнија анализа показује да постоје добри разлози зашто не постоје инструкције за постављање и брисање преосталих пет битова. Ситуација са битом S¹ је специфична пошто је његова намена да спречи коришћење STOP инструкције за оне који је виде као „нежељену“ функцију у својој апликацији.

Обезбеђивање инструкције за постављање и брисање овог бита довело би до STOP дозволе када то није пожељно или STOP забране када то није потребно. Инструкција TAP обезбеђује начин за промену S бита али и онемогућава његову нежељену измену, пошто вредност акумулатора А у тренутку извршавања TAP инструкције одређује да ли ће се (или неће) вредност S бита променити. Бит X којим се забрањује извршавање прекида XIRQ је други специфичан случај. Овај бит се дефинише као бит чију промену вредности од нуле на један није дозвољено софтверски мењати. Чињеница

¹ Stop disable – забрана заустављања

је да је та промена забрањена и хардверској логици. Ова особеност објашњава зашто не постоји потреба за инструкцијом која поставља Х бит.

Аргументи слични онима који су изнети за S бит јесу да је TAP инструкција боља од инструкције за брисање Х бита, пошто је мало вероватно да би бит Х био обрисан пре него што би то програмер намеравао.

Бит половине преноса (H) није могуће ни поставити ни обрисати. Он се аутоматски поставља после операција сабирања и одузимања. Уколико су операнди операција сабирања и одузимања BCD кодирани бројеви онда се користи инструкција DAA да би резултат превела у BCD кодирани број. Пошто се H бит не користи као услов за било коју од инструкција гранања није потребно ни да постоје инструкције за његово постављање или брисање.

За разлику од битова S, X и H било би корисно да се N и Z битови могу једноставно поставити или обрисати. На пример инструкција за брисање акумулатора A и B (CLRB и CLRB) обрисаће N бит и поставити Z бит у регистру услова. Други пример је инструкција LDAA #\$80 (пуњење акумулатора) која такође доводи до постављања N бита и брисања Z бита.

Пошто постоји велики број једноставних инструкција које могу да поставе и обришу N и Z битове није неопходно да се обезбеди посебна инструкција за ту намену.

6.8 ИНСТРУКЦИЈЕ КОЈЕ УПРАВЉАЈУ ИЗВРШАВАЊЕМ ПРОГРАМА

Ова група инструкција користи се за управљање тока (извршавања) програма, а не за рад са подацима. Подељена је у пет група:

1. инструкције гранања¹,
2. инструкције скокова²,
3. позиви потпрограма и повратак из потпрограма³,
4. вођење рачуна о прекидима⁴,
5. додатне, не класификоване⁵.

¹ Branches

² Jumps

³ Subroutine calls and returns

⁴ Interrupt handling

⁵ Miscellaneous

6.8.1 Инструкције гранања

Инструкције за условно гранање дају могућност процесору да промени редослед извршавања програма само у зависности од вредности бита (битова) регистра услова. Ова класа инструкција омогућава рачунару способност одлучивања у складу са задатим алгоритмом. Сваки блок за одлучивање у дијаграму тока одговара једној од инструкција за условно гранање које су дате у табели која следи.

Функција	Мнемоник	Релативно	Директно	Индексирано (X регистар)	Индексирано (Y регистар)	Коментар
Гранај се ако је обрисан C бит (C = 0) (Branch if Carry Clear)	BCC	X				C = 0?
Гранај се ако је постављен C бит (C = 1) (Branch if Carry Set)	BCS	X				C = 1?
Гранај се ако је резултат једнак нули (Branch if Equal Zero)	BEQ	X				Z = 1?
Гранај се ако је резултат већи или једнак нули (са знаком) (Branch if Greater Than or Equal)	BGE	X				Signed ≥
Гранај се ако је резултат већи од нуле (са знаком) (Branch if Greater Than)	BGT	X				Signed >
Гранај се ако је резултат већи од нуле (без знака) (Branch if Higher)	BHI	X				Unsigned >
Гранај се ако је број већи или исти (Branch if Higher or Same)	BHS	X				Unsigned ≥
Резултат мањи или једнак нули (са знаком) (Branch if Less Than or Equal)	BLE	X				Signed ≥
Гранај се ако је број мањи (Branch if Lower)	BLO	X				Unsigned <
Резултат мањи или једнак нули (без знака) (Branch if Lower or Same)	BLS	X				Unsigned ≥
Резултат мањи од нуле (са знаком) (Branch if Less Than)	BLT	X				Signed <
Резултат негативан (Branch if Minus)	BMI	X				N = 1?

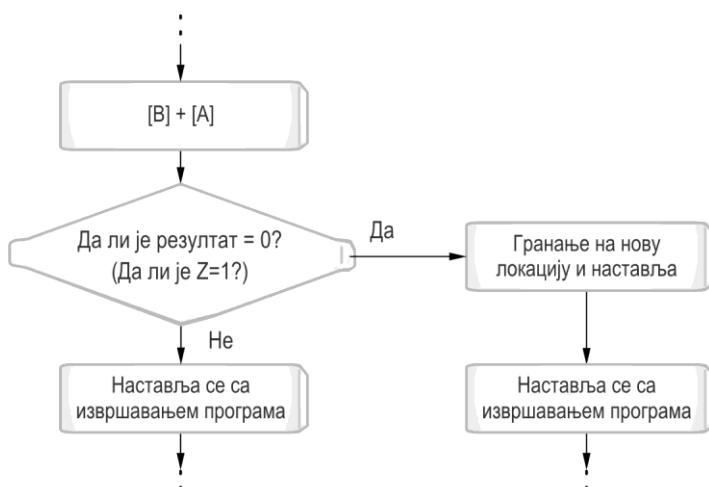
Функција	Мнемоник	Релативно	Директно	Индексирано (X регистар)	Индексирано (Y регистар)	Коментар
Резултат није једнак нули (<i>Branch if Not Equal</i>)	BNE	X				Z = 0?
Резултат позитиван (<i>Branch if Plus</i>)	BPL	X				N = 0?
Гранај се ако су битови бајта у меморији обрисани (<i>Branch if Bit(s) Clear in Memory Byte</i>)	BRCLR		X	X	X	Bit Manipulation
Никда се не гранај (<i>Branch Never</i>)	BRN	X				3-cycle NOP
Гранај се ако су битови бајта у меморији постављени (<i>Branch if Bit(s) Set in Memory Byte</i>)	BRSET		X	X	X	Bit Manipulation
Гранај се ако је бит премашења обрисан (V = 0) (<i>Branch if Overflow Clear</i>)	BVC	X				V = 0?
Гранај се ако је бит премашења постављен (V = 1) (<i>Branch if Overflow Set</i>)	BVS	X				V = 1?

Табела 6.8

Инструкције са условним гранањем извршавају се на следећи начин:

1. Процесор узима операциони код на основу кога одређује који бит из регистра услова треба да проверава.
2. Примери битова чију вредност проверава су:
 - а) Да ли је резултат једнак нули (тј. да ли је Z=1)?
 - б) Да ли је резултат негативан (тј. N=1)?
 - в) Да ли је постоји пренос (тј. C=1)?
3. Уколико је постављени услов испуњен садржај програмског бројача се мења и добија нову адресу. Извршавање програма се наставља са нове адресе.
4. Уколико услов није испуњен процесор прелази на извршавање (узима) следећу инструкцију у секвенци инструкција.

Делимични дијаграм тока на слици 6.8 илуструје описане операције. Ромбоид је блок (модул) који се користи за све инструкције са условним гранањем. На овој слици процесор тестира „да ли је резултат једнак 0”



Слика 6.8 Дијаграм тока који илуструје операцију условног гранања тестирајући да ли је постављен бит Z ($Z=1$) у регистру услова. Уколико је $Z=1$ наставља се са извршавањем програма са нове адресне локације, а уколико је $Z=0$ наставља се нормалан ток извршавања програма.

Релативно адресирање

У одељку 3.12 описана је инструкција гранања и релативно адресирање. Подсетимо се да се инструкције за условно гранање састоје од операционог кода и помераја¹. Померај је једнобајтни бинарни број са знаком, који се додаје програмском бројачу и тако добија нова адреса за гранање. Треба имати на уму да у тренутку извршавања инструкције за условно гранање садржај програмског бројача² указује на адресу прве следеће инструкције иза инструкције условног гранања.

Посматрајмо следећу инструкцијску секвенцу:

Адреса	Лабела	Инстр. код	Мнемоник	Коментар
0250	START	1B	ABA	; Сабери акумулаторе А и В
0251		27	BEQ \$027D	; Гранај се на \$027D
0252		2A		; ако је резултат 0

¹ Користе се и термини офсет или одступање.

² Користе се и термини: текући садржај програмског бројача, тренутни садржај програмског бројача.

Адреса	Лабела	Инстр. код	Мнемоник	Коментар
0253		97	STAA \$7A	; У другом случају
0254		7A		; стави резултат на адресу \$007A

Инструкција BEQ налази се на адресама \$0251-\$0252. Састоји се од операционог кода (\$27) и помераја (\$2A). Инструкција BEQ проузрокује да процесор проверава да ли је резултат нула (провером да ли је $Z=1$). Уколико резултат јесте нула процесор ће наставити са извршавањем програма почев од адресе \$027D (грانا се на адресу \$027D):

$$\begin{array}{rcl}
 \text{Тренутни садржај програмског бројача:} & & 0253_{16} \\
 & \text{Померај:} & + 002A_{16} \\
 \text{Нови садржај програмског бројача:} & & \hline
 & & 027D_{16}
 \end{array}$$

Уочимо да је текући садржај програмског бројача адреса на којој се налази операциони код прве инструкције иза инструкција BEQ; то је адреса са које процесор узима следећи операциони код уколико не дође до гранања.

Опсег помераја

Померај (одступање) је 8-битни бинарни број са знаком. Када је бит највеће тежине 0, померај је позитиван број у опсегу од $00000000_2 = 00_{16} = 0_{10}$ до $01111111_2 = 7F_{16} = +127_{10}$. Позитивно одступање, које се додаје на програмски бројач, може да обезбеди гранање унапред највише до $7F_{16} = 127_{10}$ адреса. Када је бит највеће тежине једнак 1, померај је негативни број који је у опсегу $11111111_2 = FF_{16} = -1_{10}$ до $10000000_2 = 80_{16} = -128_{10}$. Негативно одступање, када се додаје на програмски бројач, доводи до гранања унатраг, највише до $80_{16} = 128_{10}$ адреса. Видимо да је опсег гранања од -128_{10} до $+127_{10}$.

У програмској секвенци коју смо анализирали, \$2A је позитиван померај пошто је $2A_{16} = 00101010_2 = +42_{10}$. Претпоставимо да је померај $7F_{16}$. Нова адреса ће бити:

$$\begin{array}{rcl}
 \text{Тренутни садржај програмског бројача:} & & 0253_{16} \\
 & \text{Померај:} & + 007F_{16} \\
 \text{Нови садржај програмског бројача:} & & \hline
 & & 02D2_{16}
 \end{array}$$

6. Програмирање процесора MC68HC11

У овом случају процесор ће наставити са извршавањем програма са адресе \$02D2 (грана се унапред) што представља максималну вредност позитивног помераја.

Посматрајмо сада померај $FD_{16}=11111101_2=-3_{10}$. Када је померај негативан процесор га додаје на тренутни садржај програмског бројача на следећи начин:

$$\begin{array}{rcl} \text{Тренутни садржај програмског бројача:} & & 0253_{16} \\ \text{Померај:} & + & FFFD_{16} \\ \hline \text{Нови садржај програмског бројача:} & & \uparrow 0250_{16} \\ & \nearrow & \text{Занемарити} \end{array}$$

Уочимо да процесор додаје број \$FFFD на текући садржај програмског бројача. Сабирање је 16-битно, тако да је број \$FFFD 16-битна репрезентација броја -3, а пренос се занемарује. Резултат указује да ће процесор наставити са извршавањем програма са адресе \$0250 (гранати се унатраг до адресе \$0250).

Када дође до гранања процедура за одређивање нове адресе је следећа:

1. Позитивно одступање. Додати одступање на текући садржај програмског бројача. То је адреса операционог кода инструкције која се налази одмах иза бајта помераја.
2. Негативно одступање. Додати вредност бајта помераја проширеног са \$FF на текући садржај програмског бројача.

Пример 6.39 Извршавање инструкција BCS доводи до гранања на адресу \$580E ако је C бит постављен. Проверити да ли се и на коју адресу грана програм.

Адреса	Лабела	Инстр. код	Мнемоник	Коментар
57A0	START	56	RORB	; Помери садржај акумулатора А улево
57A1		25	BCS \$580E	; Гранај се на адресу \$580E
57A2		6B		; ако је C=1
57A3	KRAJ	3E	WAI	; Заустави се ако је C=0

Решење:

Тренутни садржај програмског бројача:	$57A3_{16}$
Померај:	$+ 006B_{16}$
Нови садржај програмског бројача:	$\underline{580E}_{16}$

Пример 6.40 Ако се у примеру 6.27 промени инструкцијски кођ на адреси $57A2$ на вредност $A3$ одредити адресу гранања.

Решење:

Померај (одступање) је негативни број пошто је $A3_{16}=10100011_2=-93_{10}$.

Тренутни садржај програмског бројача:	$57A3_{16}$
Померај:	$+ FFA3_{16}$
Нови садржај програмског бројача:	$\nwarrow \underline{5746}_{16}$

Занемарити

Израчунавање помераја

Видели смо како да користимо померај (део инструкције за условно гранање) да би израчунали адресу на коју се грана процесор. Ово је корисно код анализе (туђих) програма на машинском језику, нарочито када они нису пропраћени дијаграмима тока или коментарима. Наравно, програм на машинском језику треба да израчуна вредност сваког одступања и да га у програму придружи свакој инструкцији гранања. Израчунавање вредности одступања обавља се на следећи начин:

1. За гранање унапред одузми текући садржај програмског бројача од новог садржаја програмског бројача. Уколико је резултат мањи или једнак од $7F_{16}=127_{10}$, резултат је жељен померај. Уколико је резултат већи од $7F$, инструкција гранања не може да обезбеди гранање на нову адресу. Другим речима адреса је ван опсега.
2. За гранање уназад одузима се нови садржај програмског бројача од текућег садржај програмског бројача. Резултат је број адреса за колико се извршавање програма враћа уназад. Потребно је да је овај број мањи или једнак $80_{16}=128_{10}$. Уколико је услов испуњен, од добијеног броја прави се други

6. Програмирање процесора MC68HC11

комплемент, да би се добио одговарајући негативни померај. Уколико услов није испуњен, нова адреса је ван опсега.

Пример 6.41 *Израчунати вредност помераја инструкције BEQ у инструкцијској секвенци:*

Адреса	Лабела	Инстр. кођ	Мнемоник	Коментар
0370	START	4A	DECA	; Умањи за 1 садржај акумулатора А
0371		27	BEQ \$03EE	; Гранај се на \$03EE
0372		??		; уколико је резултат нула
0373		97	STAA \$50	; У противном, смести
0374		50		; [А] у меморију на адресу \$0050
0375	KRAJ	3E	WAI	; заустави се

Решење:

По декодовању операционог кода инструкције BEQ процесор додаје вредност помераја на садржај програмског бројача [PC] = 0373₁₆ (текући [PC]). Нова вредност садржаја програмског бројача је 03EE₁₆, тако да ће бити гранање унапред. Померај се добија на следећи начин:

$$\begin{array}{rcl}
 \text{Тренутни садржај програмског бројача:} & 03\text{EE}_{16} & \\
 \text{Померај:} & - 0373_{16} & \\
 \hline
 \text{Нови садржај програмског бројача:} & 7\text{B}_{16} &
 \end{array}$$

Бајт на адреси \$0372 треба да буде \$7B.

Пример 6.42 *Израчунати одступање уколико инструкција BEQ треба да обезбеди гранање на адресу \$02FF.*

Решење:

Пошто је ово гранање уназад, нови садржај програмског бројача [PC] = 02FF₁₆, треба одузети од текућег садржаја програмског бројача [PC] = 0373₁₆.

$$\begin{array}{rcl}
 \text{Тренутни садржај програмског бројача:} & 0373_{16} & \\
 \text{Померај:} & - 02\text{FF}_{16} & \\
 \hline
 \text{Нови садржај програмског бројача:} & 74_{16} &
 \end{array}$$

Пример 6.43 Поновити пример 6.42 уколико инструкција BEQ треба да обезбеди гранање на адресу \$02AA.

Решење:

Тренутни садржај програмског бројача:	0373 ₁₆
Померај:	- 0238 ₁₆
Нови садржај програмског бројача:	C9 ₁₆

Пошто је разлика већа од 80₁₆, нови садржај програмског бројача је ван опсега, тако да инструкција BEQ не може да обезбеди одговарајући померај.

Гранање ван опсега

У примеру 6.43 BEQ инструкција не може да обезбеди одговарајући померај. У таквим ситуацијама програмер може, да би добио жељени резултат, да комбинује инструкцију гранања и JMP инструкцију. Може се употребити инструкција BNE: „гранај се ако резултат није нула”. Програм у примеру 6.43 може се модификовати на следећи начин:

Адреса	Лабела	Инстр. кдд	Мнемоник	Коментар
0370	START	10	SBA	; Одузети садржаје акумулатора А и В
0371		26	BNE \$0376	; Гранај се на \$0376 уколико
0372		03		; резултат није нула
0373		7E	JMP \$02A3	; У противном, скочи на
0374		02		; адресу \$02A3
0375		A3		
0376		97	STA \$50	; Смести резултат на \$0050
0377		50		;
0378	KRAJ	3E	WAI	; Заустави се

Када резултат операције одузимања није једнак нули инструкција BNE ће изазвати гранање унапред на \$0376. Уколико је резултат једнак нули, процесор неће прећи на извршавање инструкције са нове адресе (неће се гранати) већ ће се извршити JUMP инструкција на адреси \$0373 (гранање унатраг). Даље извршавање наставља се почевши од адресе \$02A3.

6. Програмирање процесора MC68HC11

Овај начин може се користити кад год је неопходно да се процесор грана на адресу која је ван опсега инструкције за условно гранање.

Посматраћемо уобичајене примере које користе инструкције условног гранања.

Пример 6.44 Написати програм који ће почињати на адреси \$E0C7, и који испитује податке са улазног порта на адреси \$C100, испитује бит најмање тежине података и условљава гранање на адресу \$E0F1 само уколико је LSB=1.

Адреса	Лабела	Инстр. код	Мнемоник	Коментар
E0C7	START	B6	LDAA \$C100	; Напуни акумулатор А са
E0C8		C1		; улазног порта
E0C9		00		
E0CA		44	LSRA	; Помери LSB у С бит
E0CB		25	BCS \$E0F1	; Гранај се на \$E0F1 ако
E0CC		24		; је C =1
E0CD		.		; У супротном настави....

Решење:

Инструкција LDAA \$C100 учитава податак са улазног порта на адреси \$C100. LSRA инструкција помера бит најмање тежине акумулатора А у С бит. BCS инструкција проверава да ли је С бит 1 (померени бит акумулатора А) и одређује да ли ће доћи до гранања: ако С бит има вредност 1 долази до гранања, у супротном не.

Пример 6.45 Изменити пример 6.44 тако да се грана на адресу \$E090 ако је бит највеће тежине (MSB) 1.

Адреса	Лабела	Инстр. код	Мнемоник	Коментар
E0C7	START	B6	LDAA \$C100	; Напуни акумулатор А податком са
E0C8		C1		; улазног порта
E0C9		00		
E0CA		2B	BMI \$E090	; Гранај се на \$E090 ако је бит највеће
E0CB		C4		; тежине 1 (MSB = 1)
E0CC		.		; У супротном настави са извршавањем.

Решење:

У овом примеру није неопходно користити све три инструкције као у примеру 6.44, већ се може искористити инструкција BMI, пошто је бит највеће тежине (MSB бит) - бит знака. Када се LDAA инструкција извршава N бит ће се поставити на вредност бита највеће тежине податка. Затим BMI инструкција тестира N бит и грана се уколико је његова вредност 1.

Пример 6.46 Претпоставити да су у акумулаторе учитани подаци са улазног порта и да су то бројеви без знака. Написати део програма који почиње од адресе \$E000 и грана се на \$E070 уколико је садржај акумулатора A већи од садржаја акумулатора B; у супротном програм се зауставља.

Решење:

Адреса	Лабела	Инстр. код	Мнемоник	Коментар
E000	START	10	SBA	; Одузми садржаје акумулатора A и B
E001		22	BHI \$E070	; Гранај се на адресу \$E070
E002		6D		; ако је резултат > 0
E003	KRAJ	3E	WAI	; У супротном се заустави

Инструкција SBA одузима садржај акумулатора B од A и смешта резултат у акумулатор A.

Пример 6.47 Шта би требало променити у коду програма у примеру 6.46 ако би акумулатори садржали бројеве са знаком?

Решење:

Промена инструкције BHI у BGT. Инструкција BGT ће тестираће резултат одузимања бројева са знаком.

Пример 6.48 На слици 6.9 дат је дијаграм тока програма који множи два 8-битна броја без знака, са адресе \$00A1 и \$00A2 и смешта их у меморију. Множење се постиже процесом узастопног сабирања. На пример, претпоставимо да на адреси \$00A1 број 2 а адреси \$00A1 број 4. Додавањем броја 2 четири пута може се добити производ $2 \cdot 4$. Другим речима, вредност податка са адресе \$00A2 је број који указује колико пута додајемо вредност податка са адресе \$00A1.

У дијаграму тока, акумулатор А се брише и акумулатор В се пуни податком са адресе \$00A2. Петља у дијаграму тока представља поновљено извршавање сабирања податка са адресе \$00A1 и садржаја акумулатора А. После сваког сабирања садржај акумулатора В се декрементира и проверава да ли је његов садржај достигао нулту вредност. Када достигне нулу, постигнут је потребан број операција сабирања, и коначан резултат је у акумулатору А.

Написати програм за овај дијаграм тока који почиње на адреси \$0260. Претпоставити да су подаци у меморијским локацијама \$00A1 и \$00A2, респективно, и да резултат треба сместити на \$00A3.

Решење:

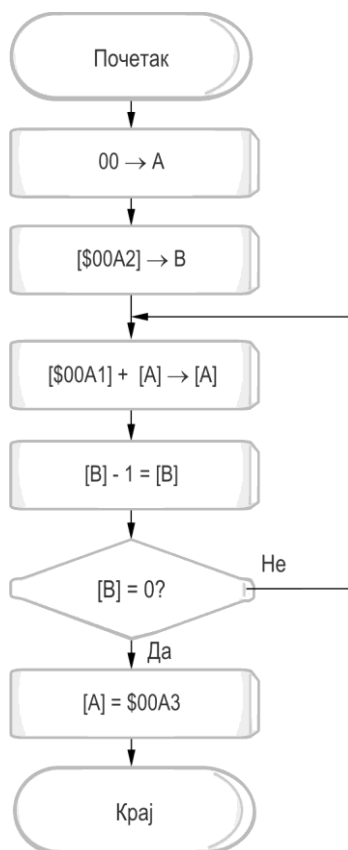
Лабела	Мнемоник	Коментар
START	CLRA	; Постави садржај акумулатора А на 00
	LDAB \$A2	; Напуни акумулатор В са \$00A2
SUM	ADDA \$A1	; Додај податак са \$00A1 на садржај акумулатора А
	DECB	; Декрементирај садржај акумулатора В
	BNE SUM	; Ако није нула гранај се и поново додај податак са \$00A2
	STAA \$A3	; Сачувај резултат на \$00A3
KRAJ	WAI	; Крај

У овом програму акумулатор В користи се као бројач уназад, и то од иницијалне вредности која је једнака другом операнду. Инструкција BNE се користи да би проверавала да ли је после сваке операције декрементирања садржај акумулатора В различит од 0 ($[B] \neq 00$). На основу тога се одређује када је извршен потребан број операција сабирања.

6.8.2 Инструкције скокова

Инструкција скока¹ омогућава да се извршавање програма настави на било којој адресној локацији 64kB меморије.

¹ *Jumps*



Слика 6.9 Дијаграм тока програма за множење преко сабирања из примера 6.48

Функција	Мнемоник	Директно	Проширено	Индексирано (X регистар)	Индексирано (Y регистар)	Имплицитно
Скок (<i>Jump</i>)	JMP	X	X	X	X	X

Табела 6.9

6.8.3 Позиви и повратак из потпрограма

Програми, обично, захтевају да се одређене функције изводе више пута за време извршавања програма. На пример, програм може да

избацује ASCII кодиране податке ка штампачу у различитим тачкама у програму. Други пример је потреба да се изведе множење два операнда неколико пута за време извршавања програма. Скуп инструкција потребних за извршавање ових функција може да буде написан у делу програма где су оне потребне. Ово је прихватљиво када је број инструкција мали (две или три инструкције) или када број понављања није много велики. У супротном програмер може да реализује функције као потпрограма¹.

Потпрограма је скуп инструкција које обављају одређени задатак. Једном се напише и смести у одређену област меморије. Кад главни програм жели да извршава тај задатак, процесор уписује у програмски бројач адресу потпрограма. После извршавања потпрограма наставља са извршавањем главног програма.

Дијаграм тока² приказан на слици 6.10 илуструје како се потпрограма користи у оквиру програма. Постоје заправо два дијаграма тока.

На левој страни цртежа је главни програм који процесор извршава. На десној страни је потпрограма који ће процесор да изврши два пута за време извршавања главног програма. Процесор извршава потпрограма у две различите тачке у главном програму. Блок 1 налаже процесору да пређе на извршавање блока 2.

Када заврши са извршавањем потпрограма процесор наставља са извршавањем инструкција главног програма (блок 3). У блоку 4, процесор поново прелази на извршавање потпрограма. После другог извршавања потпрограма, процесор наставља са извршавањем главног програма (блок 5).

Коришћење потпрограма може да се понови више пута за време главног програма. На овај начин коришћење меморије је много ефикасније од решења са понављањем скупа инструкција сваки пут када су оне потребне у главном програму.

Инструкције за позиве и повратак из потпрограма³ су BSR, JSR и RTS. Ове инструкције обезбеђују начин да се једноставно задатак већег обима

¹ Subroutine

² Flow-chart

³ Subroutine Calls And Returns



Слика 6.10 Илуструје како скок у потпрограм може да се појави у било којој тачки у главном програму; процесор се на крају потпрограма увек враћа и наставља са извршавањем главног програма

расподели између више програмера. Програм се може поделити у целине (задатке) које се називају потпрограмима.

Процесор аутоматизује процес памћења адреса у главном програму у коме наставља са извршавањем, пошто се заврши извршавање потпрограма. Ове адресе аутоматски се уписују у стек меморију код позива потпрограма и извршавања инструкција BSR и JSR. Адресе се учитавају са стек меморије за време извршавања инструкције RTS која окончава извршавање потпрограма.

6. Програмирање процесора MC68HC11

Операција	Мнемоник	Релативно	Проширено	Директно	Индексирано (X регистар)	Индексирано (Y регистар)	Имплицитно
Гранање у потпрограм (Branch to Subroutine)	BSR	X					
Скок у потпрограм (Jump to Subroutine)	JSR		X	X	X	X	
Повратак из потпрограма (Return from Subroutine)	RTS						X

Табела 6.10

Инструкције JSR и RTS контролишу начин на који процесор одлази и како се враћа из потпрограма. Извршавањем инструкција „скочи у потпрограм” JSR¹, процесор се усмерава на извршавање програма на адреси потпрограма. Извршавањем инструкција RTS² (повратак из потпрограма) процесор враћа извршавање програма у главни програм. Како се ова инструкција извршава илустровано је на примеру 6.49.

Пример 6.49 Извршавање главног програма започиње се на адреси \$C100 и извршава се до адресе \$C107. Потпрограм POM је смештен на адреси \$C160-\$C164.

Адреса	Лабела	Инстр. код	Мнемоник	Коментар
C100	GLV	96	LDAA \$A6	; Пуни акумулатор A подацима
C101		A6		
C102		BD	JSR POM	; Скок на POM потпрограм на \$C160
C103		C1		;
C104		60		
C105		97	STAA \$A7	; Наставак главног програма
C106		A7		; и пуњење акумулатора A
C107	KRAJ	3E	WAI	; Заустављање
.	.	.	.	.

¹ Jump to SubRoutine

² ReTurn from Subroutine

Адреса	Лабела	Инстр. кођ	Мнемоник	Коментар
C160	POM	44	LSRA	; Померање акумулатора А
C161		44	LSRA	; удесно четири пута
C162		44	LSRA	
C163		44	LSRA	
C164	POV	39	RTS	;Враћање на главни програм

Када процесор извршава инструкцију на адреси \$C102, она учитава у програмски бројач адресу потпрограма који се налази на адреси \$C160. Претходни садржај (текући) програмског бројача [PC]=\$C105₁₆ смештен је у стек меморију. Уочимо да је текући садржај програмског бројача адреса операционог кода инструкције, која се налази иза JSR инструкције. Ова адреса, \$C105, је адреса у главном програму на коју ће се процесор вратити по извршењу потпрограма. Назива се повратна адреса. Ако претпоставимо да је садржај стек показивача \$01FF ([SP]=\$01FF₁₆) пре извршавања JSR инструкције подаци у стек меморији после извршавања JSR инструкције биће:

Адреса стек меморије	Подаци	
01FE	C1	} повратна адреса
01FF	05	

После стављања адресе потпрограма у програмски бројач процесор започиње извршавање потпрограма на адреси \$C160. Када процесор извршава RTS инструкцију на адреси \$C164 онда она учитава податке из стек меморије у програмски бројач и наставља са извршавањем главног програма.

Пример 6.50 *Шта се дешава ако се у програм инструкција JSR замени инструкцијом JMP?*

Решење:

Разлика између JMP и JSR инструкција је у томе што када се извршава JMP инструкција не чува се адреса за повратак (садржај програмског бројача) на стек меморији, као што је случај са инструкцијом JSR. Ако имамо инструкцију JSR \$C160, и заменимо је са JMP \$C160 процесор пуни програмски бројач адресом \$C160 али на стек меморији не чува

адресу повратног скока. Затим извршава потпрограм. Када процесор извршава инструкцију RTS (у случају да је на почетку коришћена JMP инструкција), узима са стека два бајта података и смешта их у програмски бројач. Ова два бајта неће бити повратна адреса тако да процесор одлази на непредвиђену адресу као своју следећу инструкцију. Даљи резултат је извршавање неке случајне, непредвиђене операције. Другим речима долази до потпуног „слома” програма.

Ова анализа показује да JMP инструкција није предвиђена да се користи за приступање потпрограму.

Инструкција JSR може да користи проширено или индексно адресирање. Инструкција за гранање у потпрограм (BSR¹) користи релативно адресирање. Ове инструкције се ни по чему другом не разликују. Инструкција BSR има ограничење да се може гранати у потпрограме који су у оквиру опсега одступања од -128 до +127.

Повезивање (гнежђење) потпрограма

Потпрограм може да садржи JSR инструкцију; другим речима потпрограм може да скаче у други потпрограм. То се зове повезивање (гнежђење) потпрограма. У сложенијим програмима може да постоји неколико нивоа повезивања (један потпрограм скаче у други итд.) Као и у случају позива једног потпрограма, стек меморија служи за чување повратне адресе, свака следећа инструкција RTS враћа извршавање потпрограма на место где је био пре позива тог потпрограма.

6.8.4 Инструкције које воде рачуна о прекидима

Инструкције: RTI, SWI, WAI везане су за рад са прекидима.

Операција	Мнемоник	Имплицитно адресирање
Повратак из прекида (<i>Return from Interrupt</i>)	RTI	X
Софтверски прекид (<i>Software Interrupt</i>)	SWI	X
Чекање на прекид (<i>Wait for Interrupt</i>)	WAI	X

Табела 6.11

¹ Branch to SubRoutine

При извршавању инструкције за софтверски прекид (SWI) у стек меморију се смештају садржаји свих интерних регистара укључујући и повратну адресу. Инструкција SWI је специфична по томе што се прекид програмски генерише за разлику од других прекида које асинхронно изазивају спољашњи модули.

Извршавање инструкција WAI има две битне намене:

- постављање микроконтролера у стање смањене потрошње¹ док се не појави захтев за прекид;
- да се смањи време одзива (кашњења) на неке важне захтеве за прекид. Када се WAI инструкција извршава процесор смести садржаје својих интерних регистара на стек меморију. Када се прекид појави процесор је спреман да учита адресни вектор тако да је кашњење које би наступило смештањем регистара на стек меморију уклоњено.

Постоје три инструкције које програмер може да користи, а које се односе на прекиде²:

CLI³ - обриши бит (флег) који маскира прекид,

SEI⁴ - постави бит који маскира прекид и

RTI⁵ - повратак из потпрограма који обрађује прекид.

6.8.5 Некласификоване инструкције

Додатне неклаификоване⁶ инструкције су NOP, STOP и TEST.

Инструкција NOP може се искористити да унесе мало временско кашњење у ток програма. Корисна је за испуњење временских захтева у комуникацији са спорим периферијама. Убацивањем NOP инструкција могу се обезбедити и већа кашњења. Када процесор извршава NOP инструкцију, она не извршава ниједну операцију осим што инкрементира програмски бројач као припрему за следећу наредбу. Три уобичајене ситуације где је NOP инструкција корисна су:

¹ *Reduced power-consumption standby state (WAIT mode)*

² *Interrupt Handling Instructions*

³ *Clear Interrupt Mask Fleg I*

⁴ *Set Interrupt Mask Fleg I*

⁵ *Return from Interrupt*

⁶ *Miscellaneous*

1. Замена програмског кода који није више потребан у периоду развоја програма. Програм је смештен у RAM меморији где се може лако модификовати. Ако програмер закључи да одређене инструкције нису потребне, може их заменити са NOP инструкцијама. Она неће утицати ни на регистре ни на меморијске локације. Пошто се програм развије, и ради исправно, програмер може да уклони NOP инструкције.
2. Остављање места у програму за убацивање програмских кодова за време развоја програма. За време писања програма, програмер може да претпостави да ће касније желети да убаци на различитим тачкама у програму више инструкција. Да би то обезбедио NOP се поставља на одговарајућа места.
3. Да би се генерисало временско кашњење процесор ће извршавати NOP операцију у два циклуса такт сигнала. Програмер може да постави онолико NOP инструкција колико је потребно временско кашњење. За кашњења већа од неколико такт циклуса нису погодне, па се користи други метод.

Операција	Мнемоник	Имплицитно адресирање
Нема операције - кашњење од 2 циклуса (No Operation)	NOP	X
Заустављање такт сигнала (<i>Stop Clocks</i>)	STOP	X
Чекање на прекид (<i>Test</i>)	TEST	X

Табела 6.12

За време тражења грешака у програму¹ уобичајен поступак је замењивање различитих инструкција са NOP инструкцијом (односно њеним операционим кодом) да би се ефикасно уклониле важеће инструкције без потребе да се реорганизује остатак програма.

Инструкције за модификовање меморије или инструкције за праћење стања и грешака² могу се једноставно уклонити и поново вратити да би се видео њихов утицај.

Повремено програмер треба да разреши проблем финог подешавања

¹ Debugging

² Debug Monitor

кашњења кроз различите програмске целине које се извршавају. У таквим случајевима корисно је користити инструкцију - никада се не гранај (BRN) за чије извршавање су потребна три циклуса такт сигнала.

Инструкција STOP је специфична инструкција пошто доводи до тога да се „замрзава“ осцилатор и сви такт сигнали микроконтролера. Ово замрзнуто стање назива се метод рада са заустављањем¹ а са овим методом се потрошња значајно смањује. Извршавање ове инструкције зависно је од S бита у регистру услова. Уколико је S бит постављен (S=1), инструкција STOP се третира као инструкција NOP и наставља се са извршавањем следеће инструкције.

Инструкција TEST користи се једино за време фабричког тестирања и третира се као непостојећи операциони кођ у нормалном раду микроконтролера. Ова инструкција изазива недефинисана стања на адресној магистрали (нпр. бројање унатраг) и због тога није применљива у реалној апликацији.

6.9 ПРИМЕНА ИНДЕКСНОГ АДРЕСИРАЊА

Индексно адресирање користи се у програмирању са блоковима или табелама података. У овом поглављу анализираћемо неке уобичајене апликације које користе индексно адресирање.

Коришћење меморијских табела може значајно да убрза микрорачунарске операције, као што су конверзија кода, комплексна аритметичка израчунавања, генерисање порука итд. Табеле су једноставна листа везаних података смештених у узастопним меморијским локацијама. За задату апликацију подаци у овим табелама су обично константе и обично су смештене у ROM меморији, или учитане у RAM меморију са диска.

Суштинска предност табела је да елиминишу потребу за било каквим рачунањем. Резултат се добија једноставно проналажењем одговарајуће адресе у табели. За правилно коришћење табела потребно је поставити податке и дефинисати начин за приступ.

Конверзија хексадецималних података у ASCII податке

Конверзија кода је задатак који ради процесор. Улазни периферијски

¹ STOP mode

уређај обично даје податке у ASCII, BCD коду. Да би их обрадио процесор мора да их конвертује. Резултат обраде података мора се, пре слања излазном уређају, превести у одговарајући код.

Често се дешава да процесор треба да шаље хексадецималан податак ка излазном уређају који прихвата само ASCII кодирани податак. Претпоставимо, на пример, да је хексадецимална цифра В смештена на некој меморијској локацији. Уколико процесор жели да шаље цифру ка штампачу он мора да B_{16} конвертује у ASCII код карактера В. У табели 6.3 дат је ASCII код свих хексадецималних цифара. Пошто је код 7-битни, његов бит највеће тежине је 0.

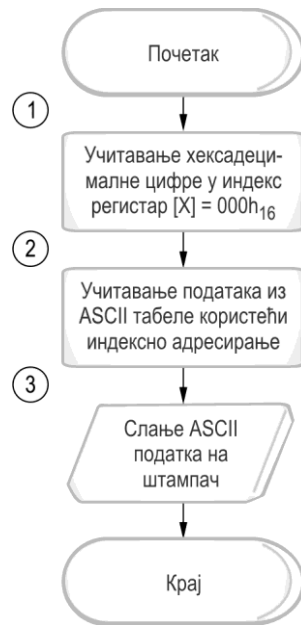
ASCII код		
Хекс. цифра	Бинарно	Хекс.
0	00110000	30
1	00110001	31
2	00110010	32
3	00110011	33
4	00110100	34
5	00110101	35
6	00110110	36
7	00110111	37
8	00111000	38
9	00111001	39
A	01000001	41
B	01000010	42
C	01000011	43
D	01000100	44
E	01000101	45
F	01000110	46

Табела 6.13

Поставићемо следећи проблем. Меморијска локација \$00F1 садржи једну хексадецималну цифру (тј. садржи $0h_{16}$). Процесор треба да конвертује

Меморија		
Адреса	Податак.	
00 В	0	30
00 В	1	31
00 В	2	32
00 В	3	33
00 В	4	34
00 В	5	35
00 В	6	36
00 В	7	37
00 В	8	38
00 В	9	39
00 В	A	41
00 В	B	42
00 В	C	43
00 В	D	44
00 В	E	45
00 В	F	46

↖ хекс број
 ↖ ASCII код хекс. број



Слика 6.11 Конверзија хексадецималног броја у ASCII код:
(а) табела података у меморији (б) дијаграм тока

ову хексадецималну цифру у ASCII код и пошаље ка излазном порту штампача на адреси \$A900. ASCII код хексадецималне цифре налази се у меморијској табели на адреси \$00B0-\$00BF, као што је приказано на слици 6.11а. Претпоставићемо да ова табела постоји. Обично је смештена у ROM делу меморије. Уочимо како је цифра најмање тежине на адреси хексадецимална цифра чији је ASCII код смештен на тој адреси. Слика 6.11б је дијаграм тока програма који обавља конверзију. Анализирајмо сваки појединачни блок.

У првом блоку X регистар је напуњен хексадецималном цифром h, коју треба конвертовати у ASCII код. Регистар X је 16-битни и садржи четири хексадецималне цифре. Два бита највеће тежине су 0, а два бита

6. Програмирање процесора MC68HC11

најмање тежине треба напунити хексадецималним бројем \$0h са адресе \$00A1. Ово се може постићи инструкцијском секвенцом:

CLR \$00F0

LDX \$00F0

Резултат CLR инструкције је $[00F0]=00_{16}$. Друга инструкција пуни регистар X са две узастопне меморијске локације \$00F0 и \$00F1 тако да је $[X]=000h_{16}$.

У другом блоку регистар X се користи као индексни регистар који бира ASCII код који одговара хексадецималној цифри h у табели што се постиже инструкцијом:

LDAA \$B0,X

Ова инструкција извршава операцију пуњења аумулатора A са меморијске локације чија је адреса $[X]+80$. На пример, уколико је $h=B$, онда ће $[X]=000D_{16}$ и аумулатор A ће бити напуњен са адресе $\$000B+\$B0=\$00BB$. Тако је $[A]=42_{16}$, тј. ASCII код за карактер B. У ствари, садржај регистра X показује улаз у табелу.

У трећем блоку се садржај акумулатора A ($[A]=42_{16}$) шаље ка излазном порту штампача уз коришћење инструкције:

STAA \$A900

Штампач ће одштампати карактер B.

У даљем тексту дат је комплетан програм. Претпоставимо да се табела са слике 6.11a већ налази у меморији. Пажљиво је проучити, анализирати програм за $[\$00B0]=04_{16}$ и за $[\$00B0]=0F_{16}$.

START	CLR \$00A0	; Обрисати \$00A0
	LDX \$00A0	; Пуни индекс регистар X са хекс. бројем који треба конвертовати
	LDAA \$80,X	; Пуни акумулатор A податком из ASCII табеле
	STAA \$A900	; шаље ASCII карактер на порт \$F900 (штампач)
KRAJ	WAI	; Заустављање

6.10 ПРОГРАМСКИ КОНТРОЛИСАНИ ВРЕМЕНСКИ ИНТЕРВАЛИ

Кад се програм користи да контролише периферије (улаз или излаз) често је потребно у програме са дефинисаним трајањем убацили временска кашњења. Програмирано временско кашњење потребно је у следећим ситуацијама:

1. Логичким сигналимa са одређених улазних уређаја потребно је време да се стабилишу; пример су механички прекидачи са својим „прекидачким одскоком”. Претпоставимо да програм налаже процесору да читава податке са улазног порта са циљем да детектује промену стања прекидача. Програм ће убацили временско кашњење између узастопних читавања, тако да се неће детектовати прекидачки одскок и интерпретирати као прелаз прекидача из једног у друго стање.
2. Програм се може користити да генерише импулсе ка излазном порту са специфичним трајањем импулса да би управљао укључивањем и искључивањем уређаја као што су мотори, монитори итд.
3. Када је процесор повезан са серијским улазно/излазним уређајима, као што је серијски штампач, треба да генерише и/или реагује на промене логичких нивоа који се јављају у одређеним временским интервалима. На пример, предајни и пријемни подаци су брзине један бит сваких 0,1ms.

6.10.1 Програмирано временско кашњење

Временско кашњење добија се у програму убацивањем скупа инструкција које процесор извршава у трајању еквивалентном жељеном кашњењу. Инструкције немају другу функцију него да повећају трајање.

Пример 6.51 *Такт сигнал процесора је 1MHz. Користећи 11. поглавље пронаћи колико је укупно време извршавања ових инструкција.*

```
LDX #$F305
NOP
NOP
NOP
NOP
DEX
```

Решење:

У 11. поглављу може да се нађе време потребно за извршавање инструкција. За извршавање LDX #F305 инструкције потребна су три такт циклуса, за инструкцију DEX четири, а за сваки NOP два такт циклуса. Тако је укупно време за извршавање $3+2+2+2+2+4=15$ такт циклуса = 15μsec.

У овом примеру за добијање временског кашњења искоришћена је директна инструкцијска секвенца. Овај метод је ограничен на релативно кратка кашњења пошто свака од инструкција доприноси само пар наносекунди; на пример, за кашњење од 1ms потребно је 500 NOP инструкција. За кашњење од 30 до 40 такт циклуса ефикаснији метод је петља са временским кашњењем.

6.10.2 Петље са временским кашњењем

Једноставна петља са временским кашњењем ради на следећи начин:

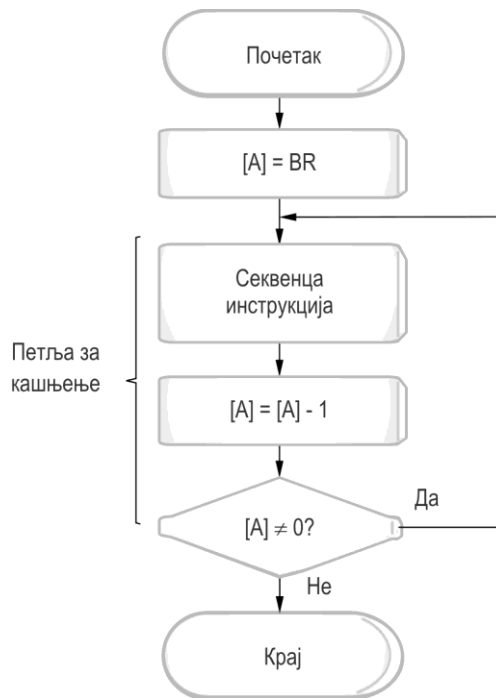
1. пуни регистар иницијалном вредношћу,
2. извршава инструкцијску секвенцу за временско кашњење,
3. декрементира регистар,
4. ако регистар није нула, понавља кораке 1 и 2.

Регистар се користи као бројач унатраг, са иницијалном вредношћу која одређује колико пута се петља извршава.

Дијаграм тока на слици 6.12 илуструје петљу са временским кашњењем која користи акумулатор А као бројач. Први блок показује да је акумулатор А напуњен бројем BR. Највећа вредност броја BR је 255₁₀ и она одређује колико пута се петља извршава. Петља за кашњење састоји се од три блока а један од њих је скуп инструкција који производи жељено кашњење.

Пример 6.52 Наредни програм у петљи за кашњење користи број пролазака кроз петљу ($BR=29_{16}=41_{10}$) и једну NOP инструкцију. Израчунати укупно време извршавања ако се зна да је такт сигнал од 1Mhz.

PETLJA	(2) NOP	; Касни два такт циклуса
	(2) DECB	; Декрементира садржај
		акумулатора В



Слика 6.12 Дијаграм тока временске петље која користи акумулатор А

(4) BNE \$0352 ; Ако није нула, иди опет на
PETLJA

KRAJ (9) WAI ; Зауостављање

Решење:

Ako se program pogleda može se приметити да се инструкција LDAB изводи само једном. NOP, DECB и BNE инструкције се изводе 41 пут ($BR=29_{16}=41_{10}$). Број у загради који претходи сваком мнемонику инструкције указује на број такт циклуса који је потребан за извршавање тих инструкција. WAI инструкција се изводи само једном. На основу овога може се израчунати укупно време извршавања:

$$2 + 41 \times (2+2+4) + 9 = 339 \text{ такт циклуca} = 339\mu\text{s}$$

Као што овај пример показује, укупно време кашњења израчунава се додавањем времена извршавања инструкција које су ван петље, времену извршавања инструкција које су у оквиру петље помноженим са бројем BR. Укупно кашњење је:

$$\text{укупно кашњење} = T_s + (BR \cdot T_p)$$

где је T_s време кашњења спољно, ван петље, а T_p време кашњења у оквиру петље.

Пример 6.53 *Колика треба да буде вредност BR у Примеру 6.52 да би се добило кашњење од приближно 1,8ms.*

Решење:

Пошто је $1,8\text{ms} = 1800\mu\text{s}$, потребно је

$$1800 = T_s + (BR \times T_p) = 11 + (BR \times 8)$$

односно

$$BR = 223,625$$

Најближа цела вредност је $BR = 224_{10} = E0_{16}$. Користећи $BR = E0_{16}$ стварно укупно кашњење ће бити:

$$11 + (224 \times 8) = 1803\mu\text{s} = 1,803\text{ms}$$

Пример 6.54 *Како би изгледао програм из претходног примера да би се добило кашњење приближно од 3,7ms.*

Решење:

Кашњење у овом случају не би могло да се добије на начин описан у претходном примеру јер је превише велико да би се једноставно постигло увећавањем BR. Када је вредност $BR = FF_{16} = 255_{10}$, кашњење ће бити:

$$11 + 255 \times 8 = 2051\mu\text{s} = 2,051\text{ms}$$

У овом случају иде се на то да се повећа укупно кашњење повећањем кашњења унутар петље (T_p), додавањем одговарајућег броја NOP инструкција, или прављењем петље унутар петљи. У примеру који следи користићемо проширивање са NOP инструкцијама. Да бисмо добили кашњење од 3,9ms, потребно је:

$$3900 = 11 + (BR * T_p)$$

или

$$BR * T_p = 3889$$

Уколико одаберемо $BR = 200_{10}$ онда је

$$T_p = 3889/200 \approx 20\mu s$$

Тада се може убацити још седам NOP (2) инструкција, да би заједно са инструкцијама DECA (2) и BNE (4) добили $T_p = 20\mu s$. Тачно кашњење биће:

$$11 + (200 * 20) = 4011\mu s = 4,011ms$$

Ово кашњење је за $111\mu s$ веће од планираног, па је у овом случају најбоље направити корекцију над бројем BR. Ако је $BR = 215_{10}$ тада добијамо:

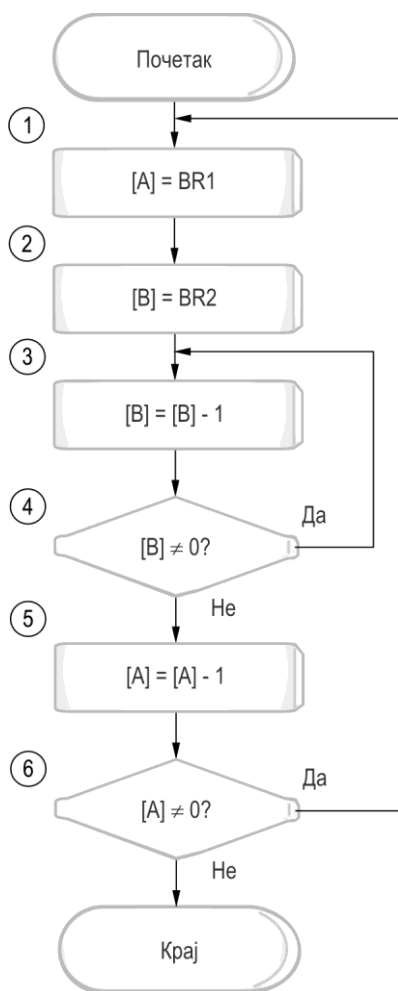
$$T_p = 3889/215 \approx 18\mu s$$

У том случају неће се користити седам NOP инструкција, него шест. Тачно кашњење ће бити:

$$11 + (215 * 18) = 3881\mu s = 3,881ms$$

Одступање од $19\mu s$ може се занемарити, ако се не захтева велика прецизност. Промењени програм је:

START	LDAA #\$D7	; Пуни акумулатора А са $BR = D7_{16} = 215_{10}$
PETLJA	NOP	; 6 NOP инструкција (кашњење)
	NOP	
	NOP	
	NOP	
	NOP	
	NOP	
	DECA	; Декрементирај садржај акумулатора А
	BNE PETLJA	; Ако је $\neq 0$ гранај се назад на PETLJA
KRAJ	WAI	; Заустављање



Слика 6.13 Дијаграм тока петље за кашњење којом се добија веће укупно кашњење

Петља у оквиру петље

У овом, последњем примеру, увећава се укупно кашњење додавањем више инструкција у петљи за кашњење. Овај метод постаје неефикасан за дужа кашњења. На пример, да би се добило кашњење од 25ms, мора се унутар петље за кашњење употребити четрдесет пет NOP инструкција.

За већа кашњења боље решење је петља унутар петље. Дијаграм тока приказан је на слици 6.13.

У овом случају оба акумулатора користе се као бројачи. Акумулатор В користи се као бројач унутрашње петље, а акумулатора А као бројач спољашње петље. Спољашња петља, блокови 2-6, извршаваће се BR1 пута. Унутрашња петља, блокови 3 и 4, извршаваће се BR2*BR1 пута. Видимо да се дуга кашњења могу добити коришћењем великих вредности за BR1 и BR2.

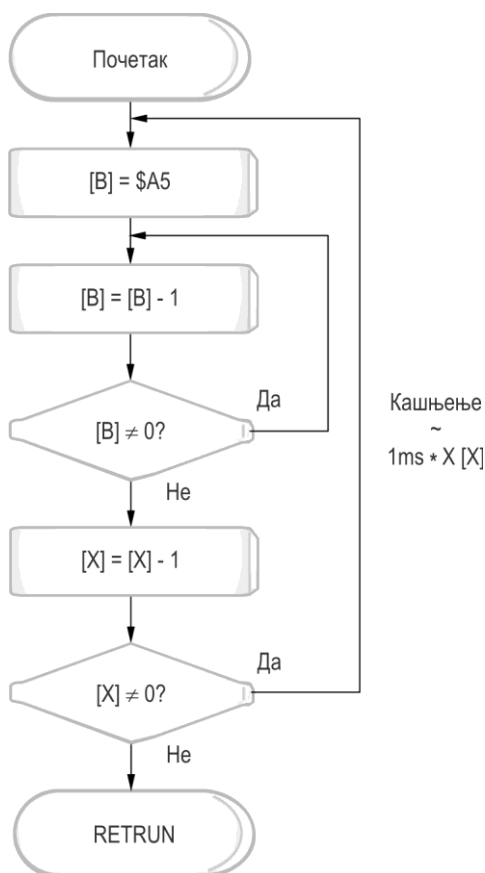
6.11 ПОТППРОГРАМИ ЗА КАШЊЕЊЕ

Када апликација захтева генерисање великог броја временских интервала, много је ефикасније користити потпрограме за генерисање временског кашњења, од уметања програмских секвенци на свим местима где се временско кашњење захтева. Дијаграм тока на слици 6.14 је потпрограм за кашњење који производи приближно кашњење од $1\text{ms} * [X]$. На пример, уколико је $[X]=0005$ пре почетка потпрограма, кашњење ће бити 5ms. Пре скока у потпрограм главни програм треба да напуни X жељеном вредношћу. Потпрограм користи X регистар као бројач спољашње петље и акумулатора А као бројач унутрашње петље.

Потпрограм за кашњење ($1\text{ms} * [X]$)

KA1MS	LDAA #A5	; $[A] = A5_{16} = 165_{10}$
	PETLJA DECA	; Декрементирај садржај акумулатора А
	BNE PETLJA	; Ако није 0, гранај се назад на PETLJA
	DEX	; Декрементирај садржај индекса регистра X
	BNE KA1MS	; Ако није 0, гранај се назад на KA1MS
RETURN	RTS	; Повратак на главни програм

Пример 6.55 а) Коликом вредношћу треба напунити индекс регистра X у претходном примеру да би се добило кашњење од 1,5s. б) Колико је максимално кашњење за овај потпрограм, и са којом вредношћу



Слика 6.14 Дијаграм тока потпрограма са кашњењем = $1\text{ms} * [X]$

треба напунити индекс регистар X за максимално кашњење.

Решење:

а) $1,5\text{s} = 1500\text{ms}$. Због тога је потребно да је $[X] = 1500_{10} = 05\text{DC}_{16}$.

б) Максимално кашњење се јавља за $[X] = \text{FFFF}_{16} = 65536_{10}$ и то је $65536\text{ms} = 65,536\text{s}$.

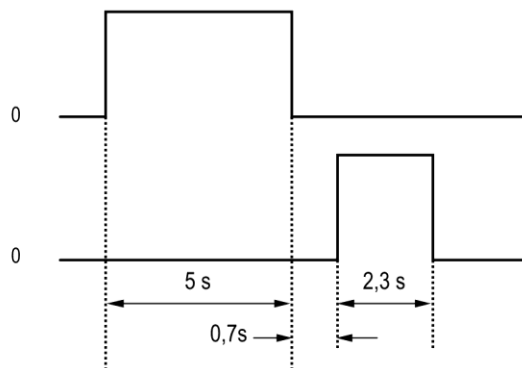
6.11.1 Примена потпрограма за кашњење

Процесор се често користи за управљање спољашњим уређајима као што су мотори, монитори, грејачи и извори светлости. На пример, у машини за копирање процесор управља следећим операцијама: када и

а)



б)



Слика 6.15 а) Процесор може да управља спољашњим уређајем тако што ће поставити и обрисати одговарајуће битове излазних регистра;

б) типичан временски дијаграм за управљање излазним уређајима.

колико ваљак треба да се креће, када су и колико лампе за развијање укључене и када треба променити бројеве на бројачу копија. Процесор типично управља овим спољашњим уређајима преко регистра излазног порта. Поједностављен дијаграм приказан је на слици 6.15а.

Неки, или сви, излази регистра Q₀- Q₇ користе се за управљање спољашњим уређајима. Претпоставимо да се користе само Q₇ и Q₆, и желимо да процесор уради следеће:

- а) обрише и Q_7 и Q_6 ;
- б) постави $Q_7=1$ за 5s, затим поново обрише;
- в) чека 0,7s;
- г) постави $Q_6=1$ у трајању од 2,3s, па обрише.

Временски дијаграм је приказан на слици 6.15б.

Постављени захтеви могу се реализовати програмом који користи потпрограм за кашњење који се налази на адреси KA1MS (нпр. \$C800). Програм треба да генерише три одвојена временска интервала: 5s; 0,7s и 2,3s. Написан је у асемблерском језику и приказан је у даљем тексту. Претпоставка је да је адреса излазног порта 8000_{16} . Проучити пажљиво програм и потврдити да генерише временске сигнале са слике 6.15б. Уочити како програм користи акумулатор В за постављање одговарајућих битова излазног регистра. Такође уочити како се добијају различити временски интервали пуњењем Х регистра са одговарајућом вредношћу и скоком у потпрограм за кашњење који се налази на адреси KA1MS.

Мнемоник	Коментар
CLR \$8000	; $Q_7 = Q_6 = 0$
LDAB #\$80	; $[B] = 80_{16} = 10000000_2$
STAB \$8000	; $Q_7 = 1$
LDX #\$1388	; Напуни Х регистар бројем за кашњење од 5s
JSR KA1MS	; Скок на потпрограм за кашњење
CLR \$8000	; $Q_7 = 0$
LDX #\$02BC	; Напуни Х регистар бројем за кашњење од 0,7s
JSR KA1MS	; Скок на потпрограм за кашњење
LDAB #\$40	; $[B] = 40_{16} = 01000000_2$
STAB \$8000	; $Q_6 = 1$
LDX #\$08FC	; Напуни Х регистар бројем за кашњење од 2,3s
JSR KA1MS	; Скок на потпрограм за кашњење
CLR \$8000	; $Q_6 = 0$
WAI	; Заустављање

Пример 6.56 Написати програм који генерише правоугли облик сигнала од 100Hz на Q_0 излазу са слика 6.15а.

Решење:

Q_0 треба да мења стање са 0 на 1 сваких 5ms, пошто је 10ms периода сигнала на учтаности од 100Hz.

Користићемо излазни регистар као бројач, и инструкцију INC која мења стање Q_0 излаза сваких 5ms. Програм који следи ће произвести жељени облик сигнала.

START	CLR \$8000	; Поставља излазних битова порта на 0
PETLJA	INC \$8000	; Инкрементирање излазног регистра
	LDX #\$0005	; Напуни X регистар за кашњење од 5ms
	JSR KA1MS	; Скок на потпрограм за кашњење KA1MS
	BRA \$0303	; Гранање натраг у петљу PETLJA

6.12 ПРОЦЕС РАЗВИЈАЊА СОФТВЕРА

Проучили смо инструкцијски скуп процесора MC68HC11, видели уобичајене технике за програмирање, и написали неке, релативно кратке програме. У пракси, стварно писање програмског кода је мали део процеса развоја софтвера. Ми ћемо у даљем тексту описати процес микрорачунарског програмирања и детаље комплетног процеса развијања програма. Указаћемо да је потребно много корака да би се добио програм који исправно ради у датој апликацији.

6.12.1 Дефинисање проблема

Да би постигао комплетан задатак програмер треба да одреди део задатка који процесор мора да уради. На пример: шта је неопходно да би процесор контролисао индустријски процес, водио комуникацију централног и удаљеног рачунара?

Дефинисање проблема захтева прво да се одреде број, тип и карактеристике улаза и излаза система. Програмер треба да постави питања као што су:

1. Који сигнал ће рачунар да прими? Које су карактеристике тих сигнала?
2. Да ли ове сигнале генерише човек или их генеришу електронска кола?
3. Шта треба урадити ако има грешака у улазном сигналу?
4. Какав излазни сигнал рачунар мора да произведе? Које су жељене карактеристике ових излаза?

Пошто многи процесори у апликације укључују људску интеракцију, програмер мора да размотри следећа питања:

1. Како ће оператер да унесе податке или управљачке информације?
2. Које грешке обично оператер може да направи?
3. Који би начин представљања излаза највише одговарао оператеру?
4. Да ли је оператеру систем једноставан за коришћење?

Програмер одређује како процесор треба да обради улазне податке да би добио жељене излазне податке, водећи рачуна о временским и меморијским ограничењима. На пример, у неким апликацијама може бити важније обрадити податке што је могуће брже независно од величине меморије која се захтева. Може се захтевати употреба табела уместо рачунских операција дугачких по трајању. Друге апликације можда захтевају ограничавање меморије да би сачувале ниску цену хардвера.

6.12.2 Пројектовање програма (прављење дијаграма тока)

После дефинисања проблема програмер ће генерално да подели проблем у мале задатке и одреди секвенце у којима ће ови задаци бити извршени. У овој фази процеса веома корисно средство је развој дијаграма тока. Он помаже програмеру да види логику коју програм мора да следи, и одређује потпрограме за задатке који се понављају.

Важно је да се у овој фази програмер одупре изазову да пише инструкцијске кодове. Пројектовање програма мора се потпуно завршити пре него што се програм стварно напише.

6.12.3 Програмско кодирање

Једном када је дијаграм тока пројектован програмер га може

претворити у инструкцијске кодове. Кодирање је писање програма у формату који рачунар може директно да разуме, или може да га преведе. Ако је програм на асемблерском језику, или на вишим програмским језицима (као C или C++) пре него што га рачунар извршава мора да се преведе у машински језик. Асемблерски изворни програм конвертује се у објектни програм на машинском језику користећи други програм који се зове асемблер. Асемблер већ постоји у рачунарској меморији; он може да буде у ROM меморији, или може да буде учитан са диска у RAM меморији. Подсетимо се да се програм писан у вишим програмским језицима конвертује у машински језик преко интерпретера или преводиоца.

6.12.4 Тестирање и проналажење грешака

У процесу развоја програма фаза тестирања и проналажење грешака¹ захтева највише времена. Сви програми (чак и најједноставнији) имаће једну или више синтаксних или логичких грешака. Синтаксне грешке се јављају када програмер не поштује неко од правила програмерског језика који се користи (нпр. недостатак \$ испред хексадецималне вредности у асемблерском језику). Логичке грешке су грешке у пројектовању програма; програм не крши никаква синтаксна правила, али не ради оно што би требало. На пример: програмер је заборавио да напуни регистар процесора пре скока у потпрограм за генерисање кашњења, који ту вредност користи. Откривање ових грешака је некад теже од синтаксних пошто рачунар нема начина да сазна шта је програмер желео да програм уради.

Постоји неколико техника за откривање грешака у извршавању програма, које омогућавају тражење грешака у програмској логици. Једна од уобичајених је корак по корак². У оваквом начину рада процесор извршава само по једну инструкцију, док не добије команду за извршавање следеће. Ово дозвољава оператору да тестира садржај регистара процесора пре него што пређе на извршавање следеће инструкције, меморијских локација, статуса улазно излазних портова.

¹ *Debugging*

² *Step by step*

Једнокорачни начин има ману што је спор. Поред тога не може се употребити за тестирање програма који генеришу кашњења, или улазно излазне операције, већ само за кратке секвенце програма.

Друга техника за откривање грешака је са прекидним тачкама¹. Прекидна тачка је тачка у програму где процесор аутоматски зауставља извршавање програма и грана се на специјалне потпрограме за прекидне тачке, који су део системског програма за надгледање. Прекидни потпрограми су инструкцијске секвенце које обично раде следеће:

1. Смештају текуће вредности регистара микропроцесоа у RAM меморију, где их може програмер испитивати, приказати на мониторима, или штампати ради лакшег испитивања.
2. Чека да програмер обрише прекидне тачке и омогући процесору повратак на извршавање главног програма.

Програмер може да дозволи процесору да извршава сегменте програма пуном брзином док не стигне до прекидних тачка, после чега се садржај регистара процесора и било које меморијске локације испитују да би се видело да ли програмски сегмент тачно ради.

Процесор MC68HC11 има специјалну инструкцију која се зове софтверски прекид, SWI, који може да се понаша као прекидна тачка. SWI је једнобајтна инструкција са операционим кодом \$3F. Програмер може да убади овај операциони код у било коју тачку у програму где се захтева прекидна тачка. Процесор реагује на SWI на исти начин као да је добио сигнал прекида од спољашњег уређаја.

Други начин за откривања грешке је логички анализатор, који је инструмент сличан осцилоскопу. Он може да запамти и прикаже логичке нивое на свим магистралама микрорачунара у одређеном броју циклуса. Може да прикаже информације у форми временских дијаграма или бинарних и хексадецималних табела.

Логички анализатор је посебно користан у откривању премашења, погрешних сигналних секвенци, грешака у временској расподели сигнала и програмским секвенцама. Иако је веома корисно средство за изналажење грешака, његова висока цена ограничава његову широку примену.

¹ Break points

6.12.5 Програмска документација

Документација укључује неколико форми описних докумената који омогућавају кориснику да разуме рад и логику програма. Такође му омогућава да направи библиотеку докумената, који могу да олакшају будуће програмерске задатке. Уобичајена форма документације укључује дијаграме тока, листање изворног¹ и објектног програма², коментаре, меморијску мапу и шеме кола (уколико је то потребно).

6.13 ПИТАЊА И ЗАДАЦИ

1. Нацртати програмски модел процесора MC68HC11. Означити сваки регистар и његову величину.
2. Који битови регистра услова процесора MC68HC11 могу бити постављени и обрисани?
3. Објаснити разлику између директног адресирања и проширеног адресирања.
4. Зашто постоји 308 различитих операционих кодова ако MC68HC11 скуп инструкција садржи само 108 различитих врста инструкција?
5. Побројати и описати класификацију MC68HC11 инструкција.
6. Испитати инструкцијску секвенцу. На коју меморијску адресу ће бити смештен садржај акумулатора А?

LDX #\$0101

TXS

PSHA

7. Написати инструкцијску секвенцу која учитава податак са улазног порта са адресе A000 и брише битове 5 (b_4) и 4 (b_3) без утицаја на остале битове податка.
8. а) Када је S бит у регистру услова постављен, коју функцију извршава STOP инструкција?
б) Коју функцију извршава STOP инструкција када је S бит у регистру услова обрисан.
9. Ако је штампач повезан преко паралелног порта штампача на адреси

¹ Source

² Object

\$8000, којом се инструкцијском секвенцом на штампач шаље ASCII карактер '0'?

10. Нека је садржај акумулатора [A]=\$40 и акумулатора [B]=\$10. Који ће садржај акумулатора А бити након извршавања инструкције ABA и коју ће вредност имати С бит у регистру стања?
11. Ако је садржај акумулатора [A]= \$80 и С бит у регистру стања је постављен на вредност 1, који је садржај акумулатора А и која је вредност N бита после извршавања инструкције RORA ?
12. Ако је садржај индекса регистра [X]= \$0100, на коју меморијску локацију се уписује садржај акумулатора А после извршавања инструкције STAA 1,X?
13. Ако је садржај индекса регистра [X]=\$1000 са које локације се учитава податак код извршавања низа инструкција:
 INX
 LDAA \$4,X
14. Којом инструкцијом се могу инвертовати појединачни битови акумулатора А?
15. Инструкцијом ANDA #\$01 издвојен је први (b_0) бит податка у акумулатору А. Која инструкција ће обезбедити гранање само под условом да први бит има вредност 0?
16. Која инструкција обезбеђује изоловање трећег бита (b_2) акумулатора А без промене садржаја акумулатора А?
17. Где ће се наставити извршавање програма после низа инструкција:
 CLRA
 BEQ NAS1
 BNE NAS2
 BRA NAS3
18. Колика је дужина (у бајтовима) инструкција које користе имплицитно адресирање?
19. Која врста адресирања се користи код инструкције ADDA #\$12?
20. Која врста адресирања се користи код инструкције BNE NST?

7. Улазно/излазни начин рада

У многим микрорачунарским апликацијама процесор комуницира са различитим улазно/излазним уређајима. Информације које се размењују између процесора и улазно/излазних уређаја могу се класификовати било као подаци, било као команде. Подаци су типично нумеричке или алфанумеричке информације претворене у одговарајући бинарни код као што су BCD или ASCII. Постоје два типа управљачких информација и то су: команде из процесора и захтев за услугу од улазно/излазних уређаја. У свим микрорачунарским системима, критични део пројектовања је преношење информација између процесора и улазно/излазних уређаја. У овом поглављу биће анализирани различите методе улазно/излазних преноса које се користе у микрорачунарским системима.

7.1 ОСНОВНИ ПОЈМОВИ

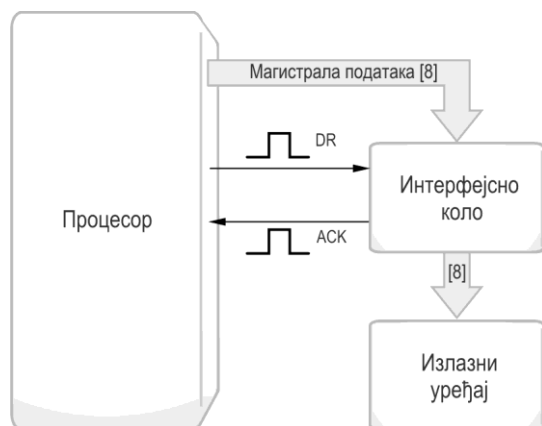
Анализа улазно/излазног рада увек претпоставља пренос података из једног уређаја у други. Користе се термини који праве разлику између уређаја који шаљу податке и уређаја који их податке. Уређај који шаље податке назива се предајник или извориште, а уређај који прима податке пријемник или одредиште.

7.2 ПРИМЕРИ УЛАЗНО/ИЗЛАЗНОГ НАЧИНА РАДА

Пре него што пређемо на детаљну анализу улазно/излазног преноса, укратко ћемо описати два једноставна примера.

7.2.1 Управљање процесима и инструментима

У применама где се процесор користи за управљање инструментима, деловима опреме или процесима, програм који се извршава стално је смештен у ROM меморији. Када се инструмент или процес укључе, генерише



Слика 7.1 Размена управљачких сигнала између процесора и излазног уређаја

се сигнал који активира $\overline{\text{RESET}}$ улаз процесора. После активирања $\overline{\text{RESET}}$ улаза¹ процесор MC68HC11 аутоматски читава садржај две узастопне локације, \$FFFE и \$FFFF. Два бајта која се добијају са ових локација шаљу се ка програмском бројачу као 16-битна адреса на којој је смештена прва инструкција. Ова два бајта називају се ресет вектор, пошто указују процесору где ће узети прву инструкцију после активирања $\overline{\text{RESET}}$ улаза. Ресет вектор може да специфицира било коју 16-битну адресу као почетну адресу програма.

Пошто процесор извршава програм смештен у ROM меморији, када је потребно да прими податке са улаза уређаја који надгледа неку променљиву физичког процеса (нпр. температуру, брзину, померање, интензитет светлости, итд.), узима ове податке и обрађује их према инструкцијама у програму. На основу резултата ових операција, процесор шаље податке или управљачке сигнале ка процесу преко одговарајућих излазних уређаја као што су дигитално/аналогни конвертори, уређаји за читавање и томе слично.

Процесор ради аутоматски и у континуитету под контролом програма смештеног у ROM меморији. Програм који извршава процесор мења се променом садржаја ROM меморије.

¹ Описано је у 5. поглављу овог уџбеника.

7.2.2 Комуникација са тастатуром и екраном

У многим ситуацијама процесор мора да комуницира са оператером. Очигледан пример да је процесор део микрорачунара је када се користи као рачунар опште намене. Други пример је електронска каса која се користи у самопослугама, где оператер даје процесору податак о шифрама појединих артикала, а процесор враћа укупан збир оператеру. Јасно је да ова и сличне апликације захтевају ефикасан и погодан начин да оператер и процесор комуницирају. Уобичајено је да се користи тастатура као улазни уређај за оператера и хексадецимални или децимални LED¹ екран као излазни уређај којим управља процесор.

У ROM меморији смештени су програми за надзор тастатуре које процесор извршава после активирања RESET улаза. Тастатура и њена интерфејсна кола имаће специфичну адресу, или адресе, које су јој додељене, а које се заснивају на меморијској мапи система, као што је описано у 5. поглављу. Програм за надзор тастатуре доводи до тога да процесор непрекидно читава тастатуру како би детектовао активности на њој. Када се детектује активност, процесор читава податак са тастатуре и одређује који тастер је притиснут, а затим извршава одговарајуће инструкције.

Код микрорачунара опште намене, оператер може да шаље различите типове информација са тастатуре. На пример, оператер може да користи хексадецимално кодирану тастатуру. Уколико се притисне тастер у хексадецималном коду за неку инструкцију, процесор под контролом програма за надзор читава кодове са тастатуре и смешта их у RAM меморију. Област RAM меморије где се програм смешта одређује сам оператер. Када се комплетан програм смести у RAM меморију оператер налаже процесору да извршава програм постављањем почетне адресе и активирањем специфичног тастера.

Када микрорачунар заврши извршавање програма оператера, може се вратити на извршавање програма за надзор тастатуре помоћу RESET сигнала, који се обично активира тастером са тастатуре. Микрорачунар је онда спреман да прихвати нове информације од оператера. Наравно, пошто микрорачунар извршава оператерски програм мора на неки начин

7. Улазно/излазни начин рада

¹ *Light Emitting Diode* - диоде које емитују светлост

¹ Описано је у 5. поглављу овог уџбеника.

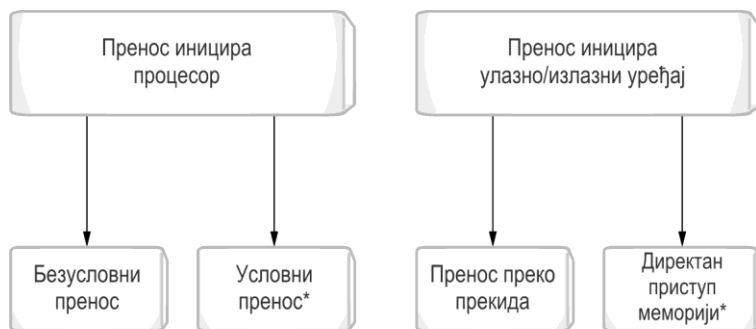
7. Улазно/излазни начин рада

да шаље информације и резултате ка оператеру. За приказивање се користе LED и LCD¹ екрани. Њима и одговарајућим интерфејсним колима биће додељене одређене адресе према усвојеној меморијској мапи система. Програм садржи инструкције које приступају тим адресама и на тај начин процесор уписује податке на екран. Често се екран користи за надзор тастатуре тј. за приказивање тастера који се активирају тако да оператер то може да види.

Ова два примера показују на који начин се може остварити комуникација процесора и улазно/излазних уређаја. У овом поглављу са више детаља анализираће се прикључивање периферија.

7.3 РАЗЛИЧИТЕ ВРСТЕ УЛАЗНО/ИЗЛАЗНОГ ПРЕНОСА

Постоји неколико различитих начина на које улазно/излазни пренос може да буде инициран и контролисан. На пример пренос података између процесора и улазно/излазних уређаја може да иницира процесор или улазно/излазни уређај. Једном када је улазно/излазни процес инициран, њиме може да управља или процесор или улазно/излазни уређај. Основна подела приказана је на слици 7.2.



* Захтева размену управљачких информација између уређаја и процесора

Слика 7.2 Основне варијанте улазно/излазног преноса

¹ Liquid Crystal Display

Када улазно/излазни пренос иницира процесор преносом се управља програмски пошто је процесор „прозван” да изврши једну или више инструкција које преносе податке између њених унутрашњих регистара (нпр. акумулатора) и периферијских уређаја. Овај пренос који иницира процесор може да се изврши безусловно, ако је улазно/излазни уређај увек спреман за комуникацију. Условни пренос захтева размене управљачких сигнала између процесора и улазно/излазних уређаја пре преноса података.

Постоје два начина преноса које иницира улазно/излазни уређај. Сваки од њих захтева размену управљачких сигнала између улазно/излазних уређаја и процесора. Пренос са прекидом¹ (интераптом) иницира улазно/излазни уређај који шаље сигнале ка улазима за прекид ($\overline{XIRQ}$, $\overline{IRQ}$) процесора MC68HC11 што је знак да је уређај спреман за пренос података. Процесор прекида извршавање програма на коме тренутно ради и започиње са извршавањем новог програма који се зове програм за обраду (опслуживање) прекида², а који садржи инструкције за пренос података ка и од улазно/излазних уређаја. Директан приступ меморији (DMA³) је друга врста преноса коју иницира периферијски уређај и једина је врста преноса која није под директном контролом (програма) процесора. Пренос података одвија се преко специјалних интерфејсних кола - контролера за директан приступ меморији (DMAC⁴). Процес се назива директан приступ меморији зато што се подаци преносе између улазно/излазних уређаја и RAM меморије директно без посредовања процесора.

7.4 БЕЗУСЛОВНИ УЛАЗНО/ИЗЛАЗНИ ПРЕНОС

Овај тип преноса користи се када је излазни модул увек спреман да прихвати податке из процесора, или када улазни модул има увек спремне податке за процесор. Тада не постоји потреба за разменом управљачких сигнала између процесора и улазно/излазних уређаја. Процесор једноставно извршава инструкције за уписивање података или за читавање података из улазно/излазних уређаја.

¹ *Interrupt*

² *Interrupt Service Routine*

³ *Direct Memory Access*

⁴ *Direct Memory Access Controller*

7.4.1 Процесор шаље податке ка LED екрану

Процесор шаље 8-битне податке за приказивање на екрану операцијом уписивања на излазни порт. Један од могућих метода приказан је на слици 7.3, где се осам LED диода¹ користи за приказивање једног бајта података који се шаљу из процесора. Инвертор са отвореним колектором и LED диода додати су HCMOS излазном порту (описано у 4. поглављу, и представљено сликом 4.17). На тај начин може се обезбедити визуелно представљање података који се уписују на овом излазном порту. Процесор шаље податке ка излазу користећи операцију уписивања на адресу \$A000. Када процесор шаље (уписује) податке ка излазном порту, подаци ће се појавити на Q излазу флип-флоп регистра. Они излази који су на високом нивоу довешће излазе инвертора са отвореним колектором на низак ниво, омогућити проток струје кроз одговарајуће LED диоде, и оне ће емитовати светлост. Подаци уписани у регистар везан за овај порт остаће уписани док процесор не упише нове податке у тај регистар.

Садржај било ког од интерних регистра процесора може се приказати на екрану на исти начин користећи одговарајућу асемблерску инструкцију. На пример, садржај акумулатора A може се приказати користећи инструкцију STAA \$A000.

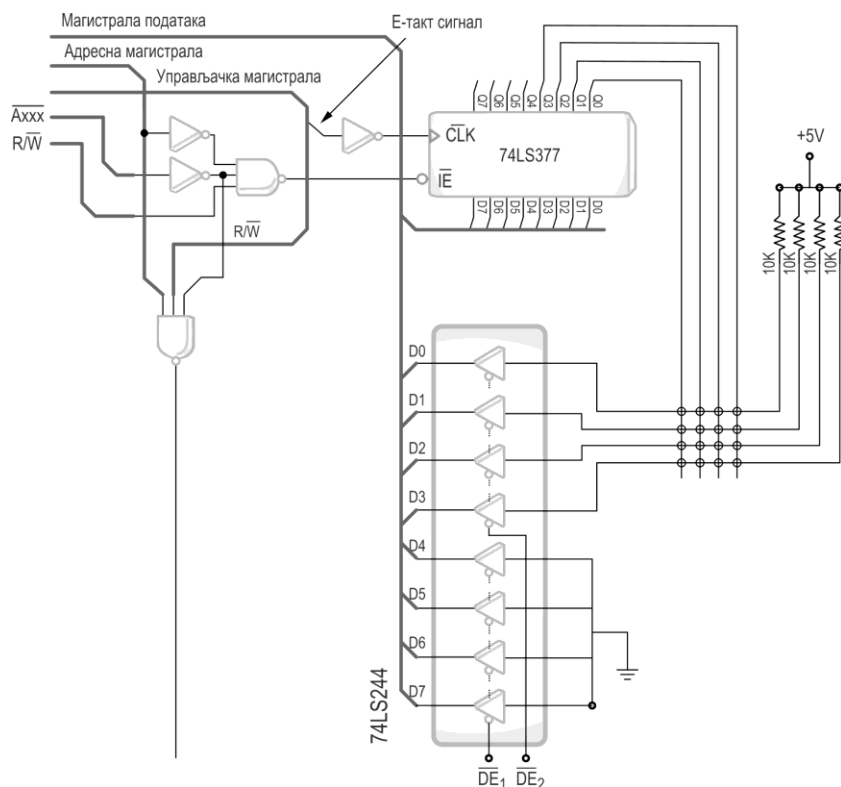
Коришћење LED диода (или екрана) за приказивање података у бинарној форми мање је пожељно него приказивање у хексадецималној форми, па се користи специјални хексадецимални LED екран са декодером и драјвером у истом интегрисаном колу.

7.4.2 Једноставан улазни порт

Други пример безусловног преноса који иницира процесор је једноставан улазни порт који се користи за унос 4-битног податка у процесор. Део управљачког програма користи се за испитивање статуса прекидача и на основу њих се извршавају одговарајуће активности. На слици 7.3 представљен је интерфејс за повезивање тастатуре² и процесора. На излазном порту на адреси \$A000 постављен је податак 0 асемблерским инструкцијама: CLRA, STAA \$A000.

¹ Користићемо термин LED диода. Правилније би било LED пошто скраћеница већ садржи енг. реч *diode*.

² Користи се термин „сканирајућа тастатура“ пошто се стања појединих тастера секвенционално испитују.



Слика 7.3 Веза процесорске јединице и тастатуре

На слици 7.3 види се да су на улазном порту на бафере са три стања додати прекидачи и одговарајући отпорници. Процесор може да прикупи податке о стању прекидача у било ком тренутку једноставном инструкцијом читавања са адресе \$A001. Ово се може постићи асемблерском инструкцијом LDAA \$A001. Ова инструкција ће проузроковати да процесор чита вредност улазних прекидача и смести их у акумулатор. Горња четири бита¹ увек су на 0, пошто су четири горње улазне линије бафера стално везане на 0. Садржај доња четири бита² зависи од тренутног стања тастера. Уколико ниједан тастер није притиснут садржај акумулатора ће бити \$0F.

¹ Среће се и под називом горњи нибл.

² Доњи нибл

7. Улазно/излазни начин рада

У случају да је очитана вредност различита од \$0F прелази се у програм за проверу стања тастатуре да би се једнозначно одредило који је тастер притиснут. Потпрограм за проверу стања тастатуре је:

```
SKAN TAST LDAB $07
          STAB $A000 ; избачена 0 на четврту колону
          LDAB $A001 ; очитавање стања тастера
          CMPB      ; да ли је неки тастер притиснут
          BEQ  NAS1
          TBA
          ORAA #$80
          RTS

NAS1
          STAB $A000 ; избачена 0 на трећу колону
          LDAB $A001 ; очитавање стања тастера
          CMPB      ; да ли је неки тастер притиснут
          BEQ  NAS2
          TBA
          ORAA #$40
          RTS

NAS2
          STAB $A000 ; избачена 0 на другу колону
          LDAB $A001 ; очитавање стања тастера
          CMPB      ; да ли је неки тастер притиснут
          BEQ  NAS3
          TBA
          ORAA #$20
          RTS

NAS3
          STAB $A000 ; избачена 0 на прву колону
          LDAB $A001 ; очитавање стања тастера
          CMPB      ; да ли је неки тастер притиснут
          TBA
          ORAA #$10
          RTS
```

Кôд податка о притиснутом тастеру налази у акумулатору А и добија се тако што се у горња четири бита акумулатора на 1 поставља бит који одговара колони која се тренутно проверава, а у доња четири бита смешта се вредност очитана са улазног порта.

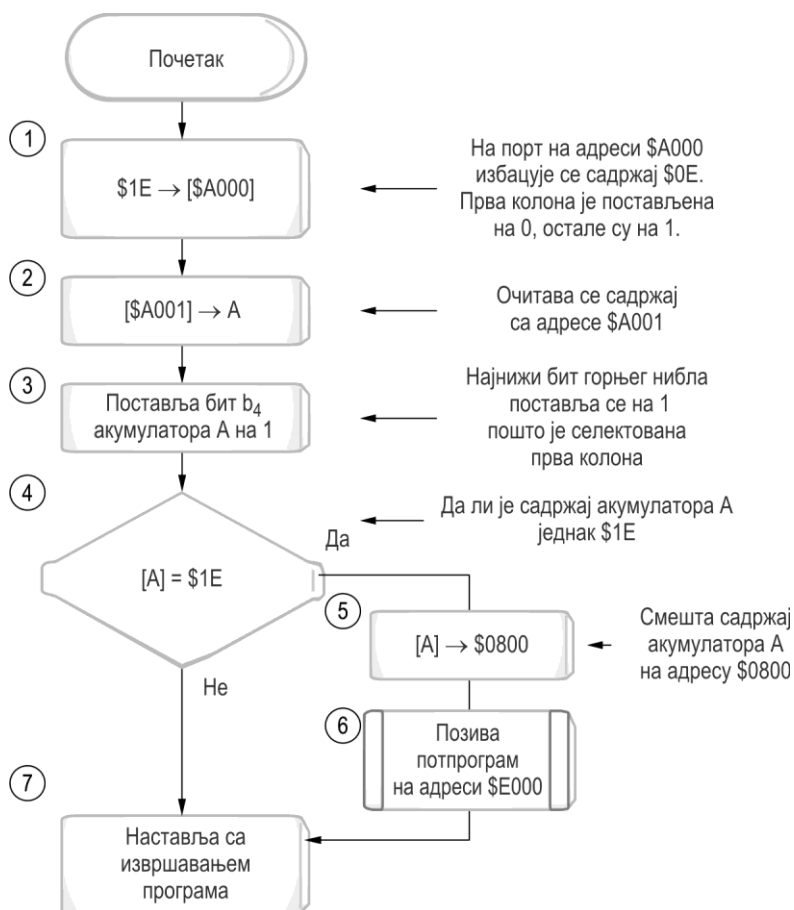
Када су подаци учитани у акумулатор А могу се користити за различите намене. У неким случајевима податак са тастера представља нумеричку вредност над којом се извршавају аритметичке операције. У другом случају тастер може да представља врсту кода који програм испитује да би видео коју следећу инструкцију треба да изврши. Да бисмо то илустровали посматрајмо ситуацију када процесор треба да изврши потпрограм на адреси \$E000 када је притиснут тастер који се налази у првој врсти и првој колони. Уколико је притиснут овај тастер садржај акумулатора А ће бити \$1E. У овом случају не мора се извршити потпуна провера стања тастатуре већ само део програма на лабели NAS3.

1. на прву колону избацује се вредност 0;
2. очитава се податак са тастера;
3. поставља се 5. бит (б₄) акумулатора А на вредност 1;
4. испитује се очитана вредност;
5. уколико је садржај акумулатора \$1E, смести се садржај тастера у меморију на локацију \$0800, пре него што се позове потпрограм;
6. позива се потпрограм на адреси \$E000;
7. наставља се са извршавањем главног програма.

Дијаграм тока једног од начина за реализацију ових операција је приказан на слици 7.4.

Асемблерски програм који одговара дијаграму тока са слике 7.4

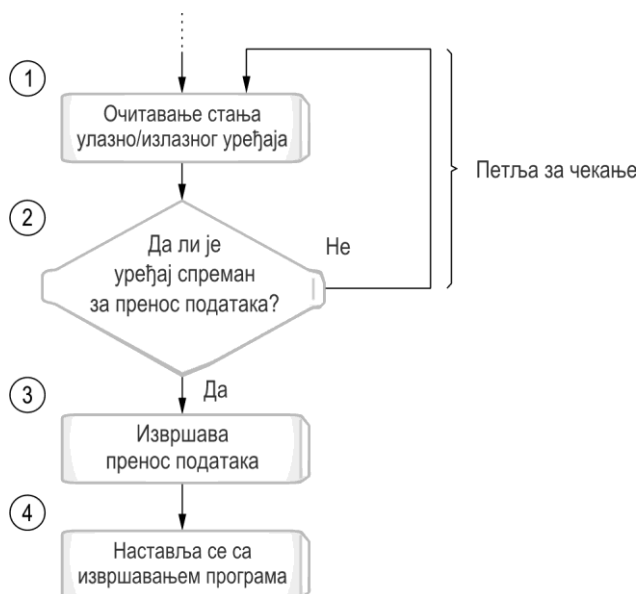
```
LDAB  #$0E
STAB  $A000    ;избачена 0 на прву колону
LDAA  $A001    ;очитавање стања тастера
ORAA  #$10
CMPA  #$1E
BNE   NASTAVI
STAA  $800
JSR   $E000
NASTAVI  NOP          ; Наставак главног програма
```



Слика 7.4 Дијаграм тока програма који испитује стање тастера (притиснут/није притиснут) у првој врсти и првој колони тастатуре

7.5 УСЛОВНИ УЛАЗНО/ИЗЛАЗНИ ПРЕНОС

У овој врсти улазно/излазног преноса процесор мора да одреди да ли су периферијски уређаји спремни за комуникацију пре него што се обави пренос података под програмском контролом. Ово је тростепени процес где процесор мора да очита статусне информације од периферијских уређаја, испита овај статус, види да ли је уређај спреман, и када је спреман



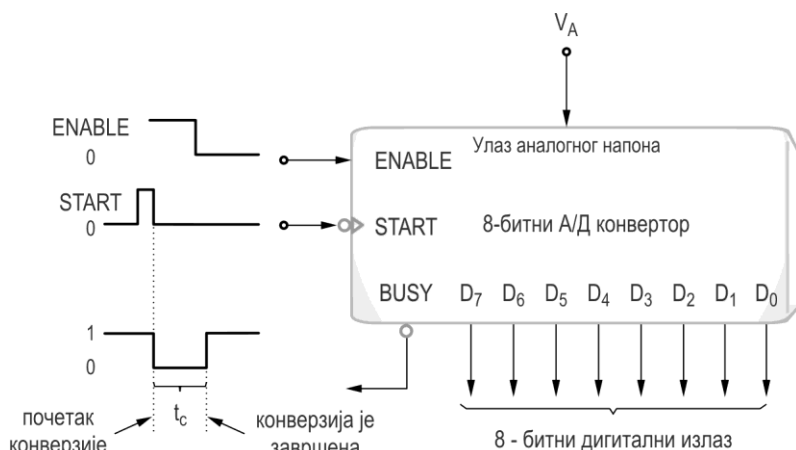
Слика 7.5 Дијаграм тока за програм код условног преноса

обавља пренос података. Ова врста улазно/излазног преноса уобичајено се назива улазно/излазни пренос са упитом¹, пошто уређај треба стално проверавати (питати) да би се видело да ли је спреман. Овај скуп корака приказан је дијаграмом тока на слици 7.5.

7.5.1 Аналогно/дигитални конвертор са три стања

У процесном управљању постоји велики број претварача који претварају физичку величину, као што је температура или интензитет светлости, у напон пропорционалан тој физичкој величини. Овај напон је аналогна величина која може да има било коју вредност у оквиру задатог опсега, као што је то нпр. 0 до 10V. Да би се обезбедио интерфејс између аналогног напона и дигиталног контролера напон се мора преко аналогно/дигиталног (А/Д) конвертора претворити у своју дигиталну презентацију. Слика 7.6 показује једну врсту аналогно/дигиталног конвертора, који се једноставно прикључује на процесор.

¹ *Polled* - проверка (испитивање, упит)



Слика 7.6 Уобичајена врста аналогно/дигиталног конвертора који се користи у микрорачунарским системима процесног управљања

Ова врста аналогно/дигиталног конвертора претвара аналогни напонски улаз (V_A) у 8-битни дигитални излаз (D_7 - D_0) који може да буде бинарна или BCD вредност. Импулс који се доводи на START улаз А/Д конвертора покреће процес конверзије. Трајање конверзије зависи од метода конверзије који конвертор користи. То време, које се назива време конверзије, t_c , може да буде од неколико наносекунди (ns) до неколико десетина милсекунди (ms). У току времена t_c , док се одвија процес конверзије, BUSY излаз А/Д конвертора биће на ниском нивоу.

Кад се заврши конверзија низак ниво на излазу враћа се на висок ниво. Дигиталне излазне линије везане су за леч (са три стања) који је део А/Д конвертора. Низак ниво на ENABLE улазу омогућиће да се дигитална презентација улазног аналогног сигнала (V_A) појави на излазу за податке. Висок ниво на ENABLE улазу поставиће излазне линије у стање високе импедансе.

Обично се ENABLE улаз поставља на низак ниво када BUSY излаз (висок напонски ниво) укаже да је конверзија завршена. Уколико се за време конверзије (t_c интервала) на ENABLE улаз доведе низак напонски ниво на излазним линијама А/Д конвертора појавиће се резултат претходне А/Д конверзије.

7.5.2 Интерфејс А/Д конвертора и процесора (условни пренос)

У уобичајеним апликацијама, улаз у А/Д конвертор је аналогни напон V_A , који генерише мерни претварач и његова пратећа кола. Дигитални излаз А/Д конвертора може се учитати у процесор одговарајућом инструкцијом. Процесор ће обрадити дигиталну вредност аналогног напона програмом који је смештен у његовој меморији, генерисати одговарајуће управљачке излазне сигнале и послати их процесу. Ове операције захтевају комуникацију између процесора и А/Д конвертора која се може описати на следећи начин:

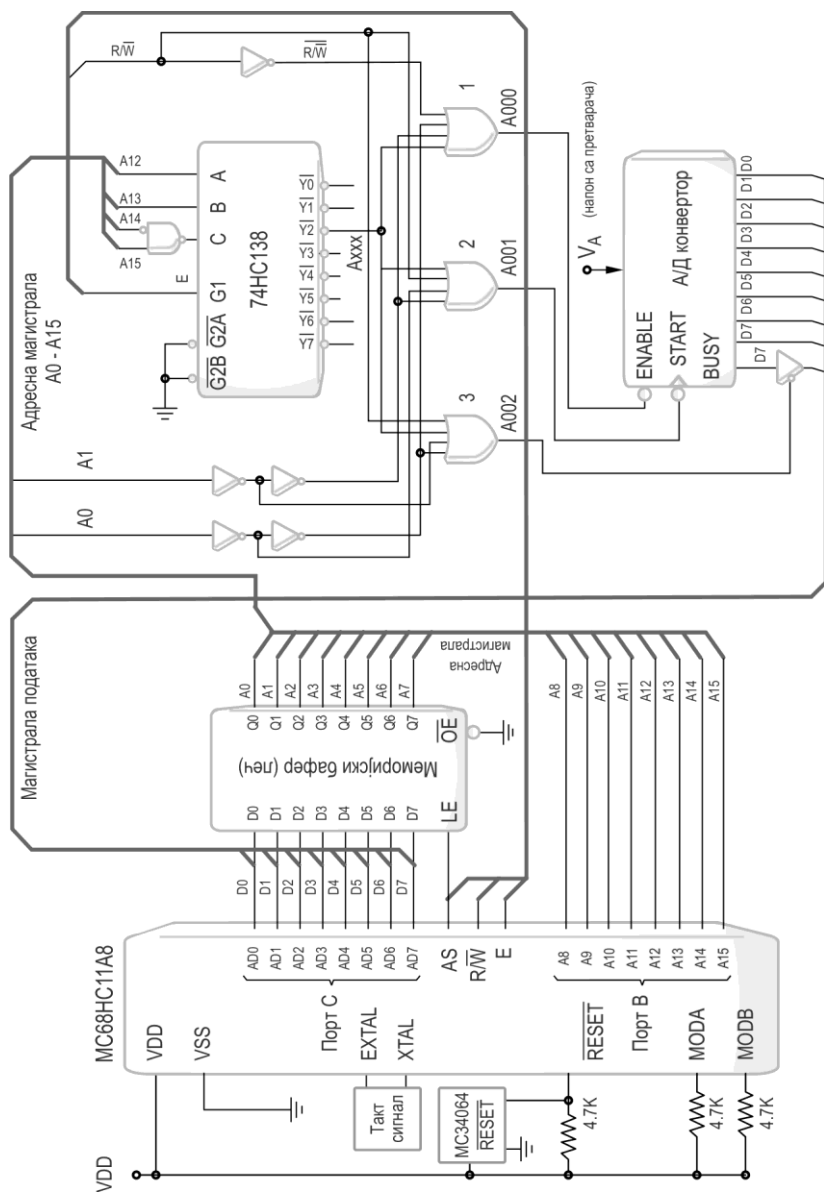
1. процесор шаље START импулс као знак А/Д конвертору да започне претварање (конверзију) текуће вредности аналогног напона (V_A) у његов дигитални еквивалент;
2. процесор континуално читава и тестира статус BUSY излаза А/Д конвертора. Када се BUSY излаз поново врати на висок ниво, значи да је конверзија завршена;
3. процесор читава дигитални излаз А/Д конвертора у један од својих интерних регистара.

Ове секвенце захтевају да се три различита управљачка сигнала измеђују између процесора и А/Д конвертора:

1. сигнал за почетак А/Д конверзије (START импулс),
2. сигнал за читавање BUSY излаза да би га процесор анализирао у својим интерним регистрима,
3. сигнал за пренос дигиталног излаза А/Д конвертора у процесор.

Једна реализацију повезивања А/Д конвертора и процесора (са условним преносом) приказана је на слици 7.7.

Излази А/Д конвертора (D_7 - D_0) везани су на магистралу података, тако да могу бити пренети у процесор када је излаз ИЛИ1 логичког кола (ENABLE улаз А/Д конвертора) активан. Преко бафера са три стања BUSY излаз везан је на линију магистрале података највеће тежине (D_7). Бафер са три стања селекује се напонским нивоом логичке нуле са излаза ИЛИ3 логичког кола. Тада се BUSY излаз може учитати у процесор. Импулс за почетак конверзије (START улаз А/Д конвертора) генерише се када процесор адресира ИЛИ2 логичко коло.



Слика 7.7 Повезивање А/Д конвертора и процесора коришћењем условног (са упитом) улазно/излазног преноса

Одабрана је $\overline{A_{xxx}}$ као линија за селектовање А/Д конвертора. Потребне су и две линије мање тежине A_0 и A_1 , да би се активирало једно од три ИЛИ кола. У даљем тексту анализираћемо рад ових кола.

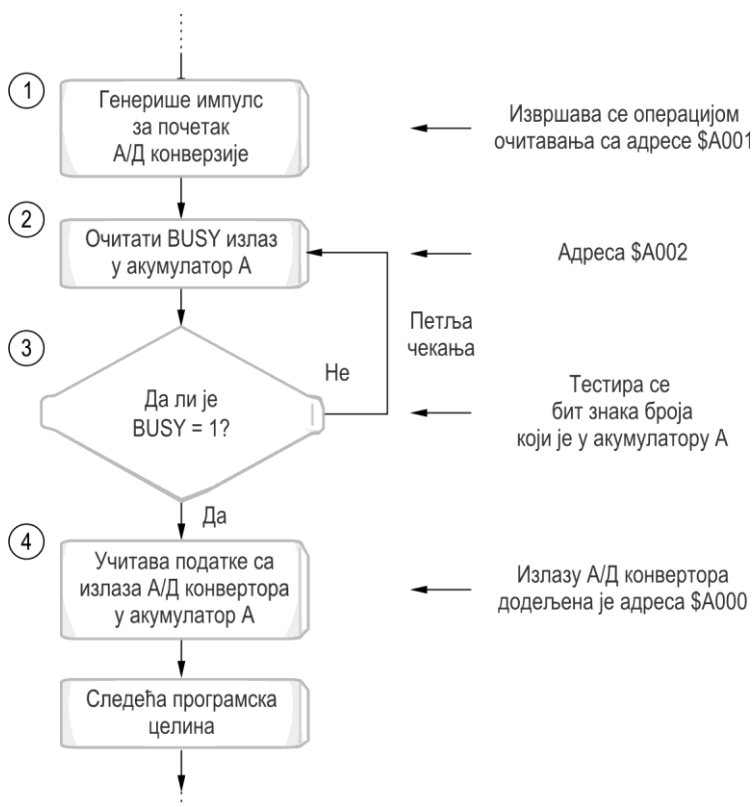
ИЛИ1 коло. Његови улази су A_0 , A_1 , и $\overline{R/\overline{W}}$, $\overline{A_{xxx}}$ и његов излаз ће бити на 0 када су сви ови улази 0. Ова ситуација се јавља када процесор извршава операцију читавања са адресе \$A000 (A_1 и A_0 су 0). То омогућава да се излази А/Д конвертора преносе у процесор преко магистрале података. Инструкција LDAA\$A000 обезбедиће смештање података из А/Д конвертора у процесор.

ИЛИ2 коло има улазе $\overline{A_0}$, A_1 , $\overline{R/\overline{W}}$ и $\overline{A_{xxx}}$ и његов излаз прећиће на нижи логички ниво кад год процесор извршава операцију читавања са адресе \$A001 ($A_1=0$ и $A_0=1$). Ова операција читавања у ствари је празно читавање (пошто се ништа не ставља на магистралу података). Она се једноставно користи као начин за генерисање импулса код читавања са адресе \$A001 којом се започиње А/Д конверзија. Инструкцијом LDAA\$A001 оствариће се генерисање START импулса за А/Д конвертор. Када ниједан уређај не поставља податке на магистралу података она је у „пливајућем” стању. Код већине процесора, кад је магистрала података пливајућа, за време операције читавања, недефинисани подаци ће бити учитани у интерне регистре процесора. Ово није никакав проблем пошто процесор не користи те податке.

ИЛИ 3 коло. Оно има улазе A_0 , $\overline{A_1}$, $\overline{R/\overline{W}}$, $\overline{A_{xxx}}$ и његов излаз ће прећи на нижи ниво кад процесор извршава читавање са адресе \$A002. Овим се поставља BUSY излаз на магистралу података D_7 за пренос у процесор. Уочимо да ће других седам линија магистрале бити у пливајућем стању. Процесор тестира само статус BUSY излаза а стање на осталим линијама није битно. Инструкција LDAA \$A002 ће учитати стање BUSY излаза у акумулатор A (на пример $[A] = D_7xxxxxx$).

Анализираћемо програм који обезбеђује пренос података из А/Д конвертора у процесор. На слици 7.8 приказан је дијаграм тока тог програма.

Видимо да када процесор долази до тачке у програму где захтева податке из А/Д конвертора, започиње слањем импулса за почетак конверзије (блок 1) тј. извршавањем празне инструкције за читавање



Слика 7.8 Дијаграм тока који показује активности које процесор извршава да би добио податке из А/Д конвертора

са адресе \$A001. Процесор затим наставља непрекидно да очитавати и тестира BUSY излаз да би одредио да ли је 1 (блок 2 и 3). Понавља процес док BUSY излаз не постане висок. Пошто је BUSY бит везан за магистралу података D7, биће учитан на 8-битску позицију акумулатора А, тј. позицију бита знака. Може се тестирати коришћењем инструкције условног гранања као што је „гранај се ако је позитиван” (BPL испитује N бит и, уколико је 0, гранаће се на траг на блок 2). Уколико је N бит 1 (указујући да је BUSY=1), програм се наставља извршавањем 4. блока: инструкције за пренос излаза А/Д конвертора у акумулатор А.

START	PSHA	; смести садржај акумулатора А на стек меморију
	TPA	; пренеси садржај регистар услова у акумулатор А
	PSHA	; смести садржај регистра услова на стек меморију
	LDAA \$A001	; генериши START импулс
TSBUSY	LDAA \$A002	; тестирање BUSY бита
	BPL TSBUSY	; гранај се натраг на TSBUSY
UPOD	LDAA \$A000	; смести излаз А/Д у акумулатор А
	STAA \$0050	; смести акум. А (излаз А/Д конв.) у меморију (\$50)
	PULA	; регистар услова са стек меморије смести у акум. А
	TAP	; пренеси акумулатор А у регистар услова (А → CCR)
	PULA	; скини акумулатор А са стек меморије
KPOD	RTS	; повратак из потпрограма

У програму који се односи на прикупљање података из А/Д конвертора постоје многе ситуације када је неопходно узети нове податке. Ово се може постићи потпрограмом. Постављањем ових секвенци инструкција у потпрограм програмер их може користити преко JSR инструкције у било ком тренутку кад је прикупљање података потребно. Пошто ова секвенца инструкција користи акумулатор А, потребно је сачувати и његов садржај и садржај регистра услова.

7.5.3 Недостаци условног улазно/излазног преноса

Пример који је описан показује основне карактеристике условног или улазно/излазног преноса са упитом. Процесор треба да чека на улазно/излазни уређај. У неким случајевима, нарочито код спорог улазно/излазног преноса, процесор ће губити доста времена у читавању и провери стања (статуса) улазно/излазног уређаја. У многим апликацијама, док чека, процесор може да ради друге послове као што је обрада података или комуникација с другим уређајима. У следећем одељку видећемо како се овај проблем може разрешити коришћењем другог начина преноса од улазно/излазног уређаја до процесора.

Софтвер са упитом може се користити у апликацијама у реалном времену¹ када је величина и сложеност система мала.

¹ Real time

7.6 ПРЕНОС КОЈИ ИНИЦИРА УЛАЗНО/ИЗЛАЗНИ УРЕЂАЈ - ПРЕКИД

Постоје многи разлози који наводе да се анализирају прекиди које иницирају улазно/излазни уређаји и да се анализира њихово усаглашавање (синхронизација). Први разлог је тај што софтвер у системима у реалном времену мора да одговори на хардверске догађаје у оквиру прописаног времена.

Прекид даје могућност да се реагује и одговори на догађаје који нису учестали али који су веома важни. На пример аларм као што је нека грешка у систему или истрошена батерија могу се одрадiti преко прекида. Периодични прекиди које генеришу часовници у одређеним временским интервалима неопходни су системима за прикупљање података¹ и управљање².

7.6.1 Основни принципи

Прекид је догађај који изазива аутоматски прелаз са извршавања текућег програма на потпрограма за обраду прекида као одговор на хардверски догађај који није ни на који начин синхронисан са извршавањем текућег програма. Хардверски догађај може да изазове било спољашњи улазно/излазни уређај (као што је SCI³ улаз/излаз) или интерни догађај као што је грешка у операционом коду или оглашавање неког часовника. Потпрограма за обраду прекида (ISR⁴) је софтверски модул⁵ који се извршава када хардвер захтева прекид. Могуће је да постоји један велики потпрограма који води рачуна о свим захтевима (прекиди са упитом) или више мањих потпрограма који су намењени одређеним прекидима (векторски прекид).

То ће довести до тога да процесор одложи извршавање програма на коме тренутно ради и да пређе на извршавање потпрограма за обраду прекида (ISR). По завршетку програма за прекид процесор се враћа на програм који је извршавао у тренутку појаве прекида. Ово је илустровано сликом 7.9.

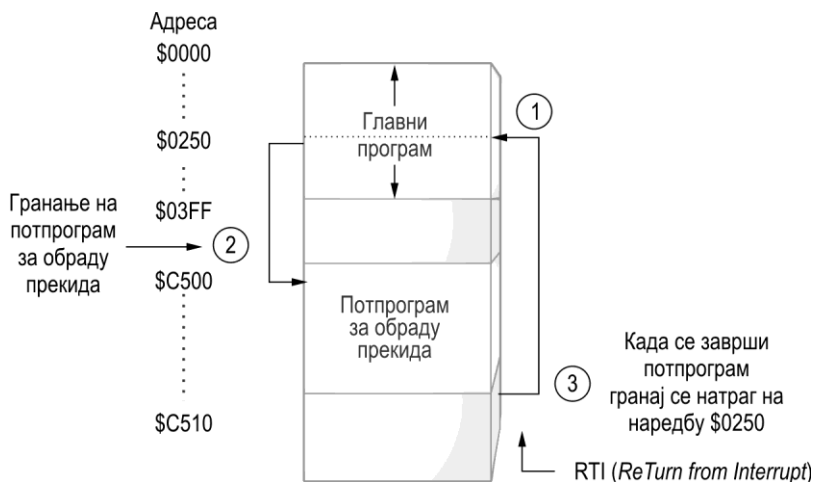
¹ *Data Acquisition Systems*

² *Control Systems*

³ *Serial Communication Interface*

⁴ *Interrupt Service Routine*

⁵ То је позив одређеног потпрограма инициран хардвером који захтева опслуживање.



Слика 7.9 Пример како прекид изазива промену у извршавању програма

На слици се види део меморијске мапе за уобичајену ситуацију. Неке меморијске локације \$0000-\$03FF (странице 00-03) садрже главни програм са инструкцијама које процесор треба да изврши. Потпрограма за обраду прекида (ISR¹) смештен је на локације \$C500-\$C510. За време нормалног рада процесор извршава главни програм. Претпоставимо да, у процесу извршавања инструкције са адресе \$0250, улазно/излазни уређај шаље сигнал прекида ка процесору (означено са 1 на слици 7.9). Процесор неће реаговати на овај прекидни сигнал док не заврши извршавање инструкције са адресе \$0250. Када заврши извршавање те инструкције садржаји регистара процесора смештају се на стек меморију. Процесор наставља са извршавањем програма (грана се) на адреси \$C500 (означено са 2 на слици 7.9) где се налази прва инструкција потпрограма за обраду прекида (ISR). Наставља са извршавањем потпрограма за обраду прекида док не дође до адресе \$C510. На њој се налази инструкција (RTI²) која указује процесору да је следећа инструкција коју ће извршити у главном програму, где је била прекинута (тачка 3).

¹ Interrupt Service Routine

² ReTurn from Interrupt

Процес са прекидима елиминише потребу да процесор троши време на непрекидну проверу да ли је улазно/излазни уређај спреман за комуникацију. Рад са прекидима је користан у апликацијама у којима се процесор везује са неколико улазно/излазних уређаја чији су захтеви за прекид асинхрони. Такви прекиди могу да наступе било када и њима се не може управљати коришћењем безусловног улазно/излазног преноса.

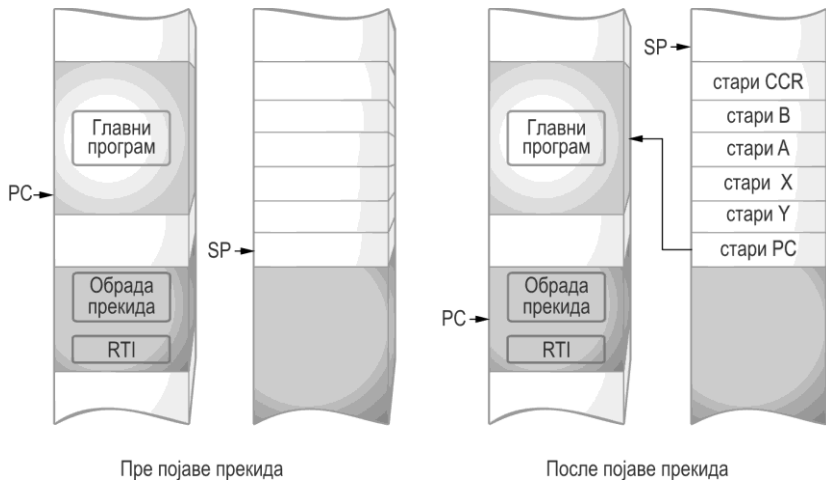
7.6.2 Повратна адреса

После извршавања потпрограма за обраду прекида (ISR) поставља се питање како процесор зна на коју адресу у главном програму треба да се врати. Пошто прекид може да наступи у било ком тренутку и нема начина да се предвиди коју инструкцију ће процесор извршавати када прекид наступи ово питање је на месту. Овај проблем аутоматски решава процесор тиме што се сачува (само привремено) текући садржај програмског бројача пре него што дође до гранања на потпрограм за обраду прекида. Процесор ће сместити садржај програмског бројача на стек меморију заједно са садржајем индекс регистара Y и X, акумулатора A и B и регистра услова (CCR). На слици 7.10 приказано је како се садржаји регистара смештају у стек меморију. Када се извршава инструкција повратка из прекида (RTI), на крају потпрограма за обраду прекида (ISR), процесор ће аутоматски учитати (повући) у интерне регистре податке са стек меморије обрнутим редоследом од редоследа уписивања. Тако ће садржаји регистра процесора бити исти као пре извршавања потпрограма за обраду прекида. Потпрограм за обраду прекида ради на принципу последњи стављен на стек меморију први се скида са стек меморије (*Last In - First Out*).

У примеру са слике 7.9 садржај програмског бројача је \$0250 када прекид наступи. Као што је већ наглашено, процесор завршава извршавање инструкције са имплицитним адресирањем¹ чији је последњи бајт на адреси \$0250, што подразумева и инкрементирање програмског бројача на \$0251. Ова вредност², \$0251, је повратна адреса коју процесор смешта у две узастопне локације у стек меморију. Треба имати на уму да је адресна

¹ Уколико је инструкција двобајтна, онда је последњи бајт инструкције на адреси \$0251, а уколико је тробајтна на адреси \$0252..

² Уколико је инструкција двобајтна, текућа вредност програмског бројача је \$0252, а уколико је тробајтна \$0253.



Слика 7.10 Како се после појаве прекида чувају садржаји регистара процесора у стек меморији

магистрала 16-битна, па су и сви адресни регистри 16-битни: програмски бројач, индекс регистара Y и X, а меморијска локација је 8-битна па су потребне две узастопне локације за смештање адресних регистара. После програмског бројача процесор ће у стек меморију ставити садржај индекс регистара X (X_L, X_H), индекс регистара Y (Y_L, Y_H), акумулатора А и В и регистра услова (CCR) у узастопне меморијске локације. Када процесор извршава RTI инструкцију (последња инструкција програма за обраду прекида) она враћа у регистре њихове сачуване вредности у стек меморији. Редослед је обрнут од редоследа при стављању у стек меморију. Последња информација која је скинута из стек меморије -повратна адреса (\$0251) смешта се у програмски бројач. Процесор узима следећу инструкцију са адресе \$0251 из главног програма.

7.6.3 Онемогућавање прекида

Шта се дешава ако улазно/излазни уређај прекида процесор, док он извршава део програма који захтева непрекидну обраду? Ово се може десити, на пример, када је процесор у временској петљи или када процесор комуницира са другим улазно/излазним уређајем. У оваквим ситуацијама, појава прекида може да доведе до нежељених резултата.

Због тога, сваки процесор може програмски да забрани (онемогући прекид). Ово се обично ради увођењем једног флип-флопа који се назива ознака (флег) за маскирање прекида (I), и инструкција које могу да поставе (I=1) или обришу (I=0). Код MC68HC11 I бит део је регистра услова (CCR).

Када се појави прекид процесор неће (или хоће) реаговати на њега у зависности од тога да ли је I бит постављен (или обрисан). Тако, програмер има могућност да забрани (онемогући) или дозволи (омогући) прекид. Ово се постиже инструкцијама SEI¹ и CLI², које се могу поставити било где у програму.

Програм који следи показује како се ове инструкције могу употребити тако да се може генерисати временско кашњење без опасности да потпрограма за обраду прекида промени време извршавања потпрограма за кашњење. Процесор неће реаговати на прекид док се CLI инструкција не изврши.

SEI	; онемогућавање прекида
LDAA #\$C0	; напунити акумулатор А бројем \$C0
DECA	; декрементирај акумулатор А
BNE \$0103	; гранај се натраг уколико је [A]=0
CLI	; омогућавање прекида
-	; наставак извршавања програма

7.6.4 Врсте захтева за прекид

У 4. поглављу видели смо да процесор MC68HC11 има две врсте спољашњих захтева за прекид: $\overline{\text{IRQ}}$ и $\overline{\text{XIRQ}}$. Многи интерни улазно/излазни уређаји могу да генеришу захтеве за прекид на основу спољашњих догађаја као што су: STRA³, часовник или серијски порт. Већина захтева за прекид привремено поставља бит I (I=1) у регистар услова (CCR) да спречи друге захтеве за прекид (укључујући и сопствени). С друге стране $\overline{\text{XIRQ}}$ захтев привремено поставља I и X битове у регистар услова. Процесор микроконтролера M68HC11A8 може да подржи:

¹ Set Interrupt Mask (постави маску за прекид)

² Clear Interrupt Mask (обриши маску за прекид)

³ Када микроконтролер ради као једно интегрисано коло (single chip mode).

- STRA прекид,
- три улазна прикључка¹ која се могу користити као генерални прикључци за прекид или за мерење временских интервала, под условом да је одабрана опција за памћење вредности часовника при промени стања на неком од ових прикључака;
- пет излазних прекида који се генеришу када се вредност часовника изједначи са вредношћу уписаном у неки од пет регистара намењених за иницирање ове врсте прекида;
- три прекида које генеришу часовници² и
- два прекида серијских портова (SCI³ и SPI⁴).

Захтеви за прекид имају тачно дефинисан приоритет али се он може променити (са нижег на виши ниво) коришћењем регистра за приоритет хардверских прекида (HPIR⁵). Уобичајено се захтев за прекид $\overline{\text{XIRQ}}$ повезује са једним уређајем највећег приоритета. Он има посебан вектор прекида (\$FFF4) и посебан бит за маскирање прекида (X). Бит X се не може софтверски поставити (програмер не може маскирати $\overline{\text{XIRQ}}$). Прекид $\overline{\text{XIRQ}}$ захтева се када је прикључак $\overline{\text{XIRQ}}$ на напонском нивоу логичке нуле и када бит X у регистру услова има вредност 0. Приликом обрада прекида $\overline{\text{XIRQ}}$ аутоматски се постављају битови I и X (I=1, X=1), а захтев за прекид не може да прекине обраду захтева за прекид $\overline{\text{XIRQ}}$.

На маскирајуће захтеве за прекиде ($\overline{\text{IRQ}}$) процесор може (ако хоће) да не реагује, под условом да је постављен бит за маскирање прекида (I=1). На захтев за немаскирајући прекид ($\overline{\text{XIRQ}}$) процесор увек реагује независно од статуса I бита. Другим речима, његов утицај програмер не може софтверски спречити.

Приоритет је тачно одређен, при чему серијски комуникациони интерфејс (SCI) има најнижи приоритет, а RESET највећи. За неке изворе прекида, као што је захтев за прекид серијског интерфејса, флегови се аутоматски бришу за време одговора на захтев за прекид. На пример флег RDRF у серијском примопредајнику (8. поглавље) брише се механизмом за аутоматско брисање,

¹ Пин

² *Timer overflow, real-time interrupt, pulse accumulator*

³ *Serial Communication Interface*

⁴ *Serial Peripheral Interface*

⁵ *Hardware Priority Interrupt Request*

7. Улазно/излазни начин рада

који се састоји од читавања статуса серијског интерфејса и читавања регистра података серијског интерфејса. Ова два корака реализују механизам аутоматског брисања, без потребе за додатним инструкцијама. Многи од могућих захтева користе исти вектор прекида. На пример постоји већи број могућих извора прекида који користе вектор прекида на адреси \$FFF2. Тако, када се обрађује прекид софтвер мора да одреди који од могућих сигнала је довео до прекида. Комплетна листа могућих прекида дата је у табели 7.1

<i>Адреса вектора прекида</i>	<i>Извор прекида, флаг</i>	<i>Маскирање</i>
\$FFFE	RESET	Није могуће
\$FFFC	COP ¹ Clock Monitor Fail Reset	Није могуће
\$FFFA	COP Failure Reset	Није могуће
\$FFF8	Unimplemented Instruction Trap	Није могуће
\$FFF6	SWI	Није могуће
\$FFF4	XIRQ	X бит
\$FFF2	IRQ или Paralel I/O handshake, STAF ²	I бит
\$FFF0	Real Time Interrupt, RTIF	I бит
\$FFEE	Timer Interrupt Capture 1, IC1F	I бит
\$FFEC	Timer Interrupt Capture 2, IC1F	I бит
\$FFEA	Timer Interrupt Capture 3, IC3F	I бит
\$FFE8	Timer Interrupt Compare 1, OC1F	I бит
\$FFE6	Timer Interrupt Compare 2, OC2F	I бит
\$FFE4	Timer Interrupt Compare 3, OC3F	I бит
\$FFE2	Timer Interrupt Compare 3, OC4F	I бит
\$FFE0	Timer Interrupt Compare 5, OC5F	I бит
\$FFDE	Timer Overflow, TOF	I бит
\$FFDC	Pulse Accumulator Overflow, PAOVF	I бит
\$FFDA	Pulse accumulator Input Edge, PAIF	I бит
\$FFD8	SPI Serial transfer Complete, SPIF	I бит
\$FFD6	SCI Transmit Buffer Empty, TDRE	I бит
	SCI Transmit Complete, TC	
	SCI Receiver Buffer Full, RDRF	
	SCI receiver Idle, IDLE	

Табела 7.1 Вектори прекида за MC68HC11

¹ Computer operating properly

² Strobe A flag

Улаз за немаскирајући прекид $\overline{XIRQ}$ реагује на ниво сигнала. Активан је када је улаз $\overline{XIRQ}$ на напонском нивоу логичке нуле. Маскирајући $\overline{IRQ}$ улаз може се програмирати да реагује на ниво сигнала¹ или на ивицу сигнала². То се може постићи контролом $IRQE$ бита (бит 5) у $OPTION$ регистру³. Када је $IRQE=0$, процесор реагује на ниво сигнала који се доводи на њен $\overline{IRQ}$ улаз. Када је $IRQE=1$ онда процесор реагује на промену нивоа сигнала (ивицу) који се доводи на улаз.

7.6.5 Повезивање А/Д конвертора и процесора са прекидом

Када се користи маскирајући прекид ($\overline{IRQ}$), захтева се размена одговарајућих сигнала између процесора и уређаја који генерише $\overline{IRQ}$ сигнал. Анализираћемо везу А/Д конвертора и процесора. Претпоставићемо да је бит $IRQE$ у регистру $OPTION$ обрисан ($IRQE=0$) што значи да ће процесор да реагује на ниво $\overline{IRQ}$ сигнала, а не на промену нивоа $\overline{IRQ}$ сигнала.

Да би се реализовало повезивање А/Д конвертора и процесора са прекидом, (слика 7.11), потребне су измене у односу на начин повезивања представљен на слици 7.7. Прво, $BUSY$ излаз није везан на магистралу података пошто процесор неће испитивати његову вредност да би се одредило када А/Д конвертор има спремне податке. Уместо тога, активна ивица $BUSY$ сигнала ће окидати JK флип-флоп, а његов излаз доводи $\overline{IRQ}$ улаз процесора на активан ниво по завршетку А/Д конверзије.

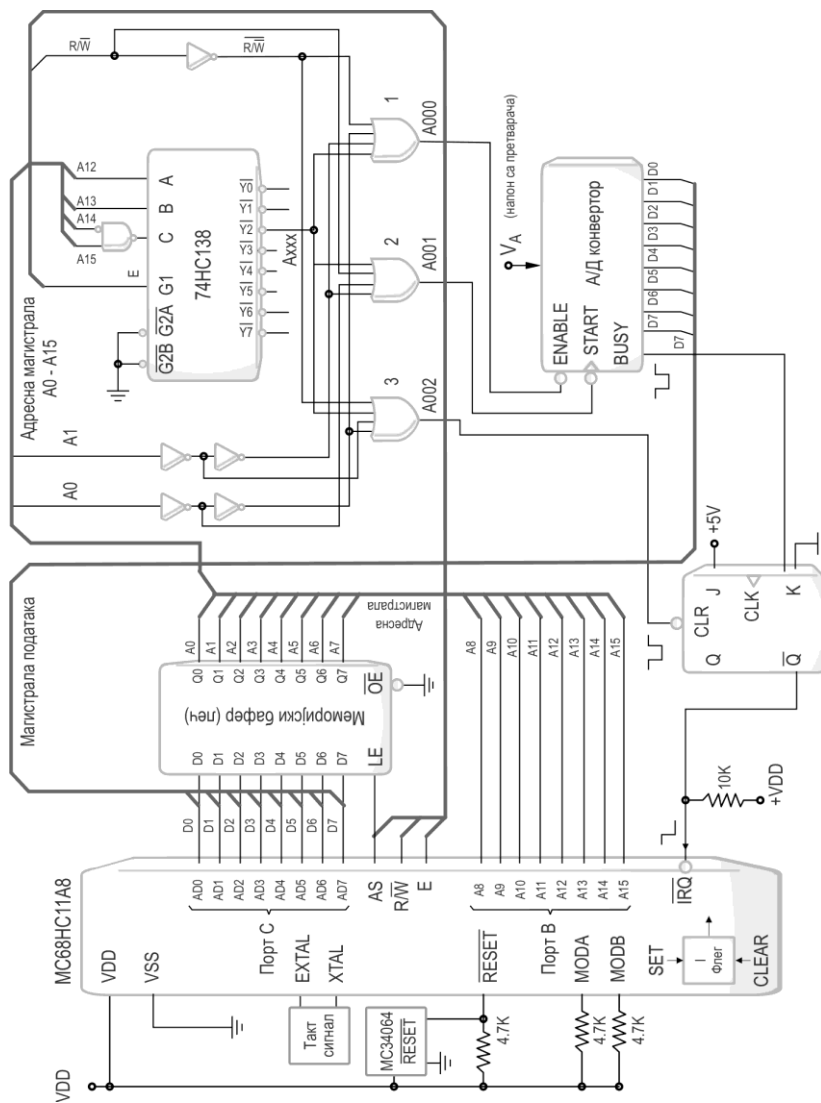
Уколико је $I=0$ када је улаз $\overline{IRQ}$ активан процесор ће на њега реаговати. Ако је $I=1$ процесор неће реаговати на $\overline{IRQ}$ улаз већ ће наставити са извршавањем инструкција текућег програма док се CLI инструкцијом I бит не обрише. Тада ће процесор реаговати на $\overline{IRQ}$ улаз (уз претпоставку да је он и даље активан) и започети процедуру за обраду прекида. Аутоматски се поставља I бит ($I=1$) да процесор не би реаговао на нови прекид за време извршавања потпрограма за прекид.

У овом примеру потпрограма за прекид садржи инструкције које учитавају излаз А/Д конвертора у акумулатор A . Такође може да садржи и инструкције за обраду података добијених из А/Д конвертора. Последња

¹ Level sensitive

² Edge sensitive

³ Сви детаљи се могу наћи у M68HC11 Reference manual



Слика 7.11 Везивање А/Д конвертора и процесора коришћењем маскирајућег прекида

инструкција у потпрограму за прекид биће инструкција RTI^1 , која враћа процесор на извршавање програма пре појаве прекида.

По извршењу RTI инструкције, из стек меморије биће враћени подаци у интерне регистре процесора. На тај начин реконструисан је и садржај регистра услова у коме је бит I био обрисан ($I=0$). Процесор може поново да реагује на исти сигнал $\overline{IRQ}=0$ (излаз JK флип-флопа). Да се то не би десило потребно је поставити $\overline{IRQ}$ улаз на напонски ниво логичке јединице пре него што се изврши RTI инструкција у потпрограму за обраду прекида. У противном, процесор би при повратку у главни програм поново реаговао на исти прекид. Један од начина да се то спречи било би постављање инструкције, негде у потпрограму за обраду прекида, која шаље импулс који брише JK флип-флоп и поставља IRQ на неактиван ниво. У примеру који је приказан на слици 7.11 користи се троулазно I коло. Његов излаз биће на активном напонском нивоу извршавањем „празне“ инструкције за читавање са адресе $\$A002$. Потпрограм који обрађује прекид који генерише A/D конвертор са слике 7.11 је:

```
ISR      LDAA  $A002 ; празно читавање у акумулатор А
          LDAA  $A000 ; читавање податка из А/Д у акумулатор А
          STAA  $ 60  ; смести А/Д податак у меморију
RETURN   RTI          ; повратак у главни програм
```

Анализирајмо асемблерски програм:

- прва инструкција генерише импулс којим се поставља $\overline{IRQ}=1$;
- друга инструкција читава податак са излаза A/D конвертора у акумулатор A ;
- трећа инструкција смешта прочитани податак у меморију и на крају
- RTI инструкција враћа процесор на извршавање главног програма, где је и била када га је A/D конвертор прекинуо.

Пошто је улаз $\overline{IRQ}$ на неактивном напонском нивоу, не може да прекине процесор до новог циклуса A/D конверзије, коју процесор мора да иницира.

¹ *ReTurn from Interrupt* - повратак из потпрограма за прекид

7.7 ПОЈАВА ПРЕКИДА ЗА ВРЕМЕ ИЗВРШАВАЊА ПОТПРОГРАМА ЗА ОБРАДУ ПРЕКИДА

Да ли се потпрограм који обрађује прекид може прекинути? Уколико се појави захтев за прекид за време извршавања потпрограма за обраду првог прекида, а новоприспели прекид може се маскирати, онда се он не може прекинути уколико је I бит постављен (што је обично случај). Уколико је I бит обрисан ($I=0$), потпрограм за обраду прекида може се прекинути. Када је $\overline{XIRQ}$ улаз активан, независно од стања I бита, потпрограм за обраду прекида биће прекинут.

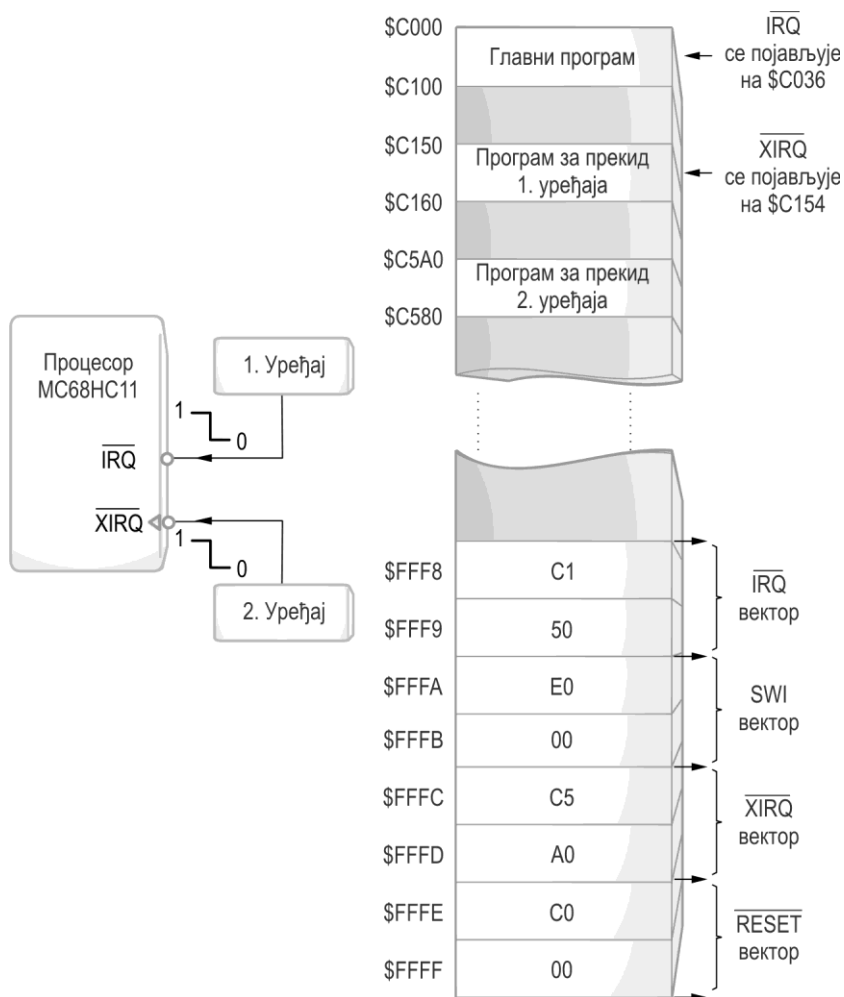
Да бисмо илустровали скуп догађаја који се јављају када треба прекинути извршавање потпрограма за прекид, користимо пример илустрован сликом 7.12. За $\overline{IRQ}$ и $\overline{XIRQ}$ улазе процесора везана су два независна улазно/излазна уређаја¹.

У овом примеру користимо меморијску мапу представљену на слици 7.12. Главни програм заузима адресни простор од $\$C000$ до $\$C100$. Потпрограм за обраду прекида за 1. уређај смештен је на адресама $\$C150$ - $\$C160$, а потпрограм за обраду прекида за 2. уређај налази се на адресама $\$C5A0$ - $\$C5B0$. Меморијске адресе $\$FFF4$ - $\$FFFF$ садрже адресе вектора прекида и ресет вектора.

Претпоставићемо да 1. уређај генерише захтев за прекид (активни ниво на улазу $\overline{IRQ}$), док процесор извршава инструкцију на адреси $\$C036$ у главном програму. Када процесор заврши извршавање ове инструкције и детектује захтев за прекид (уз претпоставку да је $I=0$), узима $\overline{IRQ}$ адресни вектор са локације $\$FFF8$ и $\$FFF9$. Да би објашњење било јасније претпостављено је да се прекид јавља у току извршавања једнобајтне инструкције², односно да је адреса следеће инструкције $\$C037$. Адресни вектор ($\$C150$) учитава се у програмски бројач пошто је претходни садржај програмског бројача ($\$C037$) смештен у стек меморију заједно са индекс регистрима Y и X , акумулаторима A и B и регистром услова. Процесор затим започиње са извршавањем потпрограма за обраду прекида за 1. уређај, од адресе $\$C150$.

¹ $\overline{IRQ}$ и $\overline{XIRQ}$ улази реагују на ниво улазног сигнала а не на промену нивоа сигнала

² У случају да је двобајтна инструкција адреса, садржај програмског бројача био би $\$C038$ а тробајтна $\$C039$.



Слика 7.12 Појава прекида за време извршавања потпрограма за обраду прекида

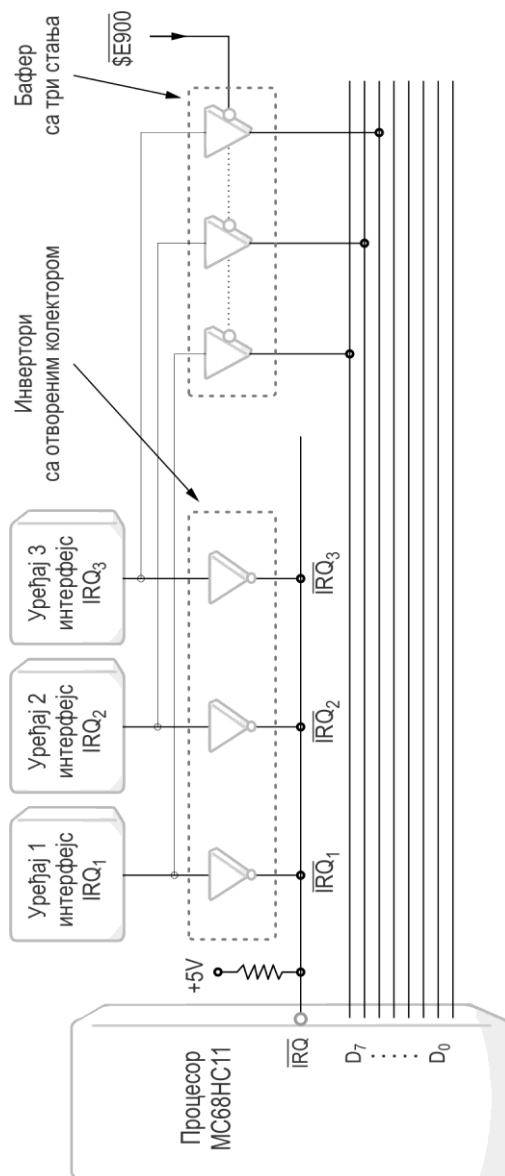
Сада ћемо претпоставити да 2. уређај генерише захтев за прекид (активан ниво на улазу $\overline{XIRQ}$) док процесор извршава једнобајтну инструкцију на адреси $\$C154$. Независно од вредности 1 бита, процесор ће реаговати на овај прекид чим заврши извршавање текуће инструкције. Узеће $\overline{XIRQ}$

адресни вектор са локације \$FFF4 и \$FFF5. Овај адресни вектор, \$C5A0 се учитава у програмски бројач пошто су његов претходни садржај (\$C155) и садржаји регистара процесора сачувани у стек меморији. Процесор сада започиње са извршавањем потпрограма за обраду прекида за 2. уређај. Овај потпрограм почиње на адреси \$C5A0.

Последња инструкција у потпрограму RTI налаже процесору да настави са извршавањем програма на адреси на којој се налазио када се $\overline{\text{XIRQ}}$ прекид појавио. Да би то остварио процесор читава девет бајтова с врха стек меморије и смешта их у одговарајуће регистре. Последња два бајта смештају се у програмски бројач. Овим ће садржај програмског бројача бити \$C155 тако да се процесор враћа на извршавање потпрограма за 1. уређај. Последња инструкција у овом потпрограму (RTI) налаже процесору да настави са извршавањем програма са адресе на којој се налазио када се прекид $\overline{\text{IRQ}}$ појавио. Процесор читава девет бајтова са стек меморије и смешта их у одговарајуће регистре. Последња два бајта смештају се у програмски бројач тако да је његов садржај \$C037. Са овим садржајем програмског бројача процесор се враћа на извршавање главног програма. Овај пример приказује не само процес прекида, већ и велики значај стек меморије у обезбеђивању нормалног повратка у главни програм после извршавања потпрограма за прекид.

7.8 РАД СА ВИШЕ УЛАЗНО/ИЗЛАЗНИХ УРЕЂАЈА

Уколико процесор има један или два улаза за прекид, како може да обради захтеве већег броја улазно/излазних уређаја? На слици 7.13, је приказан један од начина како неколико улазно/излазних уређаја могу да деле (истовремено користе) заједничку улазну линију за прекид процесора. Кола за прикључивање (интерфејсна кола) сваког од уређаја генеришу излазни, прекидни сигнал, који се преко инвертора са отвореним колектором везује за $\overline{\text{IRQ}}$ улаз процесора. Излази интерфејсних кола су преко бафера са три стања везани за магистралу података. Активан ниво на излазу било ког интерфејсног кола доведиће $\overline{\text{IRQ}}$ улаз на активан ниво и генерисати захтев за прекид. На овај начин, више улазно/излазних уређаја може генерисати захтев за прекид. Када препозна захтев за прекид процесор мора да одреди који од уређаја је генерисао захтев за



Слика 7.13. Више улазно/излазних уређаја везаних на заједничку линију за прекид

прекид да би покренуо извршавање одговарајућег потпрограма за обраду прекида.

Најједноставнији начин за рад са више извора прекида је метод са испитивањем (упитом). Овај метод је спор али захтева мало спољашњег хардвера. Када се појави захтев за прекид, процесор започиње извршавање општег потпрограма за обраду прекида којим се одређује извор прекида. Процесор испитује статус сваког уређаја, док не наиђе на уређај који је генерисао захтев за прекид. У зависности од тога који је уређај у питању процесор наставља са извршавањем потпрограма за обраду прекида за уређај.

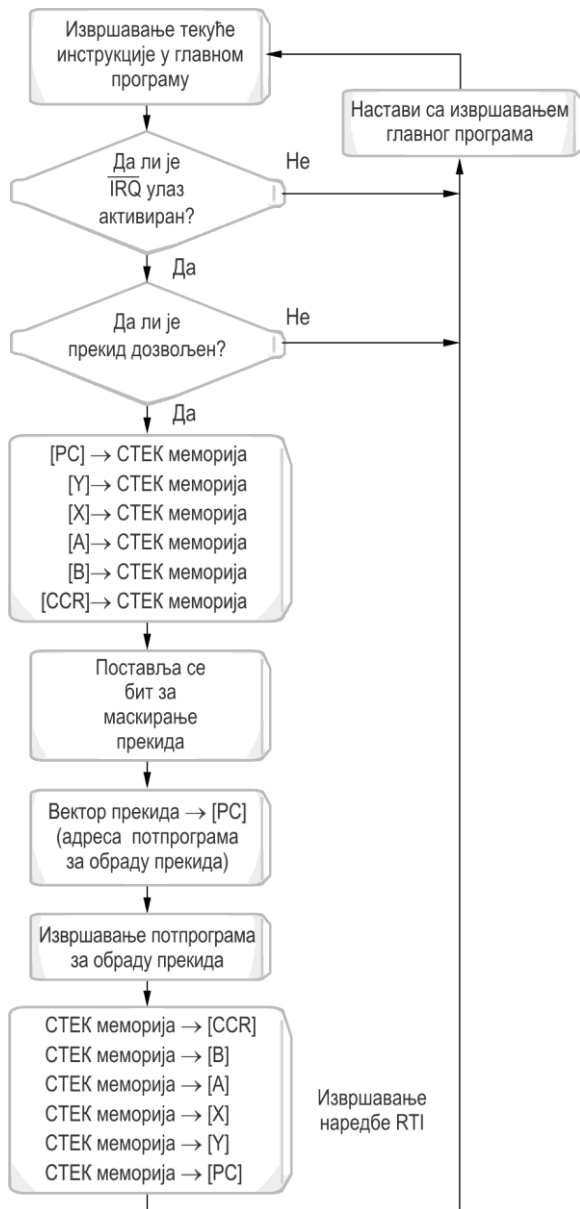
Код методе са испитивањем прекида, потребно је да постоји одређена припрема за истовремени рад са више прекида. Обично се успоставља систем са приоритетима, где се сваком од уређаја додељује различит приоритет. Када два уређаја генеришу истовремено захтев за прекид процесор ће прво обрадити захтев за прекид уређаја са највишим приоритетом, а затим захтев за прекид уређаја са нижим приоритетом. Приоритет је исти као и редослед којим се врши провера захтева. На слици 7.13 види се да када се појави прекид процесор прелази на извршавање општег потпрограма за обраду прекида, који испитује статус сваког од уређаја. Статус интерфејсних излаза, процесор може да добије извршавањем инструкције учитавања (LDAA \$E900). Одговарајућим програмом може се утврдити који од уређаја је генерисао захтев за прекид.

Обрада прекида

У скупу инструкција процесора MC68HC11 налази се и инструкција која се означава са WAI¹. Када се она извршава процесор се припрема за обраду² захтева за прекид смештањем садржаја програмског бројача (PC), индекса регистара (Y, X), акумулатора (A, B) и садржаја регистра услова (CCR) на стек меморију. Процесор ће остати у овом стању док се не појаве захтеви за прекид (IRQ или XIRQ). Пошто су садржаји регистара сачувани на стек меморији, процесор ће директно да смести у програмски бројач

¹ *Wait for Interrupt* - чекање на прекид

² Користе се термини обрада или опслуживање прекида.



Слика 7.14 Кораци које процесор извршава када обрађује маскирајући прекид

одговарајући адресни вектор (адреса потпрограма за обраду прекида). Овом процедуром штеди се осам циклуса такт сигнала, и на тај начин скраћује време одзива процесора на прекид.

Кораци које процесор обавља када обрађује прекид $\overline{IRQ}$ су представљени на слици 7.14. Сваки од корака процесора извршава се аутоматски, и не захтева посебне инструкције у програму. Наравно, програмер треба да напише потпрограм за обраду прекида и обезбеди да вектор прекида указује на тачну адресу потпрограма за прекид.

Процесор по завршетку сваке инструкције у главном програму испитује да ли је улаз за прекид ($\overline{IRQ}$) активан. Уочимо да се стек меморија користи за чување повратне адресе (садржај програмског бројача), садржаја индекса регистара (X, Y), садржаја акумулатора (A, B) и садржаја регистра услова. Исте кораке процесор ће обавити и за немаскирајући прекид $\overline{XIRQ}$, осим што се не испитује да ли је бит I постављен, пошто се $\overline{XIRQ}$ не може забранити.

7.9 ПРЕНОС ИЗМЕЋУ У/И УРЕЂАЈА И МЕМОРИЈЕ

До сада само смо посматрали пренос података између процесора и улазно/излазних уређаја. Постоје многе ситуације када пренос података треба да се обави између меморије и спољашњег уређаја. Улазни уређај може да има податке који треба да буду уписани у RAM меморију. На пример компакт диск (флеш меморија...) са кога се учитавају подаци или програм. У одговарајућем тренутку програм или блок података које је процесор обрадио могу се пренети из RAM меморије или из интерних регистара процесора ка излазном уређају. Такође, излазни уређај може да захтева податке који су смештени у RAM меморију. На пример комплетан блок алфанумеричких података требало би пребацити из RAM меморије на видео монитор. Када су улазно/излазни уређаји укључени у пренос блокова података ка или из RAM меморије, постоје два начина на која се то може урадити: програмирани пренос и директан приступ меморији.

7.9.1 Програмирани пренос

Уобичајен је програмирани пренос: процесор изводи секвенцу инструкција за пренос података између RAM меморије и улазно/излазних уређаја. На пример процесор извршава инструкције за учитавање података

из RAM меморије и смешта их у акумулатор (нпр. LDAA инструкција). Затим извршава операцију уписивања садржаја акумулатора у спољашњи уређај (STAA инструкција). Овим методом подаци не пролазе директно између меморије и периферијског уређаја, пошто прво морају да прођу кроз процесор (слика 7.15а). Исто је и код програмираног преноса из улазног уређаја ка RAM меморији. Подаци се преносе од улазног уређаја ка процесору, а затим ка RAM меморији.

7.9.2 Директан приступ меморији

Процесор троши доста времена за учитавање инструкције и података; потребно је да се сви подаци пренесу кроз његове интерне регистре. Уколико би подаци могли да се пренесу директно између периферијског уређаја и меморије пренос би био бржи. Овај процес назива се директан приступ меморији (DMA¹) и алтернатива је програмираном преносу. Директан приступ меморији значајно повећава брзину преноса података² у односу на програмирани пренос, нарочито када је потребно пренети велики број блокова података. Слика 7.15б показује да се код директног приступа меморији пренос података обавља између меморије и периферијског уређаја без посредства процесора. То значи да интерфејсна кола за директан приступ меморији треба да приступају меморији при операцији читавања и уписивања на исти начин како то ради процесор. Интерфејсно коло за директан приступ меморији треба да буде у стању да уради следеће:

1. Одговори на захтев периферијског уређаја за директан приступ меморији генерисањем управљачких сигнала ка процесору да би се он поставио у HALT стање. У HALT стању процесор одлаже операције које треба да изврши, препушта управљање над системским магистралама постављањем адресне магистрале, магистрале података и линије за управљање меморије $R/\overline{W}$ у стање високе импедансе.
2. Једном када процесор потврди да је у HALT стању. Интерфејсно коло за директан приступ меморији преузима управљање над

¹ Direct Memory Access

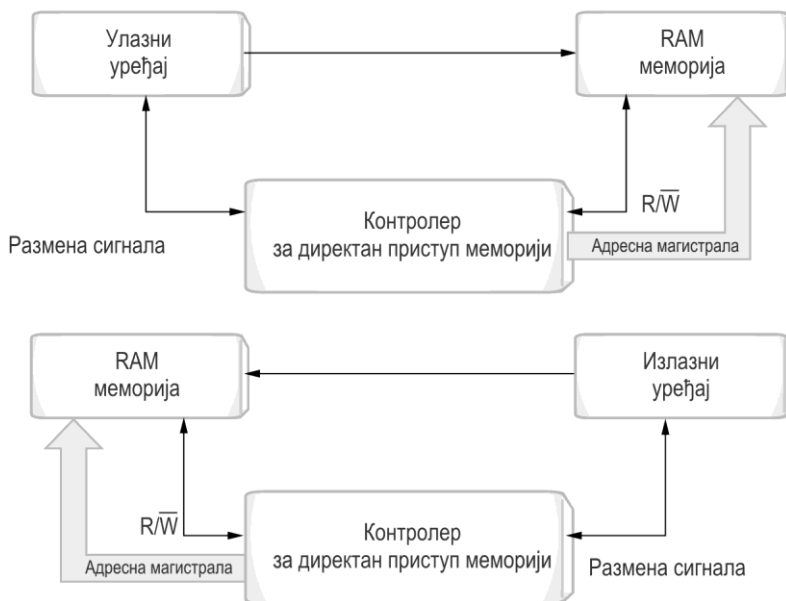
² Број пренетих речи података у секунди.

7. Улазно/излазни начин рада

а) Програмирани пренос података између RAM меморије и улазно/излазног уређаја



б) Директан приступ меморији без посредства процесора



Слика 7.15 а) програмирани пренос б) директан приступ меморији

адресном и управљачком магистралом. Оно поставља адресу RAM меморијске локације, којој периферијски уређај треба да приступи, на адресну магистралу и генерише управљачке сигнале неопходне за пренос података између RAM меморије и периферијских уређаја.

3. Интерфејсно коло за директан приступ меморији води рачуна о локацијама у RAM меморији којима се приступа, и када се пренос података заврши враћа процесор из HALT стања. Процесор поново преузима управљање над магистралама и наставља са извршавање програма.

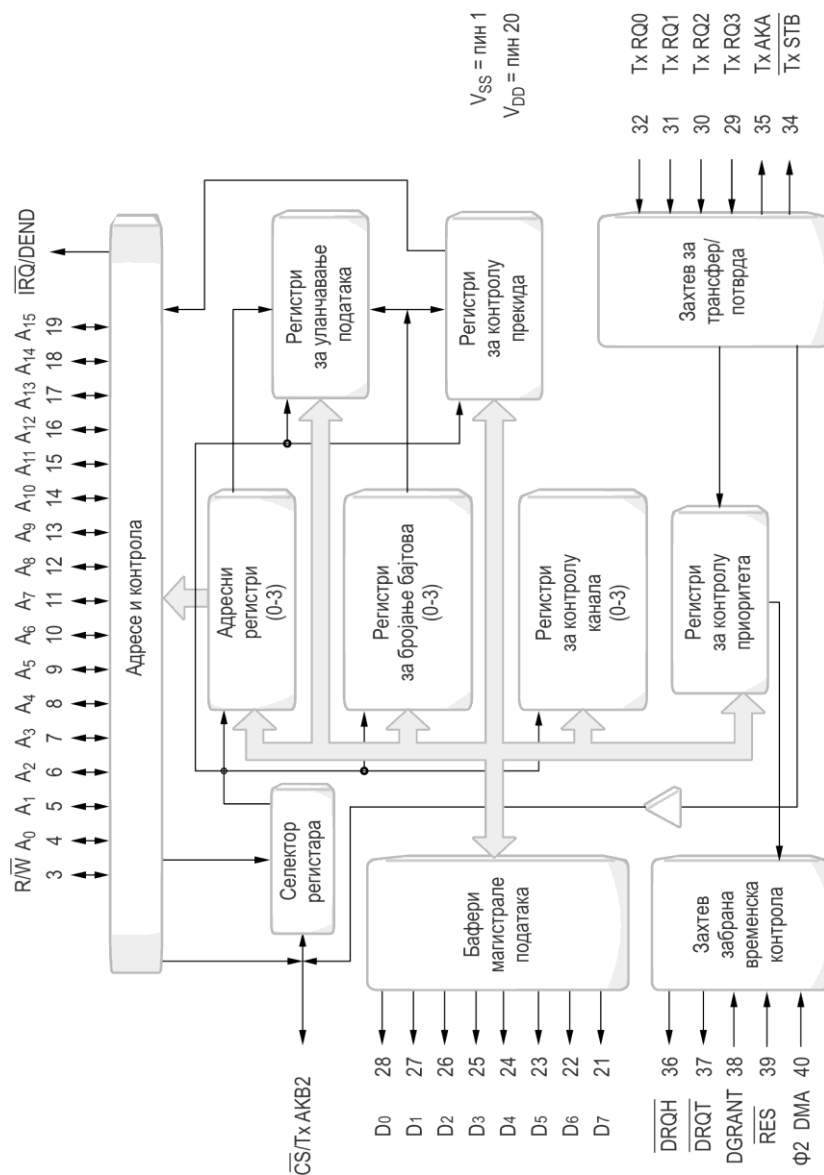
Види се да су интерфејсна кола за директан приступ меморији сложена пошто дуплирају многе функције које и процесор извршава. Motorola је развила контролер за директан приступ меморији. Микроконтролери из серије MC68HC11 не садрже интефејсна кола за директан приступ меморији. Директан приступ меморији могу реализовати само са неким од контролера. У овом уџбенику анализираћемо MC6844 контролер за директан приступ меморији. Контролер се може користити за управљање директним преносом података између меморије и највише 4 периферијска уређаја, и садржи све што је потребно да би се извршавале све описане операције.

Контролер за директан приступ меморији MC6844

Контролер за директан приступ меморији MC6844 је сложено интегрисано коло које је способно за извршавање свих функција неопходних за пренос података директно између меморије и четири периферијска уређаја. Пошто је елиминисано време потребно процесору да приступи и обради инструкцију, значајно се повећава брзина којом подаци могу да се пренесу између меморије и периферијских уређаја. Да би то постигао, MC6844 контролер мора да буде способан да преузме управљање над адресном и магистралом података, генерише одговарајуће управљачке сигнале и изврши размену потребних сигнала са процесором.

Интеракција процесора и контролера одређена је подацим уписаним у интерне регистре контролера. Процесор може да програмира шест интерних регистара контролера MC6844 (слика 7.16).

Садржај ових регистара дефинише карактеристике преноса и дозвољава контролеру MC6844 да управља директним приступом меморији. Контролер MC6844 садржи шест интерних регистара и то су:



Слика 7.16 Блок дијаграм контролера за директан приступ меморији MC6844

1. **Адресни регистар**¹. Ово је 16-битни регистар (један од четири) који се пуни почетном адресом у меморији од које ће започети пренос у директном приступу меморији.
2. **Регистар за бројање бајтова**². Овај 16-битни регистар (један од четири) напуњен вредношћу једнаком броју бајтова које треба пренети. После преноса сваког бајта садржај регистра се декрементира, а у регистар адресе учитава се нова вредност. Када бројач бајтова достигне нулу пренос се завршава.
3. **Регистар за контролу канала**³. Ово је осмобитни регистар (један од четири) који је напуњен вредношћу која одређује начин на који ће подаци бити пренети. Различити битови у овом регистру ће управљати стварима као:
 - а) Да ли ће подаци бити уписани или очитани из меморије?
 - б) Да ли ће се пренос обавити у *Halt Burst*, *Halt Steal* или *TSC*⁴ *Steal* начину рада?
 - в) Да ли ће садржај адресног регистра после пренетог бајта бити инкрементиран или декрементиран?
 - г) Обезбеђивање статусних битова који указују на завршетак директног преноса и генерисања *IRQ* прекида уколико је то дозвољено.
4. **Регистар контроле приоритета**⁵ може да управља преносом са директним приступом меморији за највише четири периферијска уређаја. Овај осмобитни регистар одређује који ће канал бити селектован када се појави захтев за пренос. Због могућности да сва четири модула могу да захтевају пренос у исто време, садржај регистра ће одредити којим приоритетом ће MC6844 контролер одговорити на захтеве. Уобичајени приступ је да канал 0 има највећи приоритет. Остали канали су са опадајућег приоритета. Други начин је ротациона шема приоритета. Уређај који је управо

¹ *Adress Register*

² *Byte Count Register*

³ *Channel Control Register*

⁴ *Tree State Control*

⁵ *Priority Control Register*

услужен добија најнижи приоритет, а онај који најдуже није био услужен, добија највећи приоритет.

5. **Регистар за контролу прекида**¹. Ово је осмобитни регистар који се користи да одреди који канал генерише $\overline{IRQ}$ захтев за прекид. Регистар такође обезбеђује статусни бит који процесор може да прочита да би видео да ли је овај или неки други уређај генерисао захтев за прекид.
6. **Регистар за уланчавање података**². То је осмобитни регистар који дозвољава контролеру MC6844 да изводи поновљено читавање блока меморије. Ово се постиже тако што контролер преноси садржај адресног регистра и регистра који броји бајтове података у одабрани канал за уланчавање, после завршетка преноса са директним приступом меморији. Овим се омогућава брзо понављање претходног преноса података без потребе да процесор поново уписује потребне информације у регистар за бројање бајтова и адресни регистар.

Уобичајени начин рада

Да бисмо помогли бољем разумевању ових уређаја анализираћемо, на функционалном нивоу, рад контролера MC6844.

Уређај који жели да користи директан приступ меморији обратиће се контролеру за директан приступ меморији. Контролер ће послати захтев процесору да му препусти контролу над магистралама, тј. да он пређе у неактивно стање³. Процесор завршава извршавање текуће инструкције и одговара постављајући магистралу података, управљачку и адресну магистралу у стање високе импедансе. Другим речима зауставља све даље активности и шаље одговарајући сигнал контролеру којим указује да му препушта управљање над свим магистралама⁴. Контролер онда тражи од

¹ *Interrupt Control Register*

² *Data Chain Register*

³ MC68HC11A8 на који смо се највише ослонили у овом уџбенику не садржи HALT улаз који иницира прелазак процесора у неактивно стање, односно стање у коме се његове: магистрала података, управљачка и адресна магистрала налазе у стању високе импедансе. На пример микропроцесор M6800 чији је начин рада објашњен у ранијем издању овог уџбеника имао је HALT улаз.

⁴ На пример сигнал: BA (*Bus Available* - магистале су доступне).

уређаја који је тражио његову „услугу“ да постави податке на магистралу података. Контролер поставља на адресну магистралу садржај свог адресног регистра. Подсетимо се да је процесор уписао у регистар контролера адресу меморије од које се почиње са уписивањем података. После уписивања првог податка контролер декрементира свој регистар за бројање бајтова и инкрементира адресни регистар. Нова адреса се поставља на адресну магистралу и нови бајт података се из улазног уређаја поставља на магистралу података. Сигнал за уписивање у меморију ($R/\overline{W}=0$) обезбеђује контролер. Овај процес се наставља док регистар за бројање бајтова не достигне вредност нула што значи да су сви подаци из улазног уређаја пренети у меморију. Контролер препушта управљање над магистралама процесора. Контролер такође активира $\overline{IRQ}$ улаз за прекид да укаже процесору да је пренос података од улазног уређаја до меморије завршен.

Начини рада у директном приступу меморији

Претходно описани начин рада при преносу података од уређаја до меморије означава се као „заустави и пренеси одједном“¹. Процесор се зауставља, контролер преузима управљање над магистралама. Тек када се цео блок података пренесе враћа се управљање над магистралама процесору.

Други начин рада контролера за директан приступ меморији је „заустави и пренеси помало“². Овај начин преноса сличан је преносу „заустави и пренеси одједном“. Разлика је у томе што се не преноси читав блок података већ само један бајт. По завршетку извршавања следеће инструкције процесора контролер ће пренети још један бајт. Процес се наставља док се цео блок података не пренесе.

Трећи начин рада је „замрзни такт сигнал, пренеси помало“³, врста рада⁴. У овој врсти рада уместо да контролер чека да процесор заврши извршавање текуће инструкције он „замрзава“ његов такт сигнал. Тада

¹ *Halt Burst*

² *Halt Steal*

³ *TSC Steal*

⁴ Односи се само на оне процесоре са динамичким меморијама. Процесор MC68HC11 је направљен у технологији статичких меморија. Микропроцесор M6800 је направљен у технологији динамичких меморија.

је процесор принуђен да постави све своје магистарале у стање високе импедансе. Контролер преузима управљање над магистралама и преноси један бајт података. Предност овог начина преноса је у предвидивом времену одзива пошто процесор не мора да заврши извршавање текуће инструкције. Недостатак је додатни спољашњи хардвер који је потребан за ову врсту рада и ограничено време за пренос, пошто интерни регистри процесора могу да изгубе податке после одређеног времена ако не долази од освежавања њиховог садржаја за које је потребан такт сигнал.

7.10 ПРОГРАМАБИЛНИ ЧАСОВНИК

Програмско генерисање променљивих временских интервала може се остварити са програмабилним часовником. У овом одељку биће дат кратак преглед програмабилног часовника MC6840.

Програмабилни часовник има три бинарна шеснаестобитна бројача, управљачке регистре за сваки од бројача и статусни регистар (слика 7.17). Радом бројача може се програмски управљати, на шта указује и његов назив. У систему се могу користити излазни такт сигнали сваког од бројача или прекиди које може генерисати сваки од бројача. Програмабилни часовник може се користити за:

- генерисање периодичних сигнала правоугаоног таласног облика,
- мерење учестаности спољашњег сигнала,
- бројање спољашњих догађаја,
- мерење трајања временских интервала,
- мерење трајања временских интервала између два догађаја,
- генерисање импулса жељене ширине,
- генерисање временски померених сигнала у односу на улазни сигнал,
- ширинску модулацију (PWM¹).

Техничка спецификација:

- напон напајања 5V (DC²),
- потпуно TTL компатибилан,
- ради са једним системским такт сигналом,

¹ Pulse-Width Modulation

² Direct Current

- системски такт сигнал може имати максималну учестаност 8MHz,
- програмабилни излаз за прекид ($\overline{IRQ}$) према процесору,
- три улаза за асинхроне спољашње такт сигнале,
- $\overline{RESET}$ улаз,
- три маскирајућа излаза за генерисање таласних облика

Блок шема програмибилног часовника MC6840 приказана је на слици 7.17. Три часовника унутар интегрисаног кола могу се независно програмирати чиме се обезбеђује истовремени рад у више режима рада (истовремено генерисање сигнала, мерење временских интервала итд.).

Магистрала података

Часовник MC6840 повезује се са процесором преко осмобитне двосмерне магистрале података. Преко магистрале се размењују подаци између часовника (интерних регистара програмабилног часовника) и процесора. Магистрала са стране часовника је у стању високе импедансе, осим у случају када је часовник селектован. Линије су улазне или излазне у зависности од тога да ли се у часовник уписују подаци или се из њега читава.

Линије за селектовање ($\overline{CS0}$, CS1)

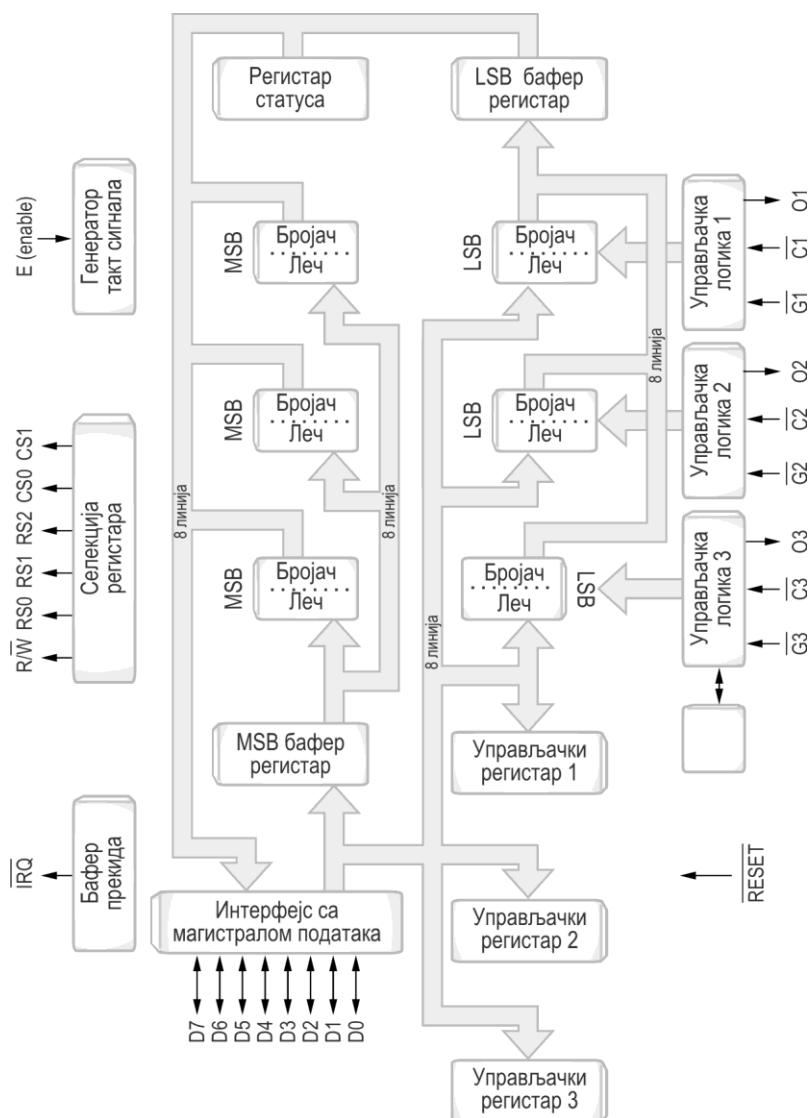
Часовник има два сигнала ($\overline{CS0}$ и CS1) који се користе за управљање магистралом података. Да би магистрала података програмабилног часовника била активна сигнал $\overline{CS0}$ мора бити на нивоу логичке нуле, а CS1 на нивоу логичке јединице. Ови сигнали генеришу се у декодерској логици.

Сигнал за уписивање/очитавање ($R/\overline{W}$)

Сигнал за уписивање/очитавање ($R/\overline{W}$) генерише процесор и њиме се дефинише смер тока података. Ако је ова линија на логичкој нули подаци се уписују у часовник, а ако је на логичкој јединици подаци се читају из часовника, али само у случају када је он селектован (посредством линија за селектовање).

Системски такт сигнал (Е - такт сигнал)

Е -такт сигнал (системски такт) синхронизује пренос података између процесора и часовника, остварује синхронизацију са спољашњим такт сигналом, спољашњим импулсом и $\overline{RESET}$ сигналом.



Слика 7.17. Блок шема програмабилног часовника MC6840

Захтев за прекид ($\overline{IRQ}$)

Излазна линија је реализована у технологији отвореног колектора тако да се више излазаних линија може везати паралелно (ИЛИ логика). Ако је у микрорачунарском систему потребно радити са више модула часовника тада се њихове линије за прекид везују паралелно. У случају да је активиран неки од прекида на линији је ниво логичке нуле. Излази $\overline{IRQ}$ часовника повезују се на $\overline{IRQ}$ улаз процесора. Информација о томе шта је узрок прекида добија се читавањем статус регистра.

Поновно укључивање (RESET)

Логичка нула на овом улазу довешће до тога да:

- се сви лечеви бројача постављају на вредност \$FFFF,
- сви битови управљачких регистара постављају на логичку нулу, осим интерног бита CR10 који се поставља на логичко 1,
- сви бројачи добијају вредност постављену њиховим лечевима,
- сви излази бројача се ресетују и такт сигнал бројача се онемогућава,
- сви битови (флегови) у регистру статуса се бришу.

Улази за селекцију интерних регистара (RS0, RS1, RS2)

Посредством улаза RS0, RS1, RS2 остварује се приступ жељеном интерном регистру програмабилног часовника. Уобичајено је да се на ове линије доводе најниже адресне линије са процесора. На улаз RS0 доводи се адресна линија A0, на улаз RS1 доводи се адресна линија A1 а на RS2 доводи се адресна линија A2. У табели 7.1 дата је адресна мапа интерних регистара.

Улази за селекцију регистра			Операција	
RS2	RS1	RS0	$R/\overline{W}=0$	$R/\overline{W}=1$
0	0	0	CR2=0 Упис у управљачки регистар бр.3 CR2=1 Упис у управљачки регистар бр.1	Нема операције
0	0	1	Упис у управљачки регистар бр.2	Очитавање статус регистра

7. Улазно/излазни начин рада

Улази за селекцију регистра			Операција	
RS2	RS1	RS0	$R/\bar{W}=0$	$R/\bar{W}=1$
0	1	0	Упис у MSB бафер регистар	Очитавање бројача часовника бр. 1
0	1	1	Упис у леч часовника бр.1	Очитавање LSB бафер регистра
1	0	0	Упис у MSB Бафер регистар	Очитавање бројача часовника бр. 2
1	0	1	Упис у леч часовника бр.2	Очитавање LSB бафер регистра
1	1	0	Упис у MSB бафер регистар	Очитавање бројача часовника бр. 3
1	1	1	Упис у леч часовника бр.3	Очитавање LSB бафер регистра

Табела 7.1 Улази за селекцију и операције у интерним регистрима

Излази часовника (O1, O2, O3)

Излази могу побудити два TTL оптерећења. На сваком од ових излаза генерише се таласни облик који одговара програмираном режиму рада припадајућег часовника.

Улази за мерење временских прозора (G1, G2, G3)

Улазне линије које примају асинхроне TTL компатибилне сигнале који се користе као окидачки сигнали или временски прозори за часовнике 1, 2 и 3 респективно. Ови улази директно утичу на 16 -битне бројаче.

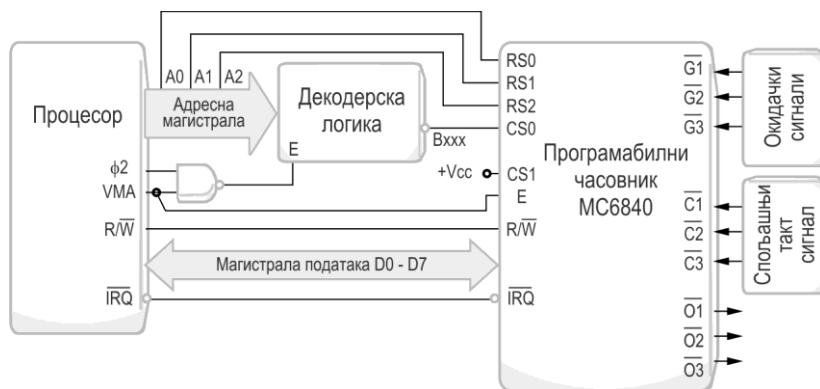
Улази за спољашњи такт сигнал (G1, G2, G3)

Улазни прикључци¹ који примају асинхроне TTL компатибилне улазне сигнале којима се декрементирају часовници 1, 2 и 3. Стабилно стање спољашњег такт сигнала не сме бити краће од трајања системског такт сигнала. У случају часовника 3 може се укључити и прескалер који дели са 8. Начин повезивања програмабилног часовника MC6840 са процесором приказан је на слици 7.18.

Управљачки регистар

Сваки од часовника има припадајући управљачки регистар у који се може уписати вредност којом се дефинише режим рада. Адресе регистра могу се видети из табеле 7.1. Управљачки регистар за часовник 2 (CR2) налази се на адреси RS0=1, RS1=0 и RS2=0. Управљачки регистри CR1

¹ Пинови

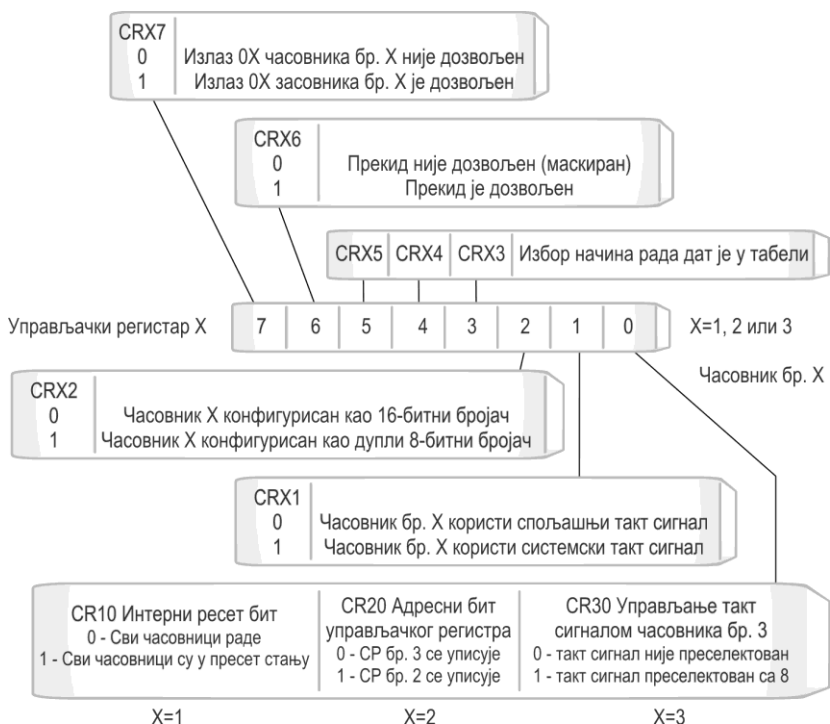


Слика 7.18. Шема повезивања програмабилног часовника са процесором

и CR3 налазе се на адреси RS0=0, RS1=0, RS2=0 и као додатни адресни бит користи се бит најмање тежине регистра CR2. У случају да је CR2=0 уписује се у управљачки регистар часовника 3. Ако је CR2=1 уписује се у управљачки регистар часовника 1. Намена битова у осмобитном управљачком регистру:

- CR7 ако је бит постављен на 1 омогућен је излаз сигнала на хардверском прикључку OX одговарајућег часовника. У супротном је забрањен;
- CR6 ако је бит постављен на 1 омогућено је генерисање захтева за прекид. У супротном је забрањено;
- CR5, CR4 и CR3 управљају режимом рада и прекидом (детаљније објашњење дато је другом делу овог одељка);
- CR2 ако је бит постављен на 0 часовник ради као 16-битни бројач у супротном ради у режиму двоструког осмобитног бројача;
- CR1 улога овог бита зависи од тога ком часовнику припадају:
 - CR10 ако је 0 сви часовници раде у програмираном режиму рада, а ако је 1 сви часовници су заустављени и остају у програмираном режиму рада (пресет стању);
 - CR20 ако је 0 омогућен је упис у управљачки регистар часовника 3, а ако је 1 омогућен је упис у управљачки регистар часовника 2;

7. Улазно/излазни начин рада



Слика 7.19 Функције битова управљачког регистра

- CR30 ако је 0 такт сигнал се директно доводи на часовник 3, а ако је 1 улазни такт сигнал се доводи преко прескалера који га дели са 8.

На слици 7.19 дата је функција сваког од битова контролног регистра. Нису приказане три независне слике пошто је једина разлика у најлакшем биту CR1 а функција овог бита за сваки од управљачких регистра је детаљно описана. Режим рада програмабилног часовника дефинише се битовима CRX5, CRX4 и CRX3. Вредност битова и одговарајући режим рада приказан је у табели 7.3.

CRX3	CRX4	CRX5	Селекција начина рада програмабилног часовника
0	0	0	Континуални режим: улази G↓, или упис у леч или ресет
1	0	0	Режим мерења учестаности: захтев за прекид ако је улазу G ↓ истекло време на које је подешен часовник

CRX3	CRX4	CRX5	Селекција начина рада програмабилног часовника
0	1	0	Континуални режим: улази G ↓ или ресет
1	1	0	Режим мерења трајања импулса: захтев за прекид ако је гејту $\downarrow \uparrow$ < истекло време на које је подешен часовник
0	0	1	Режим појединачног импулса: улази G ↓, или упис у леч или ресет
1	0	1	Режим мерења учестаности: захтев за прекид ако је улазу G $\downarrow \uparrow$ > истекло време на које је подешен часовник
0	1	1	Режим појединачног импулса: улази G ↓ или ресет
1	1	1	Режим мерења трајања импулса: прекид ако је улазу G $\downarrow \uparrow$ > истекло време на које је подешен часовник

Табела 7.3 Режији рада програмабилног часовника

Регистар статуса

Регистар статуса је интерни регистар који се може само прочитати и у њему се налазе битови за прекид (флег прекида). Битови b_0 , b_1 и b_2 одговарају часовницима 1, 2 и 3 респективно. Бит b_7 је збирни флег прекида. У случају када се деси прекид на неком од часовника одговарајући флег се поставља на 1. Брисање флега може се обавити на два начина:

- спољашњим или унутрашњим ресетом,
- секвенцом читавања регистра статуса и часовника.

Пример 7.1 Потребно је генерисати периодични правоугаони сигнал учестаности 1kHz на излазу програмабилног часовника 2 (O2). Системски такт сигнал (E) је учестаности 1MHz.

Решење:

Да би се добио тражени сигнал треба да се: часовник 2 постави у шеснаестобити континуални режим рада и упише у леч часовника 2 вредност која одговара полупериоди излазног правоугаоног сигнала. Периода излазног сигнала је 1ms. Време на које је подешен часовник смањује се за један у сваком такт сигналу. Када вредност на коју је подешен часовник дође до нуле у часовник се уписује вредност која се налази у његовом лечу, инвертује сигнал на излазу и креће нови циклус. Како је полупериода сигнала који желимо да генеришемо 0,5ms то је потребно да се на улаз

часовника доведе 500 системских тактова (потребно је да часовник одброји 500 системских тактова). Како се броји до нуле у лечеве треба уписати вредност за 1 мању, односно 499. Асемблерски програм треба да:

- обезбеди забрану прекида (шести бит управљачког регистра постављен је на 0);
- дозволи излаз часовника на линији O2 (седми бит управљачког регистра постављен на 1);
- постави системски такт сигнал као улаз часовника 2 (први бит управљачког регистра постављен на 1);
- изабере континуални режим рада при коме се часовник покреће са уписом у леч (битови 3, 4 и 5 управљачког регистра постављени су на 0);
- изабере шеснаестобитни режим рада (други бит контролног регистра постављен на 0);
- упише у леч часовника 2 вредност 499 (леч је 16-битни регистар);
- постави вредност CR10 (бит 0 управљачког регистра 1) на 0 како би се дозволио рад часовника. После ресета овај бит је на логичкој јединици и сви часовници су у пресет стању.

Како су после ресета сви часовници у пресет стању потребно је CR10 поставити на 0. Да бисмо приступили управљачком регистру бр.1 потребно је интерну адресу CR20 за избор регистара 1 или 3 поставити на 1. На основу претходних захтева дефинише се вредност коју треба уписати у управљачки регистар часовника 2 и она износи \$83. Са слике 7.18 види се да се часовник налази на адреси \$BXXX. Из табеле 7.1 види се да је интерни управљачки регистар часовника 2 на адреси \$1 па се овом регистру приступа преко адресе \$B001. Леч часовника 2 налази се на интерним адресама %100 виши бајт и %101 нижи бајт, односно \$B004 и \$B005 у меморијској мапи процесора. У управљачки регистар бр. 1 треба уписати број чији је бит 0 једнак 0. Како се часовник 0 не користи остали битови у управљачком регистру нису од интереса па ћемо их поставити на вредност 0. У интерној меморијској мапи регистар бр. 1 налази се на адреси \$0, односно у меморијској мапи

процесора на адреси \$B000 (слика 7.18). Асемблерски програм којим се могу реализовати претходно описане операције је:

```
LDAA  #$83
STAA  $B001
LDX   #$499
STX   $B004 ; аутоматски се уписује 16-битна реч у лек
CLRA
STAA  $B000 ; покретање часовника
```

Након ресета часовника и извршења програмске секвенце на излазу O2 се генерише периодични правоугаони сигнал учестаности 1kHz без даљег учешћа процесора. То је и основна улога ове компоненте јер се постиже уштеда у процесорском времену пошто се генерисање сигнала обавља без учешћа процесора.

Пример 7.2 *Претпоставимо да се користе сва три часовника и да сваки од њих може да изазове прекид. Треба да се на меморијску локацију \$F0 упише вредност \$10 ако је прекид изазвао часовник бр. 1, \$20 на меморијску локацију \$F1 ако је прекид изазвао часовник бр. 2 и \$30 на меморијску локацију \$F2 ако је прекид изазвао часовник бр. 3. Да би се решио овај задатак потребно је на уласку у потпрограма за обраду прекида читавати стање статус регистра и на основу стања флегова одредити који од часовника је захтевао прекид. Из табеле 7.3 види се да је статус регистар у интерној меморијској мапи на адреси %001 односно у меморијској мапи процесора на адреси \$B001 (слика 7.18). Како су прекиди асинхрони, потпрограма за обраду прекида проверава стање флегова за све часовнике јер је могуће да је више часовника истовремено захтевало (изазвало) прекид. Асемблерски програм који реализује постављени задатак је:*

```
IRQ_HANDLER LDAA  $B001
             CMPA  #$1   ; провера да ли је прекид изазвао часовник #1
             BNE   TIM2
             LDAB  #$10
```

7. Улазно/излазни начин рада

	STAB	\$F0	
TIM2	CMPA	#\$2	; провера да ли је прекид изазвао часовник #2
	BNE	TIM3	
	LDAB	#\$20	
	STAB	\$F1	
TIM3	CMPA	#\$4	; провера да ли је прекид изазвао часовник
	BNE	KRAJ	
	LDAB	#\$20	
	STAB	\$F1	
KRAJ	RTI		

7.11 ПИТАЊА И ЗАДАЦИ

- Директан приступ меморији односи се на:
 - способност процесора да учитава и уписује податке у било коју локацију меморије
 - способност процесора да учитава податке са сваке меморијске локације у истом времену
 - могућност улазно/излазних модула да користећи посебна интерфејсна кола размењују податке са меморијом без учешћа процесора
 - специјално пројектовано интегрисано коло које убрзава пренос података између микропроцесора и меморије
- Пренос података са улазно/излазним модулима које иницира процесор започиње:
 - када процесор има податке спремне за пренос
 - када улазно/излазни модули имају податке спремне за пренос
 - када процесор чита и провери статусне информације са улазно/излазних модула да би одредио да ли је модул спреман за пренос података
 - извршавањем инструкција READ или WRITE ка(од) улазно/излазног модула а које су укључене у програм
- Укратко описати основне функције свих портова микроконтролера MC68HC11.

4. Шта од понуђеног указује процесору где је потпрограма за обраду прекида смештен:
 - а) ресет вектор
 - б) улаз
 - в) вектор прекида
 - г) програмски бројач
5. Описати разлику између улаза за прекид.
6. Када процесор препозна да је активан улаз, зашто се аутоматски поставља 1 бит у регистру услова? Када се брише?
7. Описати метод за обраду прекида са упитом.
8. Контролер за директан приступ меморији MS6844 може да ради у *Halt Burst*, *Halt Steal* или *TSC Steal* начину рада. На који од описаних начина рада се односи опис који следи:
 - а) процесор препушта управљање магистрала после преноса сваког бајта
 - б) цео блок података је пренет док се процесор налази у HALT стању
 - в) такт сигнал процесора тренутно се „замрзава“, и не чека се да се заврши извршавање текуће инструкције
9. Код преноса са директним приступом меморији процесор не учествује директно у преносу података између меморије и улазно/излазних модула. Он учествује у постављању начина рада, параметара и обради података директног приступа меморији. Описати које операције обавља процесор пре и после завршетка процеса за директан приступ меморији.
10. Описати за шта се све може користити програмабилни часовник.
11. Колико различитих нумеричких вредности напона може да се добије на излазу 8-битног А/Д конвертора?
12. Особитни А/Д конвертор ради у опсегу 0V до 5V. Напон од 0V се конвертује у вредност \$0, а 5V у \$FF. Која вредност одговара улазном напону од 1,25V?
13. Објасните намену А/Д конвертора у микрорачунарским системима.
14. Нацртати типичан паралелни 8-битни А/Д конвертор који се користи у системима за процесно управљање базираним на микрорачунарским системима. Која се линија А/Д конвертора везује на улаз за прекид процесора?

7. Улазно/излазни начин рада

15. Објасните намену Д/А конвертора у микрорачунарским системима.
16. Описати основне карактеристике безусловног преноса и навести неки пример.
17. Која се вредност читава са излаза 8-битног А/Д конвертора, ако је на његов аналогни улаз доведен напон који је једнак половини напона напајања?
18. Која је једина врста улазно/излазног преноса података која није под контролом процесора?
19. Објаснити *Halt Steal* начин рада контролера MC6844 за директан приступ меморији.
20. Објаснити *TSC Steal* начин рада контролера MC6844 за директан приступ меморији.
21. Навести све начине рада контролера MC6844 за директан приступ меморији. Објаснити разлику између њих.
22. Нацртати електричну шему повезивања паралелног 14-битног Д/А конвертора на процесор MC68HC11. У меморијској мапи Д/А конвертор и пратећа кола (осмобитни лечеви) треба да буду на адресама \$AXXX. У декодерској логици употребите адресни декодер 3/8 (74LS138).
23. Нацртати електричну шему повезивања паралелног десетобитног А/Д конвертора на процесор MC68HC11. У меморијској мапи А/Д конвертор и пратећа кола (бафери са три стања) треба да буду на адресама \$8XXX. У декодерској логици употребите адресни декодер 3/8 (74LS138).
24. Да ли се потпрограм који обрађује прекид може прекинути? Објаснити.
25. Како се програмско генерисање променљивих временских интервала може остварити?

8. Везивање улазно/излазних уређаја

У 7. поглављу анализирали смо начине на које процесор може да комуницира са периферијским улазно/излазним уређајима. У овом поглављу биће детаљније анализирани примери повезивања процесора и различитих типова улазно/излазних уређаја. У неким примерима користе се специјална интегрисана кола са много интерфејсних функција, која значајно поједностављују пројектовање интерфејсних уређаја.

8.1 ПРАКТИЧНА РАЗМАТРАЊА ВЕЗАНА ЗА ИНТЕРФЕЈСЕ

Улазно/излазни уређај је интерфејс између рачунара и спољашњег света. Често рачунар и спољашњи уређаји раде на различит начин. Рачунар је дигитални електронски уређај који ради на одређеној учестаности такт сигнала и користи одређене нивое сигнала који представљају логичке нуле и јединице. Интерфејсна кола за спрегу са улазно/излазним уређајима претварају:

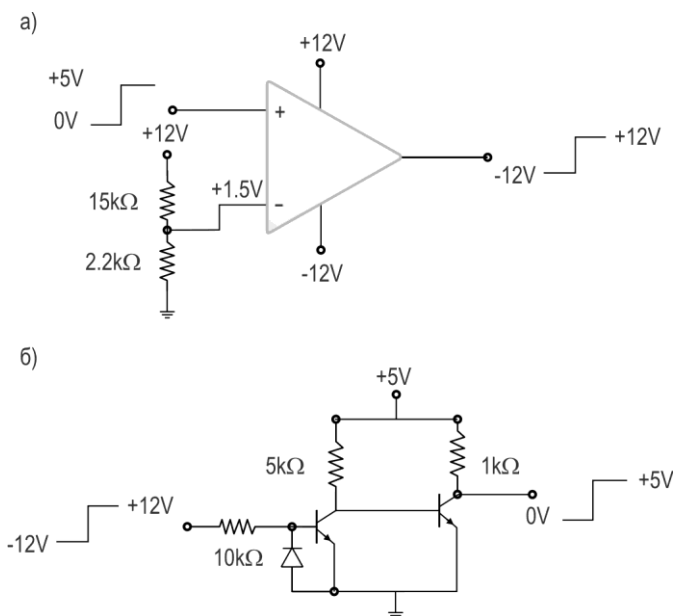
- податке на излазу из рачунара у облик који спољашњи уређај може да разуме, и
- улазне податке у облик који рачунар може да разуме.

Ова конверзија може да буде комплексан задатак. Пројектовање интерфејсних кола захтева анализу карактеристика сигнала који се преноси између рачунара и улазно/излазних уређаја као што су:

1. логички нивои,
2. оптерећење, и
3. формат.

8.1.1 Логички нивои

Дигитална кола, да би представила два логичка нивоа, користе два различита напонска нивоа. Ови напонски нивои могу се разликовати



Слика 8.1 Кола која служе за конверзију логичких нивоа између два различита система

од напонских нивоа које рачунар користи. Најчешће процесори користе напонске нивое који су од 0V до +5V за представљање логичке 0 и логичке 1. Неки улазно/излазни уређаји користе друге напонске нивое, нпр. +12V за представљање логичке 0 и -12V за представљање логичке 1. Интерфејсна кола, која се користе за такве уређаје, треба да конвертују (преводе) из једног напонског нивоа у други. Пример процеса конверзије приказан је на слици 8.1

8.1.2 Оптерећење

Кад се излазна сигнална линија веже на било које оптерећење треба водити рачуна о томе да ли излаз може да издржи такво оптерећење¹ уз прихватљиву деградацију нивоа сигнала. Већина интегрисаних кола која се користе у микрорачунарима израђена су у NMOS, CMOS и HCMOS технологији. То се односи на микропроцесоре, микроконтролере, RAM и ROM меморије, контролере за директан приступ меморији, паралелно/серијске и паралелно/

¹ Drive capabilities

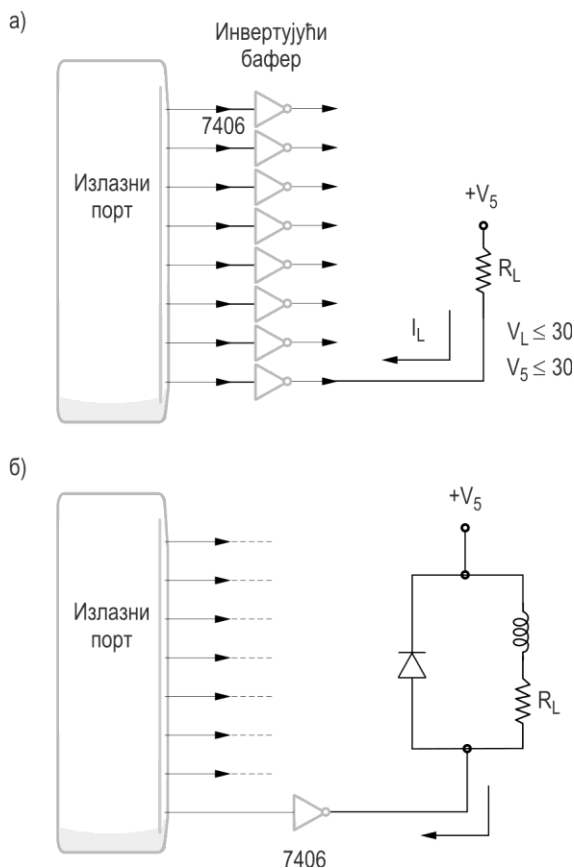
паралелне интерфејсе. Модули у NMOS, CMOS и HCMOS технологији без проблема могу да напајају NMOS, CMOS и HCMOS оптерећења која захтевају веома мале улазне струје. Ово није случај са напајањем TTL кола. У случају када се користе кола са великим улазним отпорностима нема потребе водити рачуна о струјном оптерећењу. Када се ради са колима која раде са сигнаlima високе учестаности мора се водити рачуна о улазним, паразитним, капацитивним оптерећењима логичких кола која износе неколико μF .

У микрорачунарима адресни декодери и интерфејсна кола најчешће се израђују у TTL и CMOS технологији. Три најчешће коришћене TTL серије су 74LSxx (*Low Power Schotky*), 74xx (стандардна TTL), и 74Sxx (*High Speed Shotky*). Најзаступљенији представници CMOS серије су 74HCTxx (*High-Speed CMOS са TTL компатибилним улазним напонским нивоима*) и 74HCxx (*High-Speed CMOS са CMOS компатибилним улазним напонским нивоима*). На пример, интегрисана кола са двоструким JK флип-флопом у овим серијама била би означена као 74LS76, 7476, 74S76, 74HCT76 и 74HC76. Од три поменуте TTL фамилије логичких кола, 74LSxx захтева најмању улазну струју. Типичан HCMOS излаз може да напаја десет 74LSxx улаза, два 74xx улаза и два 74Sxx улаза. Било који излаз микроконтролера MC68HC11 може да напаја 800 HCMOS улаза узимајући у обзир струјно оптерећење, али брзину рада ограничава капацитивно оптерећење.

Напајање великих оптерећења

Стандардни TTL бафери, као што су 7406 и 7407 (шест бафера у једном чипу) могу се користити за напајање оптерећења која захтевају до 30V и 40mA. На слици 8.2A приказано је како се осам 7406 инвертујућих бафера може користити за прикључивање излазног порта процесора (нпр. 8-битни излазни регистар) на ова оптерећења. На излазу 7406 су транзистори са отвореним колектором, који могу у непроводном стању да издрже напон до 30V, а када проводе могу да проведу струју до 40mA. Пошто су 7406 инвертујући бафери, логичка 0 ће закочити излазни транзистор а логичко 1 на улазу активира излазни транзистор. Супротна логика може се добити коришћењем кола 7407 које је неинвертујући бафер.

Ови бафери, када се користе за контролу индуктивних оптерећења (нпр. мотор, реле или соленоид), морају бити заштићени од индуктивног



Слика 8.2 Примена бафера за напајање великих оптерећења

пренапона који се јавља при укидању струје кроз оптерећење. Диода, која се везује паралелно са оптерећењем обезбеђује ову заштиту, одводећи индуковану струју (слика 8.26).

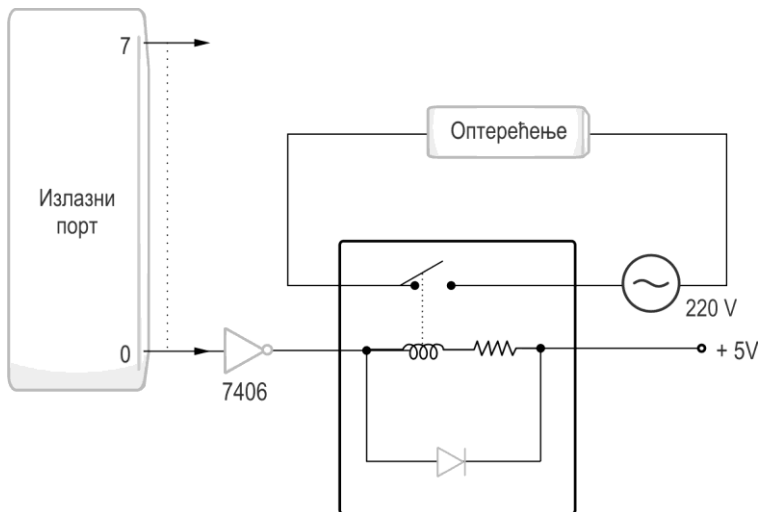
8.1.3 Галванско раздвајање

Када се излаз процесора користи за контролу оптерећења са наизменичним напајањем од 220V, потребно је изоловати оптерећење од логичких кола која управљају њима. Другим речима, излазни логички ниво процесора треба да контролише уређај са високим напонем напајања и велике снаге, без стварне електричне везе између два уређаја (галвански раздвојени).

Потребна изолација може да буде постигнута било електромагнетним релеом, било полупроводничким релеом (оптокаплером¹).

Електромагнетна (рид²) релеа могу се наћи у форми (DIP³) као било које друго интегрисано коло. Нека рид релеа могу се напајати напоном од 5V и струјом неколико mA помоћу бафера као што је на слици 8.3 приказано. Реле је индуктивно оптерећење које се побуђује када је улаз инвертујућег бафера на високом логичком нивоу. Реле управља контактима који укључују (или искључују) наизменично напајање оптерећења.

Оптокаплер садржи LED диоду и фотоосетљиви елемент, као што је фотоћелија, фотодиода, фототранзистор, и сл. На слици 8.4 приказано је неколико фотоизолатора који се најчешће користе за изоловање логичког сигнала од великог оптерећења. У случају када је логички улаз висок струја протиче кроз LED диоду, а светлосна енергија из LED диоде омогућава прикључивање напајања оптерећењу. На слици 8.4 фотодиода се понаша



Слика 8.3 Рид реле у функцији интерфејса између излазних логичких нивоа и наизменичног оптерећења

¹ Фотоелектрично реле

² Reed

³ Dual In line Package

као фотодетектор који обезбеђује струју транзистору у Дарлингтоновој спрези. На слици 8.46 је LASCR¹ који се понаша и као фотодетектор и као прекидачки елемент. Може се користити да прикључи напајање оптерећењу, али само за време половине напонског циклуса. Када се користи оптоизолатор са опоттриаком² оптерећење може добити потпуни напонски циклус (слика 8.4в).

8.1.4 Формат сигнала

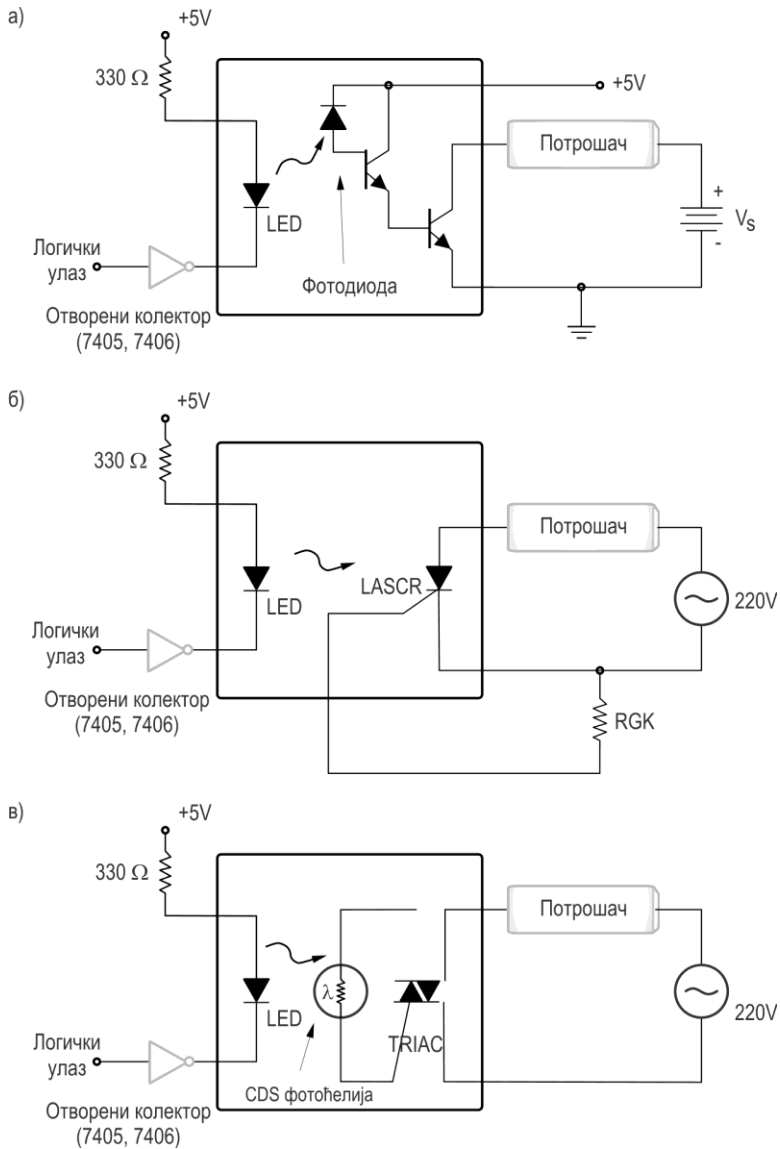
Процесор ради са дигиталним подацима у паралелном формату у коме се истовремено шаље свих осам битова 8-битне речи. Многи спољашњи уређаји не користе паралелни, дигитални формат. Велика група уређаја користи аналогне сигнале. Ови уређаји, да би комуницирали са процесором, захтевају коришћење интерфејсних кола као што су Д/А и А/Д конвертори.

Друга некомпатибилност формата сигнала јавља се када спољашњи уређај шаље и/или прима дигиталне податке у серијском формату, што значи 1 бит у јединици времена. Модем је пример уређаја који користи серијски формат података. Када процесор комуницира са серијским уређајем постоје два основна метода за превазилажење ове некомпатибилности. По првом методу процесор комуницира са серијским уређајем само преко једне линије магистрале података. Подаци се из процесора ка уређају (и обрнуто) шаљу преко те линије за податке у серијском облику.

Овај метод представља софтверско решење и захтева велики број програмских инструкција да би процесор слао и примао серијске податке у тачном временском редоследу. Иако овај метод захтева минимум спољашњег хардвера, недостатак је што захтева програм за серијско/паралелну конверзију. Такође, овим начином, процесору се одузима много времена за серијски пренос. За споре уређаје, као што је серијски штампач, време потребно да се пошаље један ASCII кодирани карактер је у опсегу од 1ms. У неким апликацијама овај се начин коришћења времена процесора не дозвољава, и неопходно је користити хардверско решење за паралелно/серијску и серијско/паралелну конверзију. Овај приступ је важан и детаљно ћемо га анализирати у следећем одељку.

¹ *Light activated SCR (silicon-controlled rectifier или semiconductor-controlled rectifier).*

² TRIAC (*TRIode for Alternating Current*)



¹ CdS (Cadmium Sulfide Photocell)

8.2 СИНХРОНИЗАЦИЈА У ПРЕНОСУ ПОДАТАКА

Код серијске комуникације сигнализациони елементи шаљу се на линију један по један. Сваки сигнализациони елемент може да буде:

- мањи од једног бита,
- један бит,
- више од једног бита.

Због једноставности у даљем објашњавању претпоставићемо случај од једног бита по сигнализационом елементу (уколико није другачије наглашено). Синхронизација је један од кључних задатака у преносу података. Пријемник треба да:

- препозна почетак и крај блока битова,
- зна трајање сваког бита,
- узима узорак (одмерак) у тачним тренуцима времена и на тај начин читава сваки бит.

Типична ситуација је она у којој пријемник одмерава линију у средини сигнализационог (битског интервала). Уколико постоји разлика у мерењу времена, рецимо за 5% између предајника и пријемника, прво одмеравање биће померено 0,05 битског времена у односу на центар бита. На крају десетог одмеравања пријемник може да погрешити. За мању разлику у времену пријемник неће погрешити, али докле год они нису на неки начин синхронизовани пријемник неће бити усаглашен са предајником.

За синхронизацију уобичајена су два приступа. Први се назива асинхрони пренос. У овом начину комуникације шаље се један по један карактер, а карактери су пет до осам битова дугачки. Синхронизација се мора постићи само у оквиру сваког карактера. На почетку сваког карактера пријемник има прилику да се поново синхронизише. Алтернатива је синхрони пренос. Велики блокови података шаљу се као јединице или нуле и пријемник мора дужи период времена да буде синхронисан са предајником.

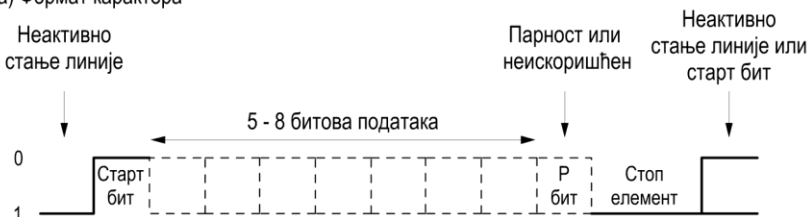
Пред синхронизацију се постављају следећи задаци:

- битска синхронизација - препознаје почетак и крај сваког бита,
- синхронизација карактера или речи - препознаје почетак и крај сваког карактера или друге мале групе података,
- синхронизација блокова или порука - препознаје почетак и крај великих група података.

8.2.1 Асинхрони серијски пренос података

Један начин за решавање проблема синхронизације је слање мањих блокова бита на почетку сваког блока за поновну синхронизацију (ресинхронизацију). Техника позната као старт-стоп или асинхрони пренос ради управо то. Асинхрони пренос заснован је на слању карактера¹ дужине 5-8 бита (слика 8.5).

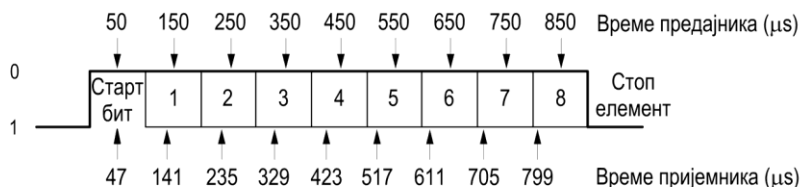
а) Формат карактера



б) 8-битна асинхрона поворка бита



в) Ефекат померања тренутка одмеравања



Слика 8.5. Асинхрони пренос

¹ Користи се и термин карактер оријентисан.

8. Везивање улазно/излазних уређаја

Када се не преносе карактери линија између предајника и пријемника је неактивна¹. Дефиниција неактивног стања на линији је конвенција и типично је еквивалентна сигнализационом елементу за бинарно 1. Пријемник испитује када долази до транзиције са 1 на 0 (старт) што указује на почетак карактера, затим узима узорак сигнала у једном битском интервалу и то понавља за свих осам интервала. Крај карактера (стоп) означава се једном до две битске позиције вредности бинарно 1. Затим очекује нову транзицију са 1 на 0. На пример ASCII карактер се обично шаље као целина од 8 битова, узимајући у обзир и бит парности. Уколико је пријемник 5% бржи или спорији од предајника 10. информациони бит ће се одмеравати са 45% померајем и још увек ће бити коректно идентификован (слика 8.5в). Посматрајмо на пример пренос брзином од 10kb/s, где је сваки бит ширине 100μs. Претпоставимо да је пријемник бржи 6% по битском интервалу од предајника, односно 6 μs по битском интервалу. Грешка као што је ова резултује у две грешке. Прво, последњи одабрани бит је неисправно примљен. Друго, бројач битова неисправно одбројава (осмо одбројавање у време седмог бита). Уколико је бит 7 јединица, а бит 8 је нула, онда је то грешка рама², пошто се карактер заједно са старт и стоп елементима назива рама³. Такође, уколико за време неактивног стања шум проузрокује лажно појављивање старт бита може да наступи грешка рама⁴. Слика 8.5 показује да што је већи рама то је и кумулативана грешка већа.

Асинхрона комуникација је једноставна и јефтина, али захтева 2 - 3 бита више по карактеру. На пример за седмобитни код, користећи једнобитни стоп елемент, на сваких девет битова два не носе никакву информацију, тако да је губитак⁵ $2/9=22\%$. Наравно, слањем великих блокова између старт и стоп елемената проценат губитка може се смањити.

Као што се на слици 8.5 види што је већи блок битова то је акумулирана грешка већа. За велике блокове битова користи се друга врста синхронизације, која се зове синхрони пренос.

Када рачунар (у ствари било који уређај) треба да комуницира са

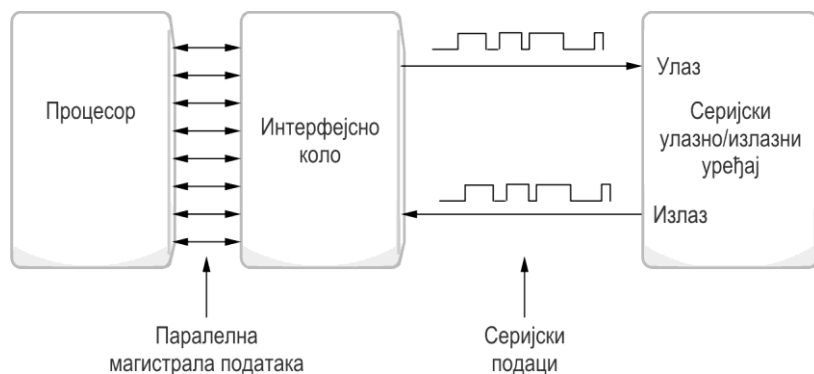
¹ *Idle state* – јалово, неактивно стање

² *Framing error*

³ *Frame*

⁴ Зато постоји СТОП елемент.

⁵ *Overhead*



Слика 8.6 Интерфејс процесора и серијског уређаја

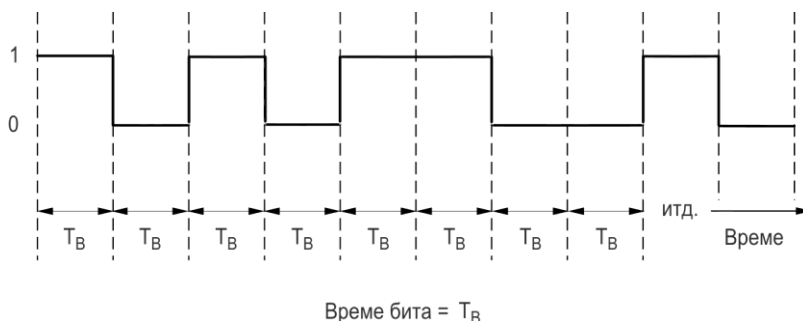
серијским улазно/излазним уређајем, уобичајен начин је асинхрони серијски пренос. Назива се асинхрони зато што предајник може да почне да шаље податке ка пријемнику у било ком тренутку без претходног синхронизовања са пријемником. На слици 8.6 приказан је блок дијаграм система за пренос података. Систем извршава две основне операције:

1. Узима 8-битну паралелну реч са магистрале података процесора и претвара је у серијску реч податка коју шаље ка серијском уређају;
2. Узима серијски податак из серијског уређаја и претвара га у 8-битну паралелну реч податка коју шаље ка процесору преко магистрале података.

Јасно је, главна функција интерфејсног кола је претварање паралелних у серијске податке и обрнуто. Пре него што пређемо на анализу како се ова конверзија може извршити, погледаћемо како изгледа сигнал који представља серијски формат података.

8.2.2 Серијски формат података

Серијски сигнал података (слика 8.7) подељен је у временске интервале који се називају трајање бита или битски интервал. За време сваког битског интервала (T_b) вредност сигнала може да буде 0 или 1, а сигнал може да мења нивое на почетку сваког битског интервала.



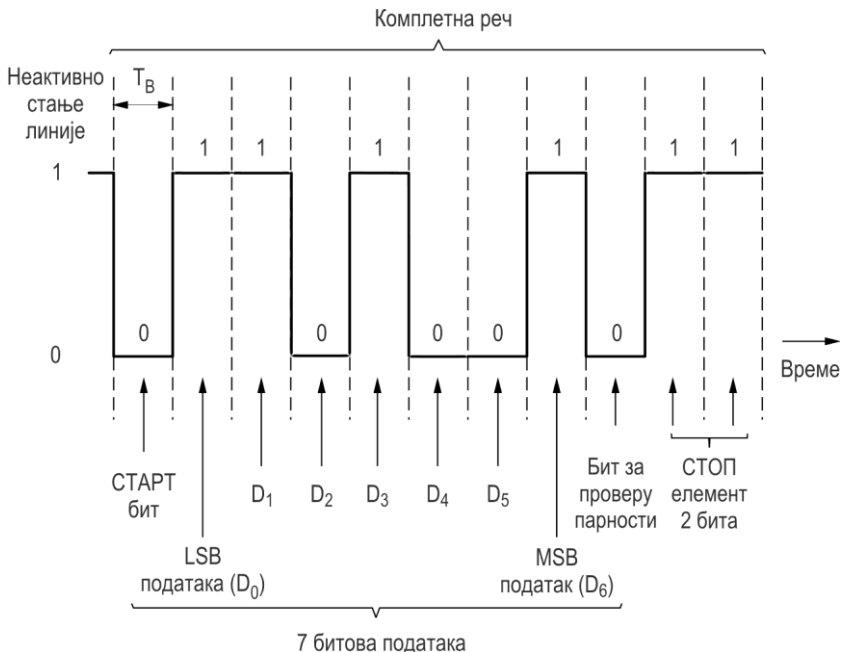
Слика 8.7 Серијски пренос података

Када се асинхрони серијски подаци преносе између два уређаја, да би се пренела једна реч податка користи се стандардни формат (слика 8.8а) који се састоји од три (или опционо четири) дела:

1. СТАРТ бит, који је увек 0;
2. пет до осам битова података, који представљају стварну информацију која се преноси;
3. опциони бит парности који се користи за детекцију грешке. Уколико је бит парности укључен, може се користити парни или непарни број јединица;
4. СТОП елемент који може да буде величине један или два бита.

За задати систем број битова података, опција са битом парности, и број СТОП битова зависе од пројектанта. На слици 8.8б приказан је пример серијске речи која користи 7 битова података што је ASCII код за алфанумеричке податке.

Потпуни серијски податак са слике 8.8в почиње СТАРТ битом који је 0. Кад се на линију не шаље податак, на линији је висок логички ниво. Почетак слања податка означава се преласком са 1 на 0, што је знак да се појавио СТАРТ бит. Иза СТАРТ бита је 7 битова податка, који почињу битом најмање тежине (LSB) и завршавају се битом највеће тежине (MSB). Тако је стварни податак који послат је 1001011, што је ASCII код за карактер К. Иза битова података је бит парности, који је у нашем примеру 0, пошто 7 битова податка садржи паран број јединица. Иза бита парности је СТОП елемент (2 бита), који увек има вредност 1.



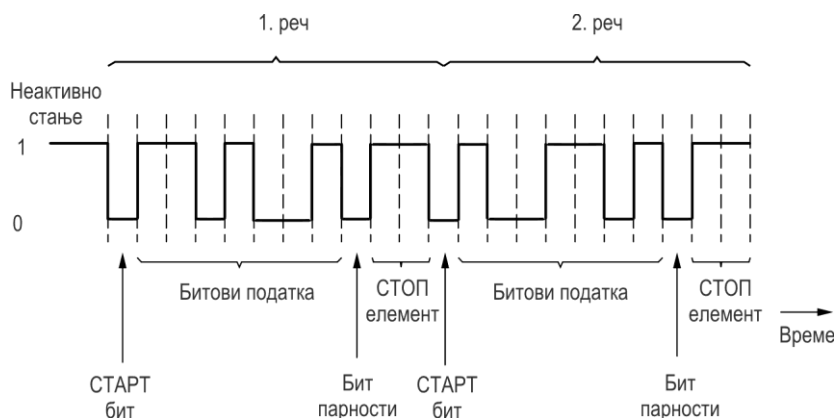
Слика 8.8 Пример серијске речи податка која користи седам битова податка, бит парности (парни), и СТОП елемент (2 бита). Податак који је овде представљен је 1001011, што је ASCII код за слово K.

Асинхрона серијска комуникација обично се користи за пренос неколико узастопних речи из једног у други уређај. Слика 8.9 показује серијски сигнал за пренос две узастопне речи. Прва реч је иста као на слици 8.8в. После СТОП елемента прве речи, долази СТАРТ бит друге речи иза кога су 7 битова податка, бит парности, и СТОП елемент. Битови податка друге речи су 1011001, што је ASCII код за Y.

Када се серијски подаци шаљу један за другим СТАРТ бит сваке нове речи појављује се одмах иза последњег СТОП елемента претходне речи. Овим је представљена максимална брзина при преносу података.

Када се серијски подаци не шаљу максималном брзином предајник шаље константан висок логички ниво (сигнал¹) између СТОП елемента последње речи и СТАРТ бита следеће речи.

¹ Користе се термини сигнал (*mark*) и пауза (*space*).



Слика 8.9 Слање серијског сигнала за две узастопне речи податка

Модул који прима серијски сигнал синхронисаће се са негативном транзицијом коју формира СТАРТ бит прве речи. Он онда зна да су следећих десет битова: податак, парност и СТОП елемент. После пријема СТОП елемента, пријемни уређај чека на следећу негативну транзицију, знајући да она представља СТАРТ бит следеће речи. На овај начин су СТАРТ бит и СТОП елемент употребљени да би синхронизовали серијски пријемник података на серијски предајник података.

8.2.3 Линијска брзина

Брзина којом се одвија серијска предаја назива се линијска брзина и изражава се у бодима¹ (Bd).

$$R \text{ [Bd]} = \frac{1}{\text{Време између транзиција}}$$

На пример ако се сигнал мења сваке милисекунде² линијска брзина ће бити:

$$R = \frac{1}{1\text{ms}} = 1000\text{Bd}$$

¹ Baud

² Среће се под именом сигнализациони интервал.

Линијска брзина мера је промене линијског (серијског) сигнала и релативни показатељ потребног пропусног опсега који се захтева за верни пренос сигнала. Већа брзина указује да се линијски сигнал брже мења и да се захтева већи пропусни опсег од система за пренос.

Некада (не увек¹) линијска брзина еквивалентна је брзини којом се битови преносе (битска брзина). Тада се 1 бит шаље у временском интервалу једнаком битском интервалу T_B , па се брзина преноса податка (D) може изразити следећом једначином:

$$D = \frac{1}{1T_B} \text{ [b/s]}$$

У том случају је линијска брзина R (изражена у Bd) једнака битској брзини D (израженој у b/s). За правилно тумачење примљених података неопходно је да пријемник серијских података ради истом линијском брзином као предајник.

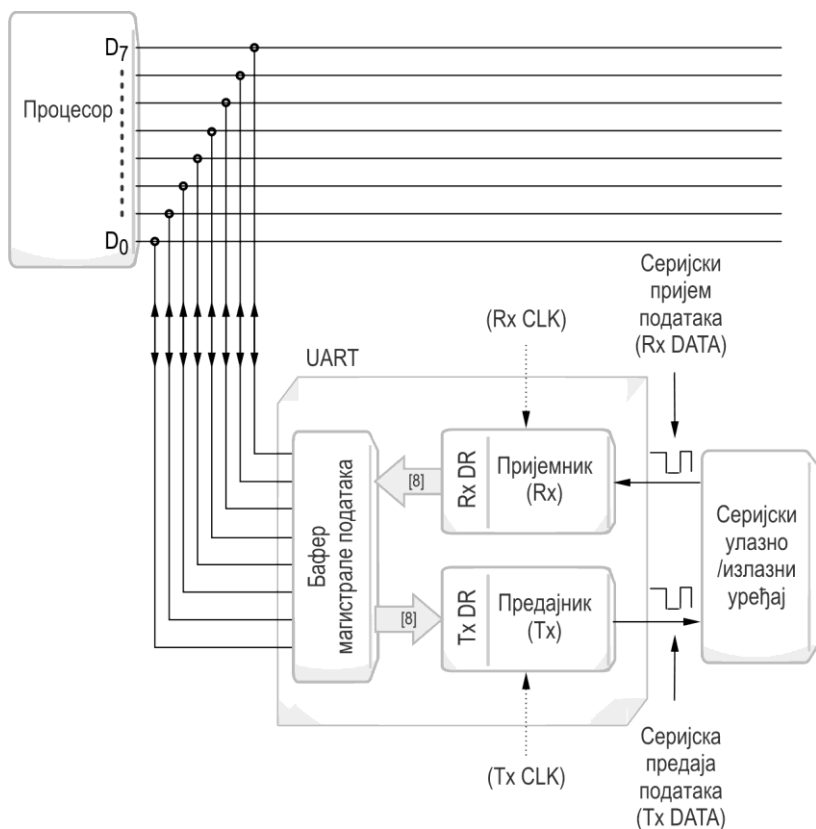
8.3 ПАРАЛЕЛНО/СЕРИЈСКИ ИНТЕРФЕЈС

Сада када знамо формат који се користи у асинхроно серијском преносу података, анализираћемо VLSI коло које се среће под називом универзални асинхрони примопредајник (UART²). Он се користи да обави серијско/паралелну конверзију када процесор захтева комуникацију са серијским улазно излазним/уређајем (слика 8.10). Иако постоје мале разлике између различитих произвођача универзалних асинхроних примопредајника, сви садрже следеће основне елементе:

1. Серијски пријемник (Rx) који узима податке са серијског улаза, претвара их у паралелни формат и ставља у пријемни регистар података (RxDR) за слање ка процесору;
2. Серијски предајник (Tx) који прима паралелну реч из предајног регистра података (TxDR) и претвара је у серијски формат, спреман за предају;

¹ Када је реч о тзв. вишенивоским модулацијама онда се у току једног сигнализационог интервала преноси више информација. Када се у току једног сигнализационог интервала преноси један бит реч је о једнонивоској модулацији, два двонивоској... Код једнонивоске модулације бодска и битска брзина су исте. То није случај код вишенивоских модулација.

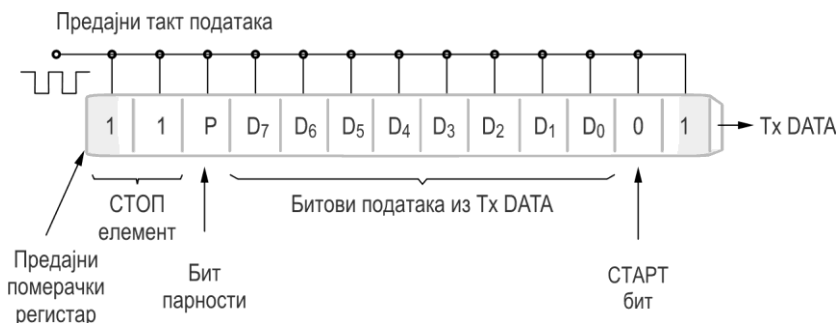
² *Universal Asynchronous Receiver Transmitter*



Слика 8.10 Основна структура универзалног асинхроног примопредајника

3. Двосмерни бафер података који пропушта паралелне податке од процесора ка предајном регистру података (TxDR), или из пријемног регистра података (RxDR) ка процесору преко системске магистрале података;
4. Спољашњи улазни такт сигнали: предајни (RxCLK) и пријемни (TxCLK).

Када процесор извршава команду за слање података да би послао податак ка серијској периферији, он шаље податак ка предајном регистру асинхроног примопредајника података (TxDR). Управљачка логика у предајном (Tx) делу узима овај податак, додаје СТАРТ бит, бит парности



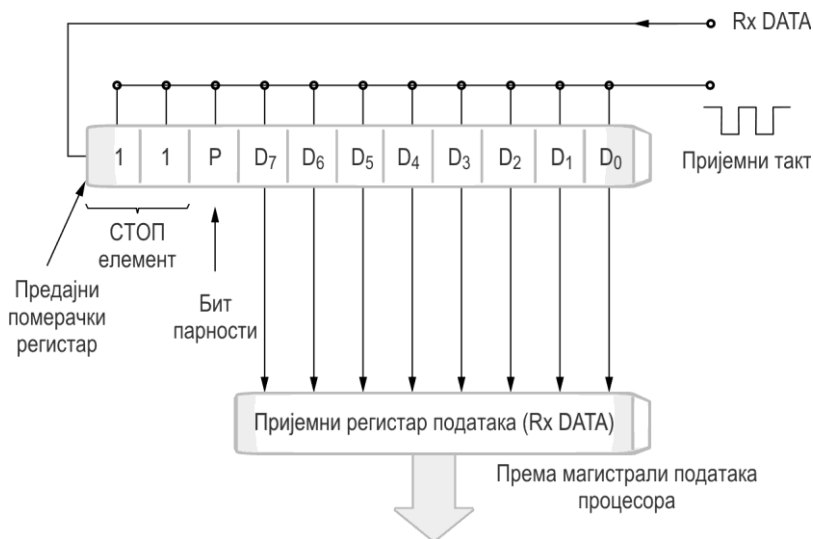
Слика 8.11 ASCII кодирана реч податка у предајном померачком регистру

(ако се користи) и жељени број битова СТОП елемента. Комплетна реч затим се ставља у регистар који се назива предајни померачки¹ регистар (слика 8.11). Садржај овог регистра помера се брзином одређеном предајним тактом података. На тај начин формира се серијска реч података (TxDATA), која се шаље ка излазном уређају.

Када серијски улазни уређај треба да пошаље податке ка процесору он шаље серијски податак ка пријемном делу асинхроног примопредајника кроз његов улаз (RxDATA). Када управљачка логика на улазној линији RxDATA детектује прелаз са високог на низак ниво интерпретира га као СТАРТ бит. Затим помера остатак серијске речи податка у пријемни померачки регистар, брзином која је одређена пријемним такт сигналом података (слика 8.12). Такт сигнал података изводи се из такт сигнала TxCLK и RxCLK.

Када је комплетна серијска реч у померачком регистру информациони део (D_7 - D_0) паралелно се пребацује у пријемни регистар података (RxDR). Процесор ће прочитати садржај пријемног регистра података преко системске магистрале података као са било које меморијске локације. За правилан рад асинхроно серијског система преноса података (слике 8.10) асинхрони примопредајник и серијски улазно/излазни уређај треба да имају усаглашен формат података. То су: број информационих битова и бит парности. Ове податке обично процесор програмира у асинхроном примопредајнику. Асинхрони

¹ Shift



Слика 8.12 Пребацивање података из померачког регистра у паралелни регистар података

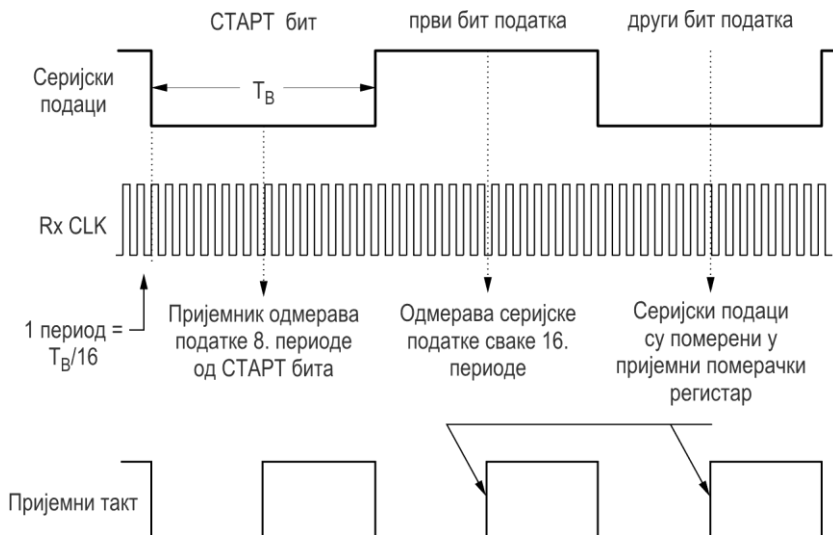
примопредајник има интерни управљачки регистар (слика 8.14) у који процесор може да уписује садржај као у било коју меморијску локацију.

8.3.1 Синхронизација пријемног дела

Као што је наглашено, пријемник се иницијално синхронизује на негативну транзицију коју иницира СТАРТ бит. Ово указује пријемнику да следе информациони битови, битови парности и СТОП бит (битови). Да би се обезбедила синхронизација асинхронног примопредајника користи спољашњи такт сигнал који је веће учестаности него што је линијска брзина (обично 16 пута). На пример ако је линијска брзина 9600Bd, спољашњи пријемни такт сигнал (RxCLK) има учестаност $16 \cdot 9600 = 153,6 \text{ kHz}$. Ово је учестаност такт сигнала која се доводи на улаз RxCLK асинхронног примопредајника. Једна периода такт сигнала RxCLK једнака је $1/16$ битског интервала, тако да је такт сигнал $RxCLK = T_b / 16$. Пошто детектује прву негативну транзицију пријемник чека 8 периода такт сигнала RxCLK и затим тестира серијски улаз да би видео да ли је још увек низак. То потврђује пријемнику да је у питању СТАРТ бит, а не сметње на серијској линији. Уочимо да је

овај узорак узет приближно на средини СТАРТ интервала, пошто се осам такт сигнала RxCLK јавља на половини битског интервала. После провере СТАРТ бита пријемник одабира серијске податке сваких 16 периода такт сигнала RxCLK (приближно један битски интервал), дајући одмерке који су постављени приближно на средини сваког битског интервала. Сваки од ових битова затим се помера у пријемни померачки регистар са такт сигналом који је једнак $RxCLK/16$. Уочимо да је ово пријемни такт сигнал који треба да се изједначава са линијском брзином.

Једном када је цела реч податка померена у пријемни померачки регистар кола у синхронном примопредајнику аутоматски тестирају да ли први бит СТОП елемента има вредност 1. Ако тај бит има вредност 0 поставља се ознака (флег) грешке рама. Ова ознака део је интерног регистра статуса асинхроног примопредајника који процесор може да прочита. У потпрограму за читавање пријемне речи из асинхроног примопредајника процесор може да читава статусни регистар примопредајника. Уколико процесор чита да је наступила грешка рама, може да започне одговарајуће активности за њено превазилажење.



Слика 8.13 Пријемник одмерава серијски податак користећи такт сигнал RxCLK, који је учестаности 16 пута веће од линијске брзине

Када је бит парности укључен кола асинхроног примопредајника проверавају његову вредност у примљеном податку. Уколико је дефинисана парна парност, кола асинхроног примопредајника одређују да ли пријемни податак (укључујући бит парности) садржи паран број јединица. Уколико не садржи паран број јединица поставља се ознака (флег) грешке парности. Ова ознака такође је део статусног регистра који процесор тестира у свом потпрограму. Кад процесор констатује грешку, може да започне одговарајуће активности.

Као што је приказано на слици 8.12 битови података су пренети паралелно у пријемни регистар RxDR, који чува податке за процесор. Кад се пренос у регистар RxDR заврши, пријемник чека на пријем следеће речи серијског улазног модула и помера је у пријемни померачки регистар. Друга реч неће бити пренета у регистар RxDR док процесор из њега не прочита претходно смештену реч. Може се догодити да се трећа серијска реч помери у пријемни померачки регистар, пре него што се друга реч пренесе у регистар RxDR, и на тај начин изгуби друга реч. Када се ово деси асинхрони примопредајник поставиће одговарајућу ознаку грешке у свој регистар статуса¹.

Грешка која наступа због губитка податка може се спречити на тај начин што ће процесор прочитати податке из регистра RxDR за исто време колико је потребно новим серијским подацима да се учитају у пријемник. Ово време зависи од брзине која се користи.

8.3.2 Асинхрони серијски примопредајник MC6850

У овом поглављу детаљно ћемо анализирати специфично интегрисано коло - асинхрони серијски примопредајник MC6850 који се у спецификацији произвођача (*Motorola*) означава као асинхрони комуникациони адаптер (ACIA²). Детаљном анализом овог интегрисаног кола илустроваћемо начин рада који је применљив и на друге асинхроне примопредајнике. На слици 8.13 приказан је блок дијаграм асинхроног примопредајника MC6850. На блок дијаграму уоквирено је шест важних функција и то су:

¹ *Overrun Error* - грешка због заузетости

² *Asynchronous Communications Interface Adapter*

1. селектовање и читавање/уписивање (CHIP SELECT и READ/ WRITE),
2. бафер магистрале података¹,
3. предајни регистар података (TxDR) и предајни померачки регистар (TSR),
4. пријемни регистар података (RxDR) и пријемни померачки регистар (RSR),
5. статусни регистар, и
6. управљачки регистар.

Начин рада регистара TxDR, TSR, RxDR и RSR и бафера магистрале података објашњени су у претходном поглављу.

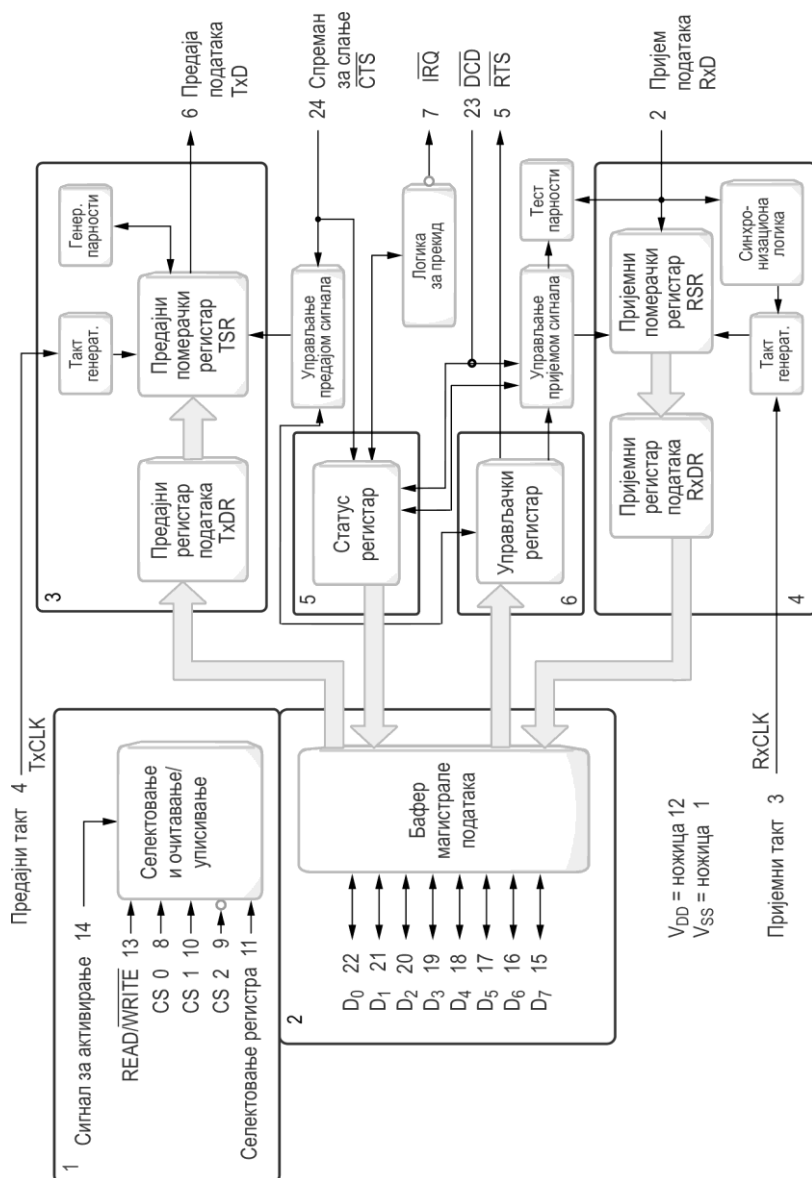
Селектовање и читавање/уписивање (CHIP SELECT и READ/ WRITE)

Процесор може да ради са великим бројем улазно/излазних уређаја. Три улаза за селектовање интегисаног кола (CS0, CS1, $\overline{\text{CS2}}$) јесу улазне линије које се користе да би се адресирао (селектовао) одређени асинхрони примопредајник. Асинхрони примопредајник се селекује када су CS0 и CS1 на високом нивоу, а $\overline{\text{CS2}}$ на ниском нивоу. Једном када је асинхрони примопредајник селектован процесор може да обавља једну од четири операције:

1. Читавање регистра статуса. У регистар статуса асинхроног примопредајника смештају се информације које се односе на грешке и статус регистара RxDR и TxDR. Процесор добија ове информације када се извршава операција читавања регистра статуса.
2. Читавање регистра RxDR. Када асинхрони примопредајник прими серијску реч од улазног уређаја процесор може да добије ове податке читавањем регистра RxDR.
3. Уписивање у управљачки регистар. Управљачки регистар асинхроног примопредајника је 8-битни регистар у који се може само уписивати². Садржај му је одређен бројем са којим треба да се дели улазни такт сигнал, величином речи, бројем битова

¹ Data bus buffer

² Write only



Слика 8.14 Блок шема асинхроног серијског примопредајника MC6850

СТОП елемента и битом парности. Процесор је тај који уписује ову 8-битну реч у управљачки регистар.

4. Уписивање у регистар TxDR. Када процесор треба да пошаље реч податка ка излазном уређају, он уписује податак у регистар TxDR. Асинхрони примопредајник тај податак шаље серијском уређају.

Ове четири операције процесора укључују читавање или уписивање садржаја регистра RxDR и TxDR, статусног и управљачког регистра.

Стварна операција која се извршава у било ком тренутку одређена је нивоима сигнала на улазима за селектовање интегрисаног кола¹ (CS), за селектовање регистра² (RS), читавање/уписивање ($R/\overline{W}$) и на улазу за активирање (ENABLE), а према табели 8.1.

За сваку од четири операције интегрисано коло асинхронног примопредајника мора да буде селектовано ($CS0=1$, $CS1=1$, $\overline{CS2}=0$). Ови улази се нормално задају из кола за декодовање адресе. Поред тога, линија за активирање мора да буде на високом нивоу. Овај улаз обично је везан за Е-такт сигнал, тако да се сав пренос података одвија када је Е-такт сигнал на високом нивоу. Подсетимо се да је ово ситуација у којој су подаци стабилни на магистрали података.

За операцију читавања статусног и RxDR регистра улаз $R/\overline{W}$ мора да буде на високом нивоу. Ниво сигнала на улазу RS одређује који ће од регистра (статусни или TxDR) бити прочитан. Улаз $R/\overline{W}$ биће на ниском нивоу за обе операције уписивања, а ниво сигнала на улазу RS одређује у који ће се од регистра, TxDR или управљачки, реч податка уписати.

Операција	CS0	CS1	$\overline{CS2}$	RS	$R/\overline{W}$	ENABLE
Учитавање статус регистра	1	1	0	0	1	1
Уписивање у управљачки регистар	1	1	0	0	0	1
Учитавање садржаја регистра RxDR	1	1	0	1	1	1
Уписивање у регистар TxDR	1	1	0	1	0	1

Табела 8.1 Операције процесора у зависности од стања на улазима асинхронног серијског примопредајника

¹ Chip Select

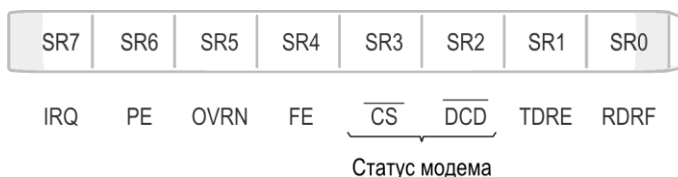
² Register Select

Статусни регистар

Статусни регистар асинхроног примопредајника MC6850 је 8-битни регистар који се самоочитава¹ (слика 8.15).

Функције различитих битова су следеће:

- Пријемни регистар података је пун² (RDRF). Овај бит се аутоматски поставља на вредност 1 када пријемни померачки регистар пребаци свој садржај у регистар RxDR. Овај бит се брише (поставља на вредност 0) пошто процесор чита садржај регистра RxDR, дајући на знање да садржај регистра RxDR није више потребан.
- Предајни регистар података је празан³ (TDRE). Овај бит се поставља на вредност 1 када се садржај регистра TxDR пренесе у предајни померачки регистар. Низак ниво указује да је регистар TxDR још увек пун и да у овом тренутку процесор ка асинхроним примопредајнику не треба да пошаље следећу реч за слање излазном уређају.
- Детектован носилац података⁴ ($\overline{\text{DCD}}$) и спреман за слање⁵ ($\overline{\text{CTS}}$) јесу специјални статусни битови који се користе у комуникацији са модемом. У једном од следећих поглавља биће више речи о модемима и комуникацији преко телефонских линија.



Слика 8.15 Садржај статусног регистра

¹ Read only

² Receive Data Register Full

³ Transmit data register Empty

⁴ Data Carrier Detect

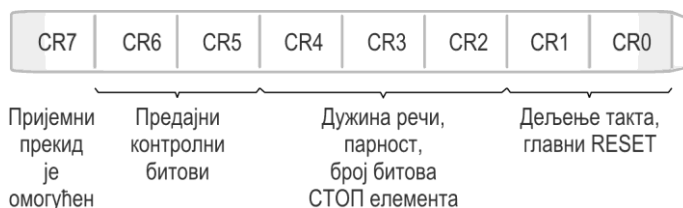
⁵ Clear to Send

- Грешка у раму¹ (FE), грешка због губитка података² (OVRN), грешка у парности³ (PE). О овим битовима било је речи у претходном одељку.
- Захтев за прекид (IRQ). Овај бит увек указује статус $\overline{\text{IRQ}}$ излазне линије асинхроног серијског примопредајника. Када је излаз $\overline{\text{IRQ}}$ на ниском нивоу одговарајући бит у статус регистру биће на високом нивоу, и обрнуто. Ово је корисно када је већи број уређаја везан за линију процесора за прекид. Када је генерисан прекид процесор може да прочита статусни регистар асинхроног примопредајника и види да ли IRQ бит има вредност 1. Уколико јесте 1, то значи да је $\overline{\text{IRQ}}$ излазна линија асинхроног примопредајника постављена на вредност 0, и да је асинхрони примопредајник уређај који је генерисао прекидни сигнал процесора. Бит IRQ се брише кад процесор изврши операцију читавања регистра RxD_R или уписивања у регистар TxD_R.

Управљачки регистар

Садржај управљачког регистра приказан је на слици 8.15, а његова намена биће описана у тексту који следи. Процесор ће уписати одговарајућу реч у управљачки регистар да би она дефинисала карактеристике серијског преноса.

Битови којима се селекује број за дељење учестаности су CR1 и CR0. Подсетимо се (слика 8.13) да постоји одређени однос пријемног такт сигнала RxCLK и пријемног такт сигнала података који се користи да



Слика 8.16 Садржај управљачког регистра

¹ Framing Error

² Receiver OverRun

³ Parity Error

8. Везивање улазно/излазних уређаја

CR1	CR0	Делилац такта (SCLK)
0	0	1
0	1	16
1	0	64
1	1	Главни ресет асинхроног примопредајника

Табела 8.2 Намена CR1 и CR0 бита управљачког регистра

помера примљене податке у асинхроном примопредајнику. Овај однос може да буде 1, 16 или 64 у зависности од вредности CR1 и CR0 бита, као што је приказано у табели 8.2. Важно је уочити да се пријемни такт сигнал податка добија дељењем пријемног такт сигнала (RxCLK) делиоцем такта (SCLK), и једнак је линијској брзини серијских података. На пример, уколико је линијска брзина 9600Bd и користи се однос дељења који је 16, учестаност RxCLK мора да буде $9600 \times 16 = 153,6 \text{ kHz}$. Ово је учестаност која се мора довести на улаз RxCLK. Исто важи и за TxCLK улаз.

Битови за селектовање формата (CR4, CR3, CR2) утичу на дужину речи, парност и број битава СТОП елемента. Табела 8.3 показује намену сваке од комбинација ових битава.

Битови за контролу предајника су CR6 и CR5. Најважнија стања за функције које се анализирају у овом поглављу приказана су у табели 8.4. Излаз RTS асинхроног примопредајника може се користити за било коју функцију, иако се најчешће користи за модемске комуникације.

CR4	CR3	CR2	Битови података	Контрола парности	Број битава СТОП елемента
0	0	0	7	Паран	2
0	0	1	7	Непаран	2
0	1	0	7	Паран	1
0	1	1	7	Непаран	1
1	0	0	8	Нема	2
1	0	1	8	Нема	1
1	1	0	8	Паран	1
1	1	1	8	Непаран	1

Табела 8.3 Намена CR2, CR3 и CR4 бита управљачког регистра

CR6	CR5	Функција
0	0	$\overline{RTS}=0$; Предајни прекид онемогућен
0	1	$\overline{RTS}=0$; Предајни прекид омогућен
0	1	$\overline{RTS}=1$; Предајни прекид онемогућен

Табела 8.4 Намена CR5 и CR5 бита управљачког регистра

Процесор може да излаз $\overline{RTS}$ постави (висок ниво) или обрише (низак ниво) преко управљачког регистра асинхроног примопредајника. Када је предајни прекид омогућен, $\overline{IRQ}$ излазна линија асинхроног примопредајника пашће на низак ниво кад год статусни бит TDRE пређе на висок ниво, указујући да је регистар TxDR празан и спреман за пријем нове речи податка из процесора. Ова $\overline{IRQ}$ излазна линија може прекинути процесор сваки пут кад је статусни бит TDRE=1.

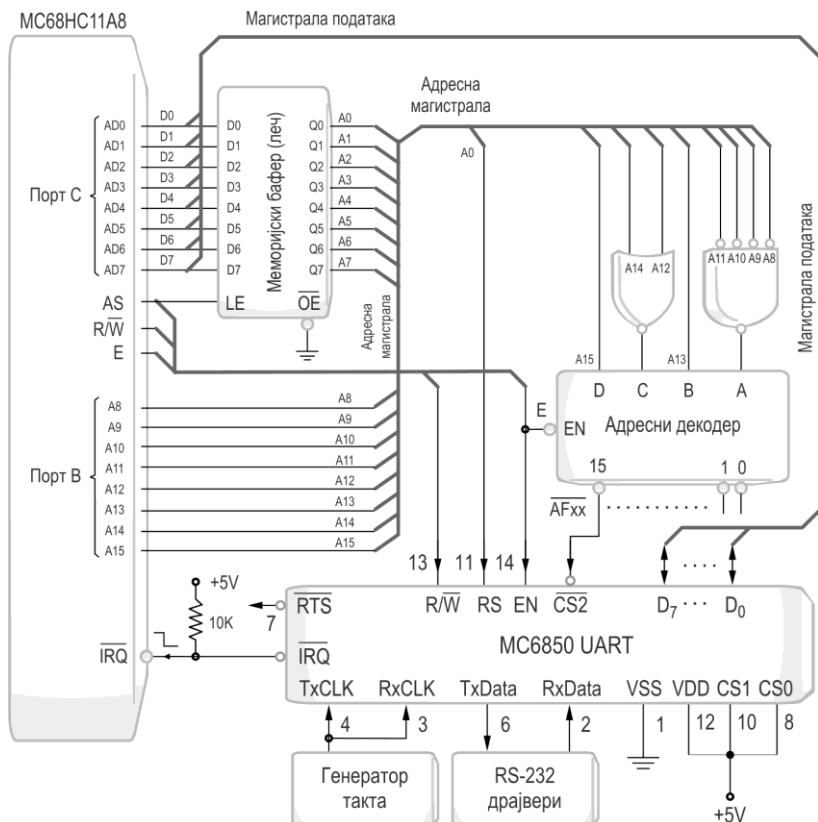
Вредност 1 бита за омогућавање прекида при пријему (CR7) омогућава пријемни прекид. Линија $\overline{IRQ}$ асинхроног примопредајника пашће на низак ниво кад год статусни бит RDRF пређе на висок ниво, указујући да асинхрони примопредајник има реч податка за процесор. Излаз $\overline{IRQ}$ се може користити за прекидање процесора кад год је статусни бит RDRF=1.

Уколико су омогућени и предајни и пријемни прекиди (CR6=0, CR5=1, CR7=1), а $\overline{IRQ}$ линија асинхроног примопредајника везана за прекидну линију процесора онда ће процесор читати статусни регистар у оквиру програма за обраду прекида да би одредио да ли су статусни битови TDRE=1 или RDRF=1 изазвали прекид. Уколико ниједан прекид није омогућен, или уколико $\overline{IRQ}$ није повезан са процесором, асинхрони примопредајник не може да прекида процесор. Тада процесор може да комуницира са асинхроним примопредајником користећи метод условног преноса.

8.3.3 Повезивање асинхроног примопредајника MC6850 и процесора

На слици 8.17 приказан је један од начина за повезивање асинхроног серијског примопредајника (MC6850) и процесора у комуникацији са серијским улазно/излазним уређајем (нпр. модем, миш, штампач...). Могућа је комуникација (начин рада) са механизмом прекида или условним преносом. Уколико се не користи комуникација са механизмом прекида $\overline{IRQ}$ веза се елиминише.

8. Везивање улазно/излазних уређаја



Слика 8.17 Веза асинхроног серијског примопредајника MC6850 са процесором (микроконтролером 68HC11)

Асинхроним примопредајнику (слика 8.17) декодујуће коло је доделило адресу \$AFxx. Пошто се улази CS0 и CS1 константно држе на високом нивоу било која адреса из опсега адреса \$AF00-\$AFFF произвешће низак ниво на CS2 улазу, и селекувати асинхрони примопредајник. Уочимо да је улаз RS везан за адресну линију A₆. Користећи адресе \$AF00 и \$AF01 процесор може да селекује асинхрони примопредајник и управља RS улазом.

Такт сигнал учестаности 153,6kHz генерише се споља и доводи на улазе TxCLK и RxCLK. Ови сигнали могу се добити из кристалног осцилатора и делитеља учестаности.

Постоје четири основне операције које процесор може да извршава са асинхроним примопредајником:

- уписивање у управљачки регистар (инструкцијом STAA \$AF00),
- читавање статусног регистра (инструкцијом LDAA \$AF00),
- уписивање у регистар TxDR (инструкцијом STAA \$AF01), и
- читавање регистра RxDR (инструкцијом LDAA \$AF01).

Уочимо да се адреса \$AF00 користи и за управљачки и за статусни регистар. Ово није проблем пошто $R/\bar{W}$ линије одређују коме се приступа. Слично, адреса \$AF01 користи се и за регистар TxDR и за регистар RxDR, али $R/\bar{W}$ линија одређује који је од ова два регистра селектован.

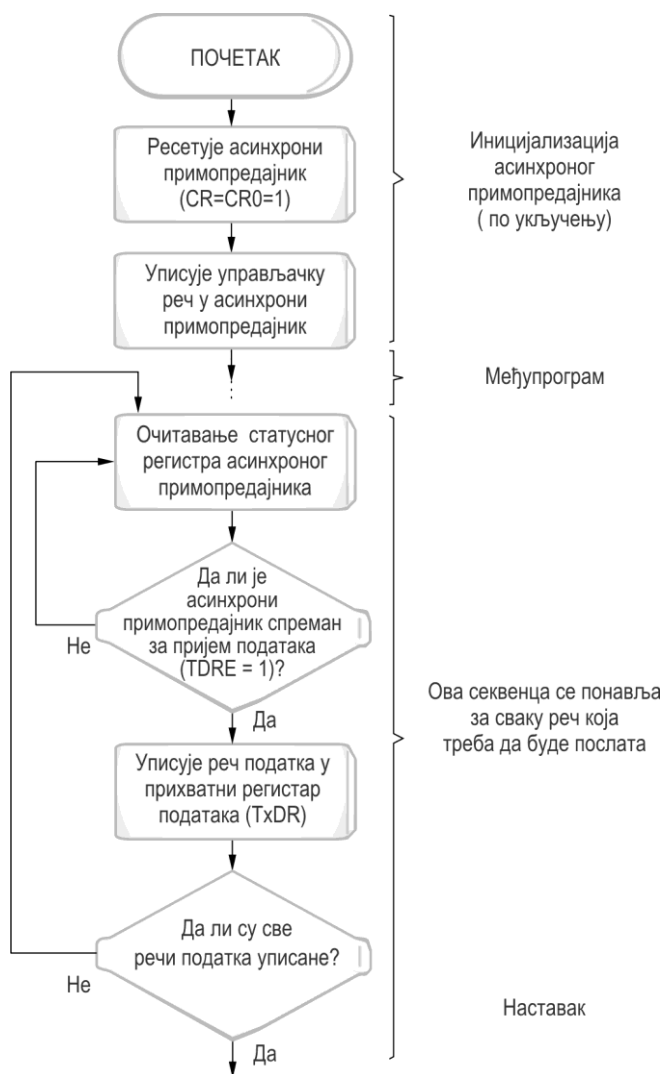
Предаја података из процесора

Претходно дефинисане инструкције користе се кад год процесор комуницира са серијским улазно/излазним уређајем преко асинхроног серијског примопредајника. Испитајмо скуп активности (корака) које процесор треба да изведе када шаље једну реч податка или блок података ка серијском излазном уређају. Ове активности су представљене дијаграмом тока који је приказан на слици 8.17.

Пре него што се било каква комуникација обави, процесор треба да иницијализује асинхрони примопредајник слањем одговарајуће (управљачке) речи у управљачки регистар. На тај начин је асинхрони примопредајник подешен на одговарајућу брзину преноса, број битова података, парност и број битова СТОП елемента.

Када процесор има податке које треба да прочита, или пошаље ка серијском уређају, он прво читава статус регистра асинхроног примопредајника и тестира бит SR1 да би одредио стање TDRE флага. Ако је TDRE=0 асинхрони примопредајник није спреман да прими податке и процесор ће поново прочитати статус регистра. Уколико је тада TDRE=1, асинхрони примопредајник је спреман да прими нову реч податка. Процесор уписује реч податка у прихватни регистар података (TxDR) и она ће затим бити послата серијски ка излазном уређају. Овај скуп активности понавља се све док процесор не пошаље (упише) све податке излазном уређају.

8. Везивање улазно/излазних уређаја

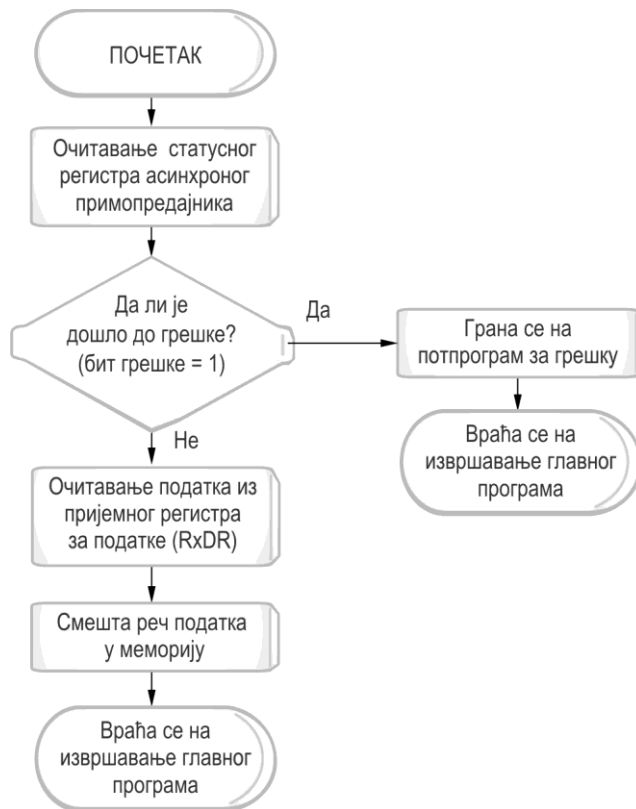


Слика 8.18 Дијаграм тока активности процесора у преносу података ка серијском излазном уређају

Пријем података у процесор

У овом делу анализираћемо активности који су потребне за пријем података у процесор из серијског улазног уређаја. За ефикасно коришћење

времена процесора погодније је користити рад са прекидом ($\overline{\text{IRQ}}$). Уколико се прекид користи код пријема података у асинхроним примопредајнику мора да, за време процеса иницијализације, бит RE буде постављен на 1 (прекид је омогућен са CR7=1). Ако асинхрони примопредајник има спремну реч податка (RDRF=1), његов излаз $\overline{\text{IRQ}}$ прелази на низак ниво, прекидајући процесор. На слици 8.19 представљен је дијаграм тока за типични потпрограма који обрађује прекид (ISR) који ће процесор извршавати када асинхрони примопредајник активира $\overline{\text{IRQ}}$ сигнал.



Слика 8.19 Дијаграм тока када процесор обрађује програм за прекид за везу са асинхроним примопредајником¹

¹ Претпоставка је да је асинхрони примопредајник био претходно иницијализован тако да је пријемни прекид омогућен и предајни прекид онемогућен.

У ствари процесор читава статус регистар и тестира битове SR6, SR5 и SR4 који указују на грешке. Уколико постоји грешка процесор прелази на извршавање одговарајућег потпрограма. Затим се враћа на програм који се извршавао пре појаве прекида. Када асинхрони примопредајник има нову реч податка овај процес се понавља.

Треба нагласити да рад с прекидом није неопходан уколико процесор нема друге активности док чека на податке из асинхроног примопредајника. С друге стране, рад с прекидом је боље решење уколико процесор има друге активности које може да извршава док чека на податке. Ово је нарочито значајно када подаци долазе из спорог улазног уређаја.

8.3.4 Комуникациони софтвер

Видели смо какав је интерфејс асинхроног примопредајника и процесора (слика 8.17), и дијаграме тока програма (слике 8.18 и 8.19) који је потребан за контролу серијске комуникације између процесора и улазно/излазног уређаја преко асинхроног примопредајника. Сада ћемо са мало више детаља анализирати потребан комуникациони софтвер. Испитаћемо уобичајене примере програма за иницијализацију, потпрограма за предају података и пријем података и програма за обраду прекида.

Програм за иницијализацију

Програм за иницијализацију треба да садржи инструкције којима се у асинхроном примопредајнику поставља жељени формат података, брзина и рад са (и без) прекида. Овај скуп (секвенца) инструкција биће део процеса иницијализације који се извршава, само једном, после укључивања напајања процесора.

Програм за иницијализацију

INIT	LDAA #\$03	; У акумулатор А уписује се ресет кôд
	STAA \$BF00	; Ресет кôд се уписује у управљачки регистар
		; асинхроног примопредајника
	LDAA #\$85	; У акумулатор А уписује се управљачка реч за MC6850
	STAA \$BF00	; Управљачка реч се уписује у управљачки
		; регистар MC6850

Пример 8.1 *Описати функције које извршава скуп инструкција (секвенца) за иницијализацију.*

Решење:

Ова секвенца инструкција извршава две функције. Ресетује асинхрони серијски примопредајник и поставља одговарајуће податке у његов управљачки регистар. Функција ресета постиже се уписивањем вредности 1 на место битова CR1 и CR0 у управљачком регистру (табела 8.1). Прве две инструкције извршавају ову операцију.

Трећа и четврта инструкција уписују управљачку реч у одговарајући регистар асинхроног серијског примопредајника. Управљачка реч је $85_{16} = 10000101_2$. Тако да имамо:

CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0
1	0	0	0	0	1	0	1

Из табеле 8.1 видимо да када су CR1=0, CR0=1 онда је однос дељење такт сигнала 16. Уколико користимо TxCLK=RxCLK=153,6kHz и као што је приказано на слици 8.14, брзина постаје 153600/16=9600b/s.

Из табеле 8.2 видимо да када су вредности CR4=CR3=0, CR2=1 онда је формат рама: 7 битова података, бит за контролу парности (непаран број јединица) и два СТОП бита.

Табела 8.3 показује да када су вредности CR6=CR5=0 онда је $\overline{\text{RTS}}$ излаз на ниском нивоу, и предајни прекид је онемогућен. Због тога асинхрони примопредајник неће генерисати прекидни сигнал на $\overline{\text{IRQ}}$ излазу када је предајни регистар података празан (TDRE=1). Када је CR7=1 пријемни прекид је омогућен. Асинхрони примопредајник ће поставити свој $\overline{\text{IRQ}}$ излаз на низак ниво кад год је пријемни регистар пун (RDRF=1), указујући да има реч податка за процесор. Низак ниво на улазу прекида рад процесора уколико бит I (у регистру услова за маскирање прекида) није претходно био постављен на вредност 1.

Потпрограма за пренос података

Потпрограма који следи може се користити за слање једне речи податка преко асинхроног примопредајника ка серијском излазном уређају. Претпоставка је да је реч податка у акумулатору А. Потпрограма користи условни улазно/излазни пренос, као што је приказано на слици. 8.18. Процесор

8. Везивање улазно/излазних уређаја

непрекидно тестира вредност TDRE бита из статусног регистра асинхроног примопредајника. Пошто потпрограм користи акумулатор А његов садржај се памти на стеку на почетку потпрограма и обнавља на крају потпрограма.

Потпрограм за предају података

PRED	PSHA	; Садржај акумулатора А на стек меморију
TEST	LDAA \$BF00	; Учитај у акумулатор А статус регистар асинхроног примопредајника
	ANDA #\$02	; Изолије TDRE бит маскирањем 2. бита
	BEQ TEST	; Ако је TDRE=0 гранај се на лабелу
		; TEST и провери поново TDRE бит
	STAB \$BF01	; Ако је TDRE=1 пошаљи податке регистру TxDR
		; асинхроног примопредајника
	PULA	; Обнови садржај акумулатора А
POV	RTS	; Врати се на главни програм

Пример 8.2 *ASCII кодови карактера речи "ZDRAVO": \$5A, \$44, \$52, \$41, \$56, \$4F смештени су на меморијске локације \$0051-\$0056, обрнутим редом. Користећи претходни описани потпрограм написати програм који шаље ову кодирану реч ка асинхроном примопредајнику за који је везан серијски уређај. Претпоставити да је асинхрони примопредајник већ иницијализован.*

Решење:

Програм који следи користи индексирано адресирање за слање ASCII кодираних података ка асинхроном примопредајнику.

Потпрограм за пренос података

START	LDX # \$0006	; Пуни се индекс регистар бројем \$0005, [X] = \$0006.
POD	LDAB \$50, X	; Пуни се акумулатор В подацима са адресе [X] + \$50.
	JSR \$0E00	; Скок на потпрограм за пренос података
	DEX	; Декрементирање X регистра.
	BNE POD	; Ако садржај индекс регистра није 0, врати се на POD.
KRAJ	WAI	; Ако је 0, пренос је завршен.

Потпрограм за обраду прекида код пријема података

Потпрограм за обраду (опслуживање) прекида (ISR¹) је секвенца инструкција коју процесор извршава као одговор на активирање $\overline{\text{IRQ}}$ сигнала из асинхроног примопредајника. Подсетимо се да је бит за омогућавање прекида пријемника у асинхроном примопредајнику постављен на вредност 1 за време извршавања програма за иницијализацију.

Потпрограм за обраду прекида пријемника²

ISR	LDA	\$BF00	; Учитавање садржаја статусног регистра асинхроног
			; примопредајника у акумулатор А
	ANDA	#\$70	; Маска за статусни бит грешке
	BEQ	POD	; Ако је резултат 0 (нема грешке) гранај
			; се на POD (пребацавање података)
	JSR	\$0FC0	; Ако није позови потпрограм за грешку на адреси \$0FC0
			; који утврђује до какве грешке је дошло
POV1	RTI		; Повратак у главни програм.
POD	LDA	\$BF01	; Пребацавање податка из пријемног регистра
			; асинхроног примопредајника у акумулатор А
	STAA	\$0200	; Сачувај податак у меморији због касније обраде
POV2	RTI		; Повратак у главни програм

8.4 СИНХРОНА СЕРИЈСКА КОМУНИКАЦИЈА

Асинхрона серијска комуникација је врста серијске комуникације која се често користи у микрорачунарским системима. Као што смо видели, асинхрони формат користи СТАРТ бит и један или више битова СТОП елемента за раздвајање појединих речи или карактера који се шаљу. Пријемник се сам синхронизише на ивицу СТАРТ бита који претходи сваком карактеру.

¹ Interrupt Service Routine

² Потпрограм је направљен по дијаграму тока са слике 8.18. Не садржи специфичне инструкције за чување садржаја регистра микропроцесора. Подсетимо се, на основу анализе рада са прекидима, да ће процесор аутоматски сачувати и обновити садржаје свих регистра као одговор на прекид.

Синхрони пренос је значајно ефикаснији начин комуникације. У овом начину рада блокови битова шаљу се без СТАРТ и СТОП елемената. Могуће је предвидети тачно време које је потребно да стигне бит. Да би се спречио временски померај између предајника и пријемника њихов такт мора да буде међусобно синхронисан. Једна могућност је да се обезбеди посебна линија за такт између предајника и пријемника. Други начин је да такт сигнал буде убачен¹ у сигнал података. Код дигиталних сигнала синхронизација се може постићи специјалним врстама кодирања². Када се преносе аналогни сигнали могу се користити бројне технике: једна од њих је синхронизација пријемника на основу фазе сигнала носеће учестаности.

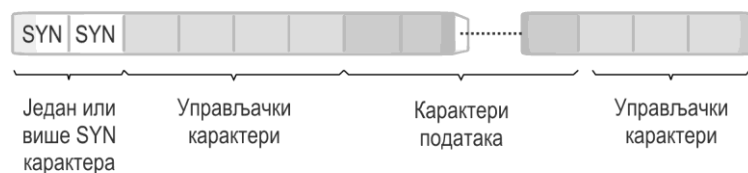
Са синхроним преносом (слика 8.20) захтева се и други ниво синхронизације који обезбеђује пријемнику да одреди почетак и крај блока података. Да би се то обезбедило сваки блок започиње преамбулом (битска комбинација која означава почетак) и завршава се постамбулом (битском комбинацијом која означава крај). Ова комбинација битова је управљачка информација, а не подаци. Подаци, заједно са управљачком информацијом називају се рам. Тачан изглед рама зависи од тога да ли је трансмисиона шема карактер оријентисана или бит оријентисана.

Код карактер оријентисаног преноса блок података се третира као секвенца карактера (обично 10-битна). Све управљачке информације су у форми карактера. Рам започиње са једним или са више синхронизационих карактера. Синхронизациони карактер (обично означен са SYN) је јединствена комбинација битова која сигнализира пријемнику да је то почетак блока. Постамбула је други јединствен карактер који се користи у истој шеми. Пријемник се поставља на блок података преко SYN карактера и прима податке докле год се не детектује карактер постамбуле. Пријемник даље може да очекује нову SYN комбинацију битова. Други приступ био би да се укључи дужина рама као део управљачких информација. Пријемник затим гледа у SYN карактер, одређује дужину рама, читава означени број карактера, гледа када ће да се појави нови SYN карактер да би започео пријем новог рама.

¹ *Embedded*

² Манчестер (или диференцијалним Манчестер) кодирањем. Манчестер кодирање се користи код локалних рачунарских мрежа Етернет типа

а) Рам код карактер оријентисаних протокола



б) Рам код бит оријентисаних протокола



Слика 8.20 Синхрони пренос

Са бит оријентисаним преносом блок података се третира као секвенца битова. Ни податке, ни управљачке информације није потребно интерпретирати као 10-битне јединице, тј. карактере. Као и са карактер оријентисаном шемом специјална комбинација битова сигнализира почетак блока. У бит оријентисаном преносу преамбула је 8 битова дугачка и назива се ознака о почетку рама¹.

Иста ознака (флег) користи се за крај рама (постамбула). Да би сигнализирао почетак рама пријемник чека појављивање комбинације битова која означава почетак. Ово је праћено одређеним бројем управљачких поља, затим пољем података променљиве дужине, па са више управљачких поља и на крају пољем које означава крај.

За велике блокове података синхрони пренос је ефикаснији него асинхрони. У синхронном преносу поље за управљачке информације је обично мање од 100 битова. На пример, једна од често коришћених шема бит оријентисаних протокола је HDLC², која садржи 48 битова управљачких информација заједно са ознакама за почетак и крај рама. За поруку од 1000 битова губитак је само $48/1000 = 4,8\%$.

¹ Застава (*flag*)

² *High-Level Data Link Control*

8.5 СТАНДАРД ЗА СЕРИЈСКУ КОМУНИКАЦИЈУ EIA-232-F

Већина дигиталних модула има ограничену способност преноса. Типично, они генеришу једноставне дигиталне сигнале, а растојање преко кога могу да преносе податке је ограничено. Таква опрема се ретко везује директно на преносни систем.

Уобичајена ситуација приказана је на слици 8.21. Опрема коју анализирамо укључује терминале и рачунаре и означава се са DTE¹. Она користи преносни систем посредством DCE². Као пример последњег је модем. На једној страни DCE је одговоран за предавање и примање битова са преносног медија. На другој страни DCE мора да буде у вези са DTE. Да би се то остварило потребан је пренос и података и управљачких информација преко скупа линија које се називају кола за размену³. Два DCE морају да разумеју један другог, што значи да пријемник једног DCE-а мора да користи исту шему кодирања (на пример Манчестер или PSK-Pulse Code Modulation) као предајник другог DCE-а. Поред тога сваки пар DTE-DCE мора да буде пројектован тако да ефикасно обави интеракцију. Да би се поједноставио овај процес развијени су стандарди који специфицирају тачну природу интерфејса DTE-DCE. Ови стандарди су познати и као протоколи физичког слоја и обухватају први слој OSI⁴ референтног модела.

У овом делу анализираћемо природу и функционалност овог интерфејса. Интерфејс има четири битне карактеристике:

- механичке,
- електричне,
- функционалне,
- процедуралне.

Механичке карактеристике односе се на стварну физичку везу DTE-DCE. Типично, сигнал и управљање воде се каблом који се завршава

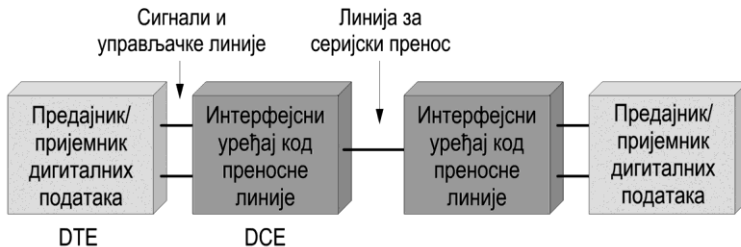
¹ *Data Terminal Equipment* - опрема терминала података

² *Data Circuit Termination Equipment* - опрема за завршетак кола података

³ *Interchange Circuit*

⁴ OSI (*Open System Interconnection*) је стандард који је донела организација ISO (*International Standard Organization*). Циљ је коришћење јединственог седмослојног модела протокола који би омогућио међусобни рад различитих мрежних уређаја и софтвера. Примена OSI модела потиснута је применом TCP/IP протокола.

а) Општи модели интерфејса ка трансмисионом медијуму



б) Уобичајена конфигурација



Слика 8.21 Интерфејс ка трансмисионом медијуму

утичницама¹. На сваком крају се DTE и DCE морају физички повезати преко каблова и утичница супротног типа. Електричне карактеристике односе се на напонске нивое и тренутке времена у којима долази до промене напона. И DTE и DCE морају користити исти код (нпр. NRZI²), морају користити исте напонске нивое и морају користити исто трајање сигнализационих елемената. Ове карактеристике одређују брзину преноса података и растојање које се може остварити.

Функционалне карактеристике специфицирају функције које се извршавају доделом значења колима за међусобну размену сигнала. Функције се могу класификовати у следеће категорије:

- подаци,
- управљање,
- временски распоред и
- уземљење.

¹ Конектори, тзв. мушки и женски.

² NRZI (*Non Return To Zero Inverted*) – без повратка на нулу инвертовано. Различите врсте кодова (комбинација напонских нивоа који представљају логичке нуле и јединице) имају за циљ обликовање спектра дигиталног сигнала.

8. Везивање улазно/излазних уређаја

Процедуралне карактеристике дефинишу редослед догађаја за слање података заснован на функционалним карактеристикама интерфејса. Постоји велики број стандарда који се односе на интерфејсе. У овом поглављу описаћемо један од њих: V.24/EIA-232-F.

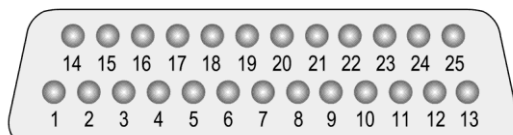
8.5.1 Стандард V.24/EIA-232-F

Највише коришћени стандард за интерфејсе је ITU-T стандард V.24. У ствари стандард само специфицира функционални и процедурални део интерфејса. Стандард V.24 се референцира на друге стандарде када су у питању електрични и механички део интерфејса. У САД постоји спецификација EIA-232-F која одговара V.24 стандарду. Стандарди на које се V.24 ослања су следећи:

- механички - ISO2110,
- електрични -V.28,
- функционални -V.24,
- процедурални -V.24.

Прву верзију спецификације EIA-232 је 1962. године издала организација EIA¹ под ознаком RS-232. Најновија верзија чија је ознака EIA-232-F појавила се 1997. године. Текуће верзије стандарда V.24 и V.28 појавиле су се 1998. и 1993. године. Овај интерфејс користи се за прикључивање DTE опреме за модеме телефонских канала². Такође се користи и за друге врсте међусобног повезивања.

Механичка спецификација EIA-232-F је представљена на слици 8.22. То је конектор са 25 прикључака³ дефинисан у стандарду ISO2110.



Слика 8.22 V.24 / EIA-232 конектор

¹ Electronic Industries Alliance

² Када се каже телефонски канал, или говорни канал, мисли се на канал за говор који се користи у јавним телефонским мрежама.

³ 25-пински

Овај конектор је прикључак 25-жичног кабла који се може користити да би се повезали DTE (терминал) и DCE (нпр. модем). У пракси се користи кабл са мање жица.

Електричне карактеристике означавају сигнализацију између DTE и DCE опреме. У свим колима за размену користи се дигитална сигнализација. Стандард је да се напон негативнији од -3V интерпретира као бинарно 1, а напон позитивнији од +3V интерпретира као бинарно 0. Брзина на интерфејсу мања је од 20kb/s и раздаљина од 15m. Исти напонски нивои користе се за управљачке сигнале; напон негативнији од -3V интерпретира се као искључење, а напон позитивнији од +3V интерпретира се као укључење.

Функционална спецификација најважнијих кола за размену дата је у табели 8.5. Кола се могу груписати у следеће категорије: подаци, управљање, временски редослед и уземљење.

V24	EIA-232	Име	Усмерен ка	
Сигнали података				
103	BA	Слање података <i>Transmitted data</i>	DCE	Шаље DTE
104	BB	Пријем података <i>Received data</i>	DTE	Прима DTE
118	SBA	Секундарно слање података <i>Secondary transmitted data</i>	DCE	Шаље DTE
119	SBB	Секундарни пријем података <i>Secondary received data</i>	DTE	Прима DTE
Управљачки сигнали				
105	CA	Захтев за слањем <i>Request to send</i>	DCE	DTE жели да шаље
106	CB	Спреман за слање <i>Clear to send</i>	DTE	DCE је спреман да прима, одговара на RTS
107	CC	DCE је спреман <i>DCE Ready</i>	DTE	DCE је спреман за рад
108.2	CD	DTE је спреман <i>DTE Ready</i>	DCE	DTE је спреман за рад
125	CE	Индикатор звоњења <i>Ring Indicator</i>	DTE	DCE прима сигнал звоњења

8. Везивање улазно/излазних уређаја

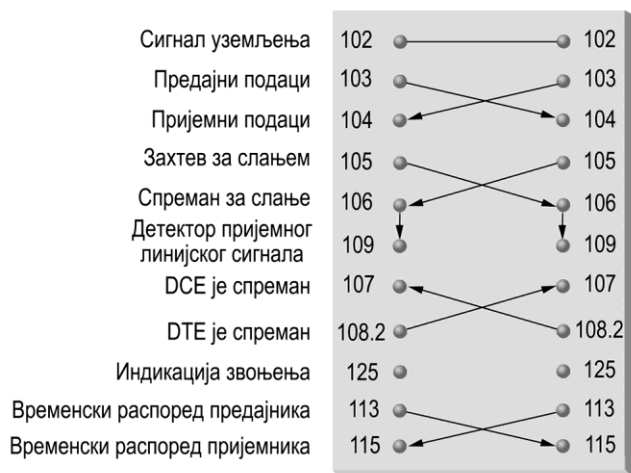
V24	EIA-232	Име	Усмерен ка	
109	CF	Детектор сигнала на пријемној линији <i>Received Line Signal Detector</i>	DTE	DCE прима сигнал са одговарајућим ограничењима у каналу
110	CG	Детектор квалитета сигнала <i>Signal quality detector</i>	DTE	Указује да постоји велика вероватноћа грешке у примљеним подацима
111	CH	Селектор брзине података <i>Data Signal Rate Selector</i>	DCE	Селекује једну од две брзине података
112	CI	Селектор брзине података <i>Data Signal Rate selector</i>	DTE	Селекује једну од две брзине података
133	CJ	Спреман за спање <i>Ready for receiving</i>	DCE	Укључена/искључена контрола тока
120	SCA	Секундарни захтев за спањем <i>Secondary request to send</i>	DCE	DTE жели да шаље преко повратног канала
121	SCB	Секундарни спреман за спање <i>Secondary clear to send</i>	DTE	DCE жели да прима преко повратног канала
122	SCF	Детектор сигнала на секундарној пријемној линији <i>Secondary received line signal detector</i>	DTE	Исто као 109, за повратни канал
140	RL	Удаљена затворена петља <i>Remote loopback</i>	DCE	Указује удаљеном DCE да направи затворену петљу
141	LL	Локална затворена петља <i>Local loopback</i>	DCE	Указује DCE-у да направи затворену петљу за сигнал
142	TM	Врста рада - тестирање <i>Test mode</i>	DTE	Локални DCE је у стању тестирања
Временски сигнали				
113	DA	Временски распоред предајних сигнализационих елемената <i>Transmitter signal element timing</i>	DCE	Такт сигнал; транзиција на укључено (ON) и искључено (OFF) јавља се у средини сваког сигнализационог елемента
114	DB	Временски распоред предајних сигнализационих елемената <i>Transmit signal element timing</i>	DTE	Такт сигнал; и 113 и 114 односе се на сигнале у колу 103

V24	EIA-232	Име	Усмерен ка	
115	DD	Временски распоред предајних сигнализационих елемената <i>Receiver signal element timing</i>	DTE	Такт сигнал за коло 104
Уземљење				
102	AV	Уземљење / заједничка маса <i>Signal ground/common return</i>	DTE, DCE	Заједничко уземљење се односи на сва кола

Табела 8.5 Кола за размену V.24 / EIA-232 - F

Спецификација процедура дефинише секвенце догађаја у којима се користе различита кола за одређену намену.

Потребно би било размотрити ситуацију у којој је раздаљина између два DTE толико мала да могу директно да комуницирају. У том случају кола за размену V.24/EIA-232-F могу се и даље користити, али не постоји потреба за DCE опремом. Да би такав систем могао да ради потребан је нул модем¹ који повезује линије на такав начин да је сваком DTE-у представљено као да је везан на модем (слика 8.23).



Слика 8.23 Пример нул модема

¹ Null Modem

8.6 ПАРАЛЕЛНО/ПАРАЛЕЛНИ ИНТЕРФЕЈС

Асинхроно серијски примопредајник MC6850 је сложено интерфејсно интегрисано коло које омогућава комуникацију уређаја који раде са паралелним подацима (процесор) са уређајима који раде са серијским подацима. Постоје и бројни уређаји који раде са паралелним подацима као што су: А/Д, Д/А конвертори, тастатуре, неки штампачи и велики број инструмената за мерења. Иако се ови уређаји могу прикључити преко једноставних улазних и излазних портова, произвођачи су развили специјална интегрисана кола - паралелне интерфејсе са већим могућностима од обичног улазно/излазног порта. Неке од ових карактеристика су:

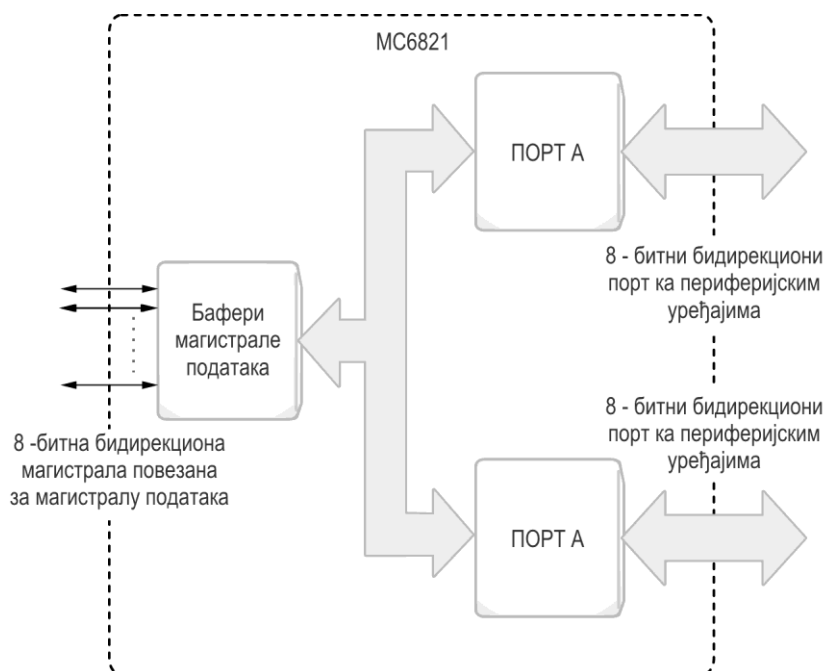
1. Програмабилност. Портови, и у многим случајевима поједини битови сваког порта, могу се конфигурисати било као улази или излази процесора. Ова функција обезбеђује се уписивањем у управљачки регистар процесора.
2. Размена сигнала. Неколико управљачких сигнала може се добити како би се управљачке информације пренеле између процесора и периферије.
3. Управљање прекидима. Ова функција омогућава процесору да управља периферијским уређајима користећи прекиде.
4. Часовници¹. Нека паралелна интерфејсна интегрисана кола укључују часовнике у самом интегрисаном колу. Ови часовници раде са системским тактом сигналом и процесор их може испрограмирати тако да раде на различите начине.

8.6.1 Периферијски интерфејсни адаптер MC6821

Периферијски интерфејсни адаптер MC6821 (PIA²) је паралелно/паралелно интегрисано интерфејсно коло које обезбеђује многе од горепомнутих карактеристика. На слици 8.24 приказан је поједностављен блок дијаграм интерфејсног кола MC6821. Види се да уређај комуницира са процесором преко 8-битне двосмерне магистрале података која је повезана са магистралом података процесора.

¹ Timer

² Parallel Interface Adapter



Слика 8.24 Поједностављени дијаграм интерфејсног адаптера MC6821

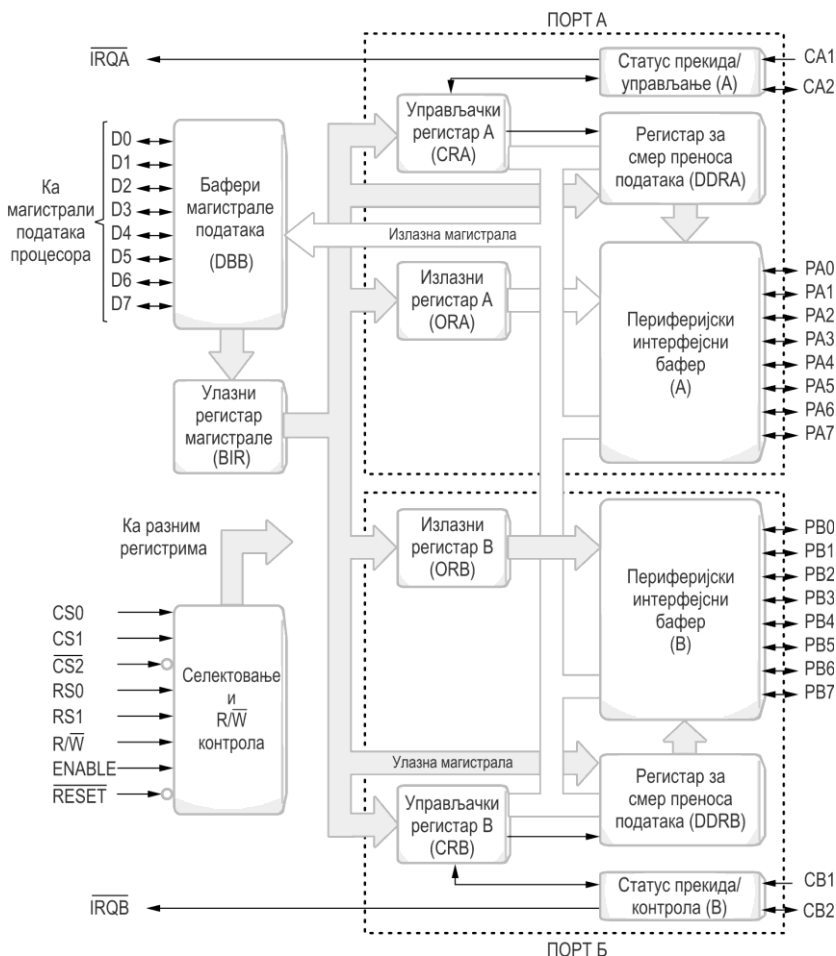
Паралелним периферијским уређајима су на располагању два потпуно независна двосмерна порта. У овом једноставном примеру процесор може да шаље информације ка портovima А и В операцијом уписивања у специфицирану адресу порта. Процесор може да прима информације из периферијског уређаја преко портova А или В операцијом читавања са специфициране адресе порта.

Да би се добио потпун увид у могућности и флексибилност које MC6821 кола пружа пројектантима микрорачунарског систем потребна је детаљнија анализа овог интерфејсног адаптера.

На слици 8.25 приказан је детаљни блок дијаграм интерфејсног адаптера MC6821 који се може посматрати као два комплетна, независна У/И порта. Портovi А и В садрже регистре за одређивање смера преноса¹ (DDRA и DDRB) који омогућавају програмеру да специфицира независно

¹ Data Direction Register

8. Везивање улазно/излазних уређаја



Слика 8.25 Коришћења паралелног интерфејсног адаптера MC6821 за повезивање са 16 тастера

поједине прикључке (пинове¹) портова А и В било као улазне било као излазне. Постављањем на вредност 0 одговарајућих бита регистра за одређивање смера преноса (DDR), одговарајући прикључци порта се понашају као улазни а постављањем на вредност 1 као излазни.

¹ Pin (контакт)

Управљачки регистри¹ (CRA и CRB) омогућавају програмеру да одабере врсту прекида и начин управљања периферијским уређајима. Такође, управљачки регистар обезбеђује статусне информације које су везане за рад са прекидима.

Излазни регистар² (ORA и ORB) чува податке који се шаљу ка излазном уређају док тај уређај није спреман да их прими. Улази CA1, CA2, CB1 и CB2 могу се употребити као статусни улази за условни улазно/излазни пренос или интерфејс за рад са прекидима. Користећи управљачки регистар може се излазни регистар програмирати тако да се улази CA2 и CB2 користе и као излази за размену управљачких сигнала.

Када ради са овим интегрисаним колом процесор треба да комуницира са шест различитих регистара. Управљачки део (слика 8.25) за селектовање интегрисаног кола (CHIP SELECT и $R/\overline{W}$) има само два улаза за селектовање регистара RS0 и RS1. На први поглед изгледа као да процесор може да комуницира само са четири регистра. У ствари један бит управљачког регистра се користи да би се направила разлика између адресирања регистра за одређивање смера преноса и излазног регистра. Ово даје могућност за комуникацију са шест различитих регистара: два регистра за одређивање смера преноса (DDRA и DDRB), два управљачка (CRA и CRB) и два излазна регистра (ORA и ORB). Улази за селектовање регистра³ (RS) и селектовање интегрисаног кола⁴ (CS) користе се на исти начин као код асинхроног примопредајника MC6850.

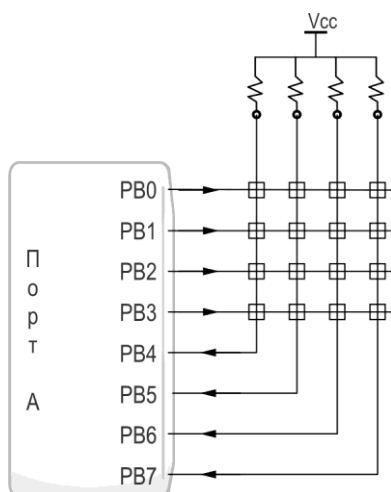
Претпоставимо да је у систему потребна тастатура са 16 тастера. Најједноставнији начин реализације је повезивње једног тастера на једну улазну линију (слика 8.26). Оваквим приступом се у значајној мери расипају хардверски ресурси микроконтролера на који се оваква тастатура повезује. Примера ради код микроконтролера MC68HC11A8 имамо на располагању 16 улазно/излазних линија (A и B 8-битни портави). Предност оваквог начина повезивања је једноставност софтвера за читавање тастатуре. Наиме потребно је само прочитати садржај са два порта (две узастопне LDA инструкције за фамилију MC68XX).

¹ Control register

² Output register

³ Register Select

⁴ Chip Select



Слика 8.26 Повезивање тастатуре (16 тастера) на један 8-битни порт

У случају тастатура са већим бројем тастера проблем се не може решити без додавања спољних компоненти. На слици 8.26 је приказано како се тастатура са 16 тастера може повезати на само један 8-битни порт што је значајна уштеда хардверских ресурса. Цена која се плаћа је значајно сложенији софтвер за читавање тастатуре. Са слике 8.26 види се да је за читавање тастатуре потребно 4 излазне и 4 улазне линије. Процесор на све излазне линије избацује вредност логичке 0. Уколико ни један тастер није притиснут на свим улазним линијама се читава вредност логичке 1 (због отпорника везаних на улазне линије). Уколико се притисне било који од тастера нека од улазних линија добија вредност логичке 0. Међутим детекција 0 на некој од линија не одређује једнозначно који од тастера у колони је притиснут. Због тога информација о постојању логичке 0 на неком од улазних портова даје процесору информацију да је неки од тастера притиснут али не и који. Да би се тачно одредио притиснути тастер процесор мора да пређе у режим сканирања тастатуре. Процесор на излазну линију порта PB0 избацује вредност логичке 0 а на остале вредност логичке 1 и након тога чита вредност улазног порта. Уколико су на свим улазним портovima логичке јединице процес се наставља, логичка 0 се избацује на излазну линију порта PB1 и чита улазни порт. Оног тренутка

када се на некој линији улазног порта чита вредност логичке 0 прекида се процес сканирања и врши се декодовање. Тастер који је притиснут се налази у пресеку врсте на коју је избачена логичка нула и колоне у којој је логичка 0 читана. Софтверска реализација се додатно компликује уколико је потребно детектовати више истовремено притиснутих тастера. У практичним реализацијама најчешће се среће овај тип тастура без обзира на сложеност софтверске реализације.

У микрорачунарским системима за везу са корисником најчешће се користе тастатура и LCD екран са 2 или 4 реда и 16 или 20 карактера у реду (слика 8.27). Број линија улазно/излазних портова ограничен је о чему се у пројектовању изузетно води рачуна када се ове две периферије користе.

Паралелни интерфејсни адаптер MC6821 и безусловни пренос

Паралелни интерфејсни адаптер MC6821 може се користити за безусловни пренос ка (од) улазно/излазним уређајима као што је анализирано у 7. поглављу. Тада се процесор повезује са бафером података интерфејсног адаптера MC6821 преко магистрале података а портови А и/или В се повезују са улазно/излазним уређајем.

Да би се порт А користио као излазни порт процесор мора да упише све јединице у регистар за одређивање смера преноса (DDRA). Процесор шаље податке ка излазном уређају који је везан на порт А једноставном операцијом уписивања у излазни регистар А (ORA).

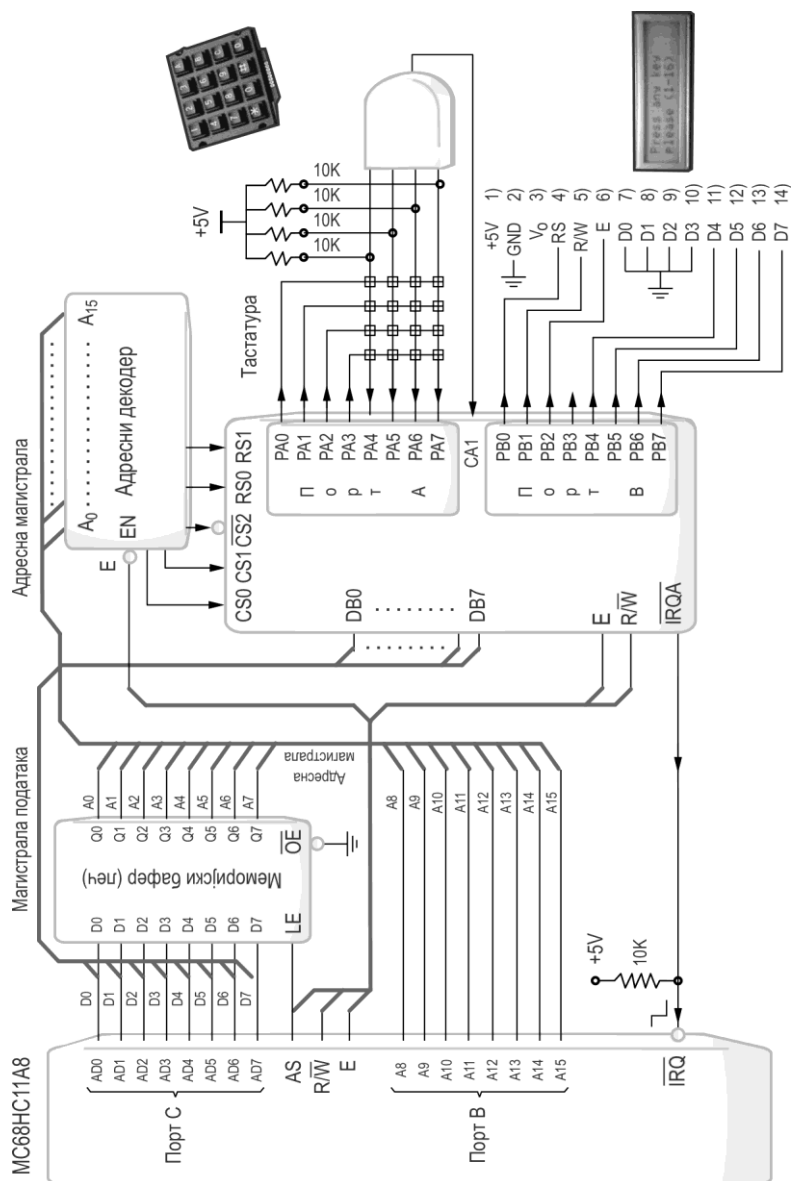
Интерфејсно коло MC6821 је могуће програмирати. Када порт А треба да се користи као улазни порт процесор уписује све нуле у регистар за одређивање смера преноса (DDRA). Тако, када процесор извршава операцију читавања са порта А, логички ниво присутни на улазима PA₀ до PA₇ интерфејсног адаптера MC6821 појавиће се на магистрали података и уписати у интерне регистре процесора.

Наравно на исти начин се може користити и порт В.

Паралелни интерфејсни адаптер MC6821 и условни пренос

Условни пренос података који процесор иницира може се једноставно реализовати коришћењем интерфејсног адаптера MC6821. Два бита¹

¹ Bit flag



Слика 8.27 Коришћења паралелног интерфејсног адаптера MC6821 за повезивање са 16 тастера

управљачког регистра користе се за статусне информације које процесор може да читава¹. Битове статусних информација постављају улази CA1, CA2 или CB1, CB2 респективно. Интерфејсни адаптер MC6821 може се програмирати тако да управљачки регистар реагује било на улазну, било на силазну промену сигнала на овим улазима. Одговарајућа промена (транзиција) на CA или CB улазима поставиће одговарајуће статусне информације у управљачким регистрима. Када се извршава условни пренос процесор може да читава статус управљачког регистра и испита битове статусних информација и на основу тога изврши одговарајуће операције. Када процесор изврши операцију читавања са одговарајућег порта, или, када се генерише RESET сигнал, битови статусних информација се бришу (постављају на вредност 0).

Пример 8.3 *Слика 8.27 показује један од начина везивања интерфејсног адаптера MC6821. Порт B се користи као излазни порт и за њега је везан 2X16 LCD екран док је горњи нибл порта A улазни порт а доњи нибл порта A излазни и за њега је везана тастатура 4X4. Описати операције које је потребно извршити да би се омогућио рад са тастатуром.*

Решење:

Излаз: Пренос података из процесора ка тастатури је безусловни пренос. Пинови PA₀ до PA₃ су програмирани као излазни, а пинови PA₄ до PA₇ су програмирани као улазни. Процесор то постиже уписивањем \$0F у регистар за одређивање смера преноса DDRA; ова операција обично се извршава на почетку програма (иницијализација). Процесор шаље 8-битну реч податка ка тастатури операцијом уписивања у излазни регистар ORA. Адресе које се користе за адресирање регистра интерфејсног адаптера одређене су адресном декодерском логиком, као што је било и у примеру повезивања синхроног примопредајника и улазно/излазних уређаја.

Улаз: Пренос података из тастатуре ка микропроцесору је безусловни пренос. Пинови PA₄ до PA₇ су програмирани као улазни. Процесор на порт A избацује податак 0, што значи да ће на доњем ниблу порта A бити све нуле. Процесор у петљи чита порт A. Уколико је вредност горњег нибла различита од 0xF0 значи да је неки од тастера

¹Read only

притиснут и прелази се у рутину за сканирање тастатуре да би се одредио тастер који је притиснут. Комплетан асемблерски програм који извршава горепоменуте операције дат је у оквиру примера 8.4

Пример 8.4 *Написати потпрограм за сканирање тастатуре приказане на слици 8.27. Потпрограм треба у акумулатору А да врати код стиснутог тастера. Код тастера се представља индексима врсте и колоне у чијем се пресеку налази притиснут тастер. У горњем ниблу је индекс врсте а у доњем индекс колоне. Индекси се представљају бројевима од 0 до 3. У случају да није притиснут ниједан тастер враћа се вредност \$FF. Претпоставимо да се порту приступа преко адресе \$1020.*

```
SKAN_TST LDAB    #$70
          STAB    $1020 ;избачена 0 на четврту врсту
          LDAB    $1020
          ANDB    #$0F
          CMPB    #$0F
          BEQ     NAS1
          JSR     DEKOD
          CMPA    #$F
          BNE     TAST_4
          LDAA    #$FF
          RTS
TAST_4   ORAA    #$30
          RTS
NAS1     LDAB    #$B0
          STAB    $1020 ; избачена 0 на трећу врсту
          LDAB    $1020
          ANDB    #$0F
          CMPB    #$0F
          BEQ     NAS2
          JSR     DEKOD
          CMPA    #$F
          BNE     TAST_3
          LDAA    #$FF
          RTS
```

TAST_3	ORAA	#\$20	
	RTS		
NAS2	LDAB	#\$D0	
	STAB	\$1020	; избачена 0 на другу врсту
	LDAB	\$1020	
	ANDB	#\$0F	
	CMPB	#\$0F	
	BEQ	NAS3	
	JSR	DEKOD	
	CMPA	#\$F	
	BNE	AST_2	
	LDAA	#\$FF	
	RTS		
TAST_2	ORAA	#\$10	
	RTS		
NAS3	LDAB	#\$E0	
	STAB	\$1020	; избачена 0 на прву врсту
	LDAB	\$1020	
	ANDB	#\$0F	
	CMPB	#\$0F	
	BEQ	NAS4	
	JSR	DEKOD	
	CMPA	#\$F	
	BNE	TAST_1	
	LDAA	#\$FF	
TAST_1	RTS		
NAS4	LDAA	#\$FF	
	RTS		; није притиснут ниједан тастер
DEKOD	CMPB	#\$0E	; провера да ли је притиснут тастер у првој колони
	BNE	ISP_1	
	LDAA	#\$0	
	RTS		
ISP_1	CMPB	#\$0D	; провера да ли је притиснут тастер у другој колони
	BNE	ISP_2	
	LDAA	#\$1	
	RTS		
ISP_2	CMPB	#\$0B	; провера да ли је притиснут тастер у трећој колони

8. Везивање улазно/излазних уређаја

	BNE	ISP_3	
	LDAA	#\$2	
	RTS		
ISP_3	CMPB	#\$07	; провера да ли је притиснут тастер у четвртој колони
	BNE	ISP_4	
	LDAA	#\$3	
	RTS		
ISP_4	LDAA	#\$F	; није притиснут ниједан тастер
	RTS		

Паралелни интерфејсни адаптер MC6821 и механизам прекида

За пренос података између интерфејсног адаптера MC6821 и улазно/излазног уређаја који користе механизам прекида, уз сваки порт (А и В) постоје одговарајући излази за прекид ($\overline{IRQA}$ и $\overline{IRQB}$). Излази за прекид су активни када су на ниском логичком нивоу и могу се везати на маскирајући улаз за прекид процесора ($\overline{IRQ}$). Излази за прекид могу се програмски дозволити (омогућити) или забранити (онемогућити) уписивањем одговарајуће речи у управљачки регистар. Када се користе излази за прекид ($\overline{IRQA}$ и $\overline{IRQB}$) онда су они активни (прелазе на низак ниво) кад се постави ознака о прекиду (флег) у одговарајући управљачки регистар. Ознака о прекиду је исти статусни бит који се користи код условног преноса. Активира се (поставља на висок ниво) одговарајућим променама на СА или СВ улазима. Улази СА и СВ понашају се као улази за прекид које улазно/излазни уређаји могу да активирају. Пример 8.5 то и илуструје.

Пример 8.5 *Описати како тастатура на слици 8.27 користи механизам прекида за пренос података ка процесору.*

Решење:

Да би се обезбедио рад интерфејсног адаптера са механизмом прекида неколико ствари се мора урадити. Прво, пошто тастатура користи порт А треба излаз $\overline{IRQA}$ везати за $\overline{IRQ}$ улаз процесора. Друго, процесор треба да омогући прекид уписивањем управљачке речи у регистар CRA. Треће, процесор треба да специфицира да негативна транзиција сигнала на улазу СА1 поставља статусни бит (флег) прекида. Тако, када се притисне

неки од тастера CA1 линија прелази са високог на ниски ниво, а када се отпусти прелази са ниског на висок напонски ниво (позитивна транзиција). Улаз CA1 поставља ознаку о прекиду у регистар CRA. Ово, даље доводи излаз $\overline{IRQA}$ интерфејсног адаптера и улаз $\overline{IRQ}$ процесора (на који је он везан) на низак ниво и прекида се рад процесора. Процесор „одговара“ на овај прекид, читавањем података са тастатуре везане на порта А интерфејсног адаптера. После читавања података брише се статусни бит прекида и доводи излаз $\overline{IRQA}$ на висок ниво.

Размена управљачких сигнала интерфејсног адаптера MC6821

Битна карактеристика интерфејсног адаптера MC6821 је могућност да се прикључци CA2 и CB2 могу испрограмирати тако да раде као излази а не као улази. Уписивањем одговарајуће речи у управљачки регистар излази CA2 и CB2 омогућавају три различита начина размене управљачких сигнала и управљања периферијским уређајима.

Код првог начина се CA2 или CB2 излази постављају на висок или низак логички ниво у зависности од логичког нивоа који је уписан у одговарајућу битску позицију управљачког регистра.

Када је примењен други начин, за комуникацију са процесором користи се механизам прекида постављајући CA1 улаз на низак ниво, а CA2 излаз на висок ниво. Када процесор чита податак са порта А онда се излаз CA2 поставља на низак ниво, указујући периферијском уређају да је податак учитан и нови податак може да се пошаље. За рад са портом В, CB2 излаз се поставља на низак ниво када процесор извршава операцију уписивања ка порту В. Пошто излазни уређај прими ову информацију он активира улаз CB1 чиме се CB2 поставља на висок ниво и активира излаз $\overline{IRQB}$ указујући процесору да је информација примљена. Тако је процесор „обавештен“ да је излазни уређај примио информацију. Види се да је порт А боље користити за комуникацију са улазним уређајима, а порт В са излазним уређајима.

Трећи начин назива се импулсни начин рада. Излаз CA2 генерише импулс сваки пут када процесор читава садржај података са порта А, а излаз CB2 генерише импулс када процесор уписује податке користећи порт В.

Као што је претходном анализом показано интерфејсни адаптер MC6821 је уређај великих могућности које дају велику слободу пројектанту

8. Везивање улазно/излазних уређаја

микрорачунарског система. Најважнија карактеристика интерфејсног адаптера MC6821 је могућност програмирања. Његове основне функције могу се, под програмском контролом, мењати чак и када је систем у раду.

8.7 ПИТАЊА И ЗАДАЦИ

1. Пројектовати неопходна кола за превођење TTL логичких нивоа (0, +5V) у нивое EIA-232-F +12V и -12V за логичко 0 и 1.
2. Зашто је потребно користити оптокаплере и релеа када се на излазни порт везује потрошач наизменичне струје?
3. Шта од понуђеног није део стандардног асинхроног серијског формата:
а) СТАРТ бит (0) б) 5-8 битова података в) бит статуса
г) један или више СТОП битова д) опциони бит парности
4. Описати процедуру која се извршава када процесор прима податке са серијског улаза користећи асинхрони серијски примопредајник (UART).
5. Набројати 4 регистра асинхроног серијског примопредајника MC6850 са којима процесор може да комуницира.
6. Информације које одређују карактеристике преноса између процесора и асинхроног серијског примопредајника MC6850 налазе се у:
а) регистру статуса б) пријемном регистру података
в) предајном регистру података г) управљачком регистру
7. Коју управљачку реч треба процесор да пошаље управљачком регистру асинхроног серијског примопредајника MC6850 да би радио на следећи начин:
а) 9600Bd користећи спољашњи такт сигнал 614,4kHz
б) 8 битова података в) парна парност
г) СТОП елемент дужине једног бита
д) $\overline{\text{RTS}}$ 0 предајни прекид онемогућен
8. Која је намена регистра за одређивање смера преноса података (DDR) код периферијског интерфејсног адаптера MC6821 (PIA)?
9. Периферијски интерфејсни адаптер MC6821 нема одређени статус регистар. Где се смештају статусне информације које процесор може да проверава?
10. Које механизме прекида користи периферијски интерфејсни адаптер MC6821?

9. Микроконтролер MC68HC11

Микроконтролер MC68HC11 спада у категорију микроконтролера са великим бројем интегрисаних периферија и широким спектром примене. Може да ради у четири различита режима:

- једно интегрисано коло¹ - када се сва програмска меморија, улазно/излазни модули и меморија за податке налазе у самом интегрисаном колу;
- проширени режим² рада - када је део меморије и улазно/излазних модула ван самог интегрисаног кола;
- бутстреп³ режим рада који се може користити за учитавање програма у RAM меморију и
- тест⁴ режим рада који користи произвођач (*Motorola*).

У оквиру MC68HC11 фамилије постоји девет важнијих серија: А, D, Е, F, G, K, L, М и Р. Свака од њих има своје специфичности везане за (не) постојање А/Д конвертора, мултиплексирање, брзину, меморију и улазно/излазне модуле. У табели 4.2 (4. поглавље) представљени су неки од најчешће коришћених представника ових серија, а на слици 9.1 представљен је комплетан блок дијаграм микроконтролера MC68HC11. У претходним поглављима ове књиге анализирали смо рад главних делова микроконтролера и његових пет портова. Ово је функционални блок дијаграм тако да има мало детаља о међусобној повезаности самих блокова. На слици 9.1 приказано је и како су ови блокови повезани за прикључке интегрисаног кола.

У централном делу слике 9.1 је процесор. У горњем левом углу су RAM, ROM и EEPROM меморије самог интегрисаног кола⁵. Логика којом се бира режим рада налази се у доњем левом углу дијаграма. Блок који

¹ Single chip

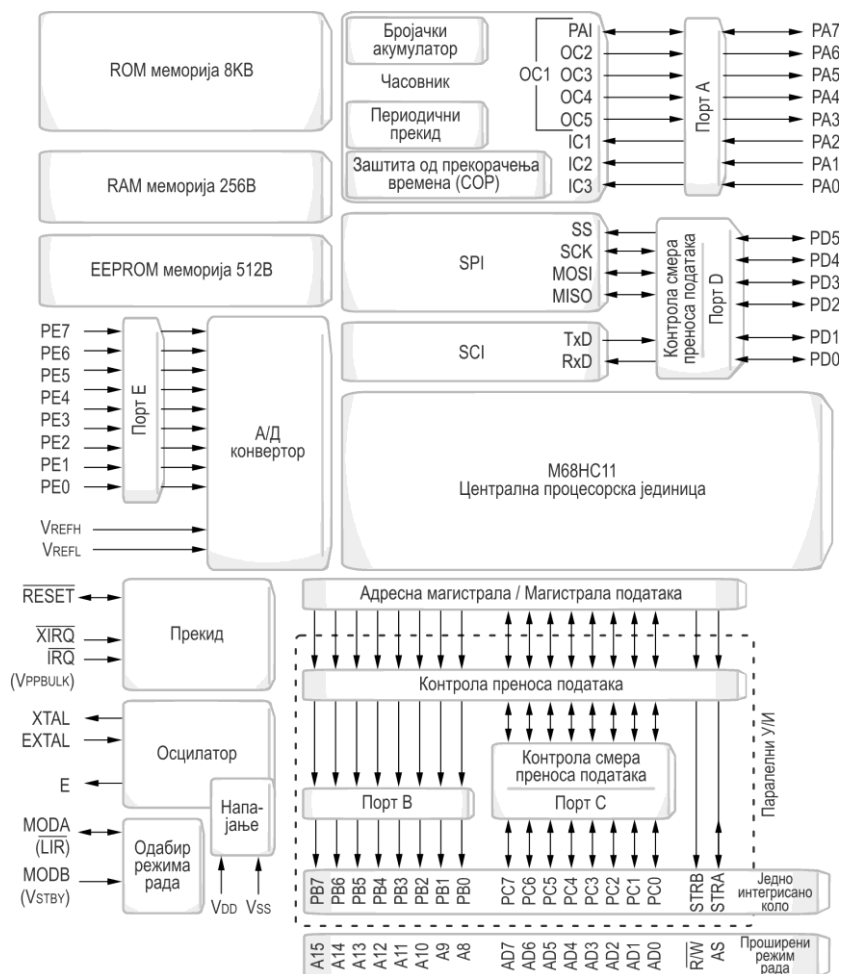
² Expanded mode

³ Bootstrap mode

⁴ Test mode

⁵ On chip RAM

9. Микроконтролер MC68HC11



Слика 9.1 Блок дијаграм микроконтролера MC68HC11

генерише Е-такт сигнал и друге временске сигнале налази се управо изнад блока за одабир режима рада. У доњем десном углу дијаграма су портови В и С са два могућа начина рада. Када микроконтролер ради у режиму једног интегрисаног кола онда они имају функцију стандардних паралелних портова: порт В је излазни а порт С је улазно/излазни порт. У проширеном режиму рада они раде као адресне линије из микроконтролера.

У горњем, централном делу дијаграма приказан је блок система часовника¹. Десно од овог блока је вишенаменски порт А који часовник понекад у свом раду користи. Испод порта А је порт D, 6-битни двосмерни (бидирекциони) порт који се користи за серијску комуникацију са спољним светом.

Лево од процесора је А/Д конвертор који се налази у оквиру самог интегрисаног кола. Порт Е је улазни порт А/Д блока и као што ћемо видети (одељак 9.5) може се користити или као аналогни улазни порт А/Д блока или као стандардни дигитални улазни порт микроконтролера.

9.1 Улазно/излазни портови микроконтролера MC68HC11

Микрорачунарски системи користе улазно/излазне уређаје за повезивање са спољним светом. Улазни уређаји омогућавају да рачунар прикупља информације из спољнег окружења, док излазни уређаји омогућавају да рачунар прикаже информације, или да утиче на окружење. Основна особина која одваја управљачке рачунарске система од обичних рачунара је тесна повезаност рачунара са спољним светом, а улазно/излазни портови су специфичне компоненте микрорачунара које им омогућавају интеракцију са окружењем. У овом одељку описани су основни улазно/излазни портови микроконтролера MC68HC11, и основе програмирања тих портова.

Код микрорачунара из серије MC68HC11, као и многих других микрорачунара, улазно/излазни портови² су меморијски мапирани³. Значи, софтвер приступа порту једноставним читавањем садржаја са одговарајуће адресе (за улазни порт), или уписивањем податка на одговарајућу адресу (за излазни порт) која се само по вредности разликује од адреса за меморију. Сваки порт, односно њему придружен регистар, има јединствену адресу у адресном простору. Да би програми били разумљивији, уобичајено је да се укључе симболичке дефиниције улазно/излазних портова, тако да се уместо физичких адреса портова користе њихова имена.

¹ *Timer system block*

² У/И портови

³ Детаљније је објашњено у 4.11 одељку овог уџбеника.

9. Микроконтролер MC68HC11

Основни модели микроконтролера из серије MC68HC11 имају 40 улазно/излазних прикључака¹. Сваки од ових прикључака може бити улазно/излазни прикључак опште намене (прикључак паралелног интерфејса), или остваривати функцију неке од периферија реализованих на интегрисаном колу.

Прикључци порта А могу се користити као улази или као излази система часовника. Када се порт А користи као паралелни порт три прикључка (РА0-РА2) дефинисана су као улази, четири прикључка (РА3-РА6) као излази, док је један прикључак (РА7) двосмерни². Смером прикључка РА7 управља бит 7 (DDRA7) управљачког регистра PACTL.

Порт В, порт С, строб А (STRA) и строб В (STRB) прикључци у проширеном начину рада³ користе се као адресне линије и као линије података (порт В и порт С) и као управљачки сигнали (STRA и STRB). Када микроконтролер ради као једно интегрисно коло⁴ ових осамнаест линија је на располагању као улазно/излазне линије опште намене. Порт В је осмобитни само излазни порт, док је порт С осмобитни бидирекциони порт. Свака од линија порта С може да буде независно дефинисана као улазна или као излазна линија уписивањем одговарајућег садржаја у регистар смера порта С (DDRC⁵).

Порт D је шестобитни бидирекциони порт. Свака од линија порта D може да буде независно дефинисана као улазна или као излазна линија уписивањем одговарајућег садржаја у регистар смера порта D (DDRD⁶). Два прикључка порта D алтернативно се користе за асинхрони серијски комуникациони интерфејс (SCI⁷), док се четири преостала прикључка алтернативно користе за синхрони периферијски интерфејс (SPI⁸).

Порт Е је осмобитни, само улазни, порт. Алтернативно се прикључци порта Е користе као улази осмоканалног А/Д⁹ конвертора.

¹ У/И пинова

² Бидирекциони

³ *Extended mode*

⁴ *Single-chip mode*

⁵ *Data Direction Control Register for port C*

⁶ *Data Direction Control Register for port D*

⁷ *Serial Communications Interface subsystem*

⁸ *Synchronous Serial Peripheral Interface subsystem*

⁹ *Analog-to-digital (A/D) converter channel inputs*

9.2 Паралелни улазно/излазни порт

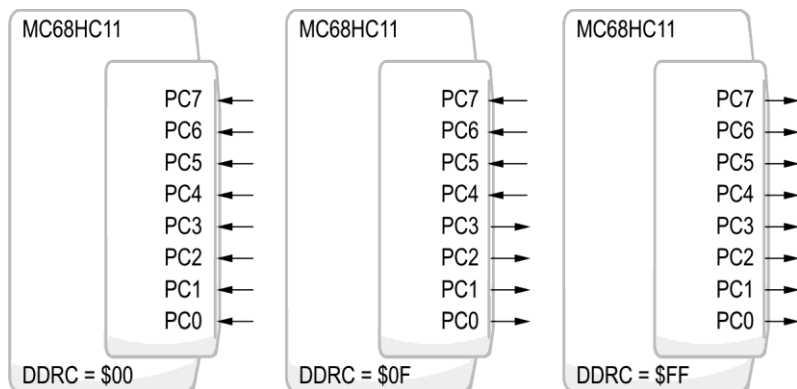
Као што је поменуто код процесора MC68HC11 улазно/излазни портови су меморијски мапирани, што значи да софтвер приступа порту једноставним читавањем садржаја са одговарајуће адресе (за улазни порт), или уписивањем податка на одговарајућу адресу (за излазни порт). Следећи програм показује како се у асемблеру дефинишу адресе неких портова и контролних регистара (у овом случају паралелни улазно/излазни портови и регистри за дефинисање смера). Програм асемблерске дефиниције неких улазно/излазних портова микроконтролера MC68HC11 је:

```

PORTA EQU $1000 ; PA7 улаз/излаз, PA6-3 излази, PA2-0 улази
PACTL EQU $1026 ; бит 7(DDRA7) одређује смер PA7
PORTB EQU $1004 ; PB7-PB0 излази који се могу читавати
PORTC EQU $1003 ; PC7-PC0 могу бити улази или излази
DDRC EQU $1007 ; регистар смера за Порт C
PORTD EQU $1008 ; PD5-PD0 могу бити улази или излази
DDRD EQU $1009 ; регистар смера за Порт D
PORTE EQU $100A ; PE7-PE0 улази

```

Порт C на микроконтролеру MC 68HC11 је осомбитни порт чији прикључци могу бити дефинисани као улази или као излази. На слици 9.2 приказане су три различите хардверске конфигурације порта C.



Слика 9.2 Улазно/излазни смер бидирекционих портова одређен је садржајем регистра смера одговарајућег порта: на пример порта C (DDRC)

9. Микроконтролер MC68HC11

Да би прикључак бидирекционог порта био излазни одговарајући бит у регистру смера (DDR) мора бити постављен на вредност један. По поновном укључењу (ресету) садржај свих регистара смера поставља се на вредност нула, што значи да су сви портови иницијално дефинисани као улази. Према захтевима апликације дефинише се који прикључци ће бити излазни, а који улазни уписивањем одговарајућег садржаја у регистар смера (DDR). На пример помоћу инструкција:

```
LDAA    #$0F
STAA    DDRC
```

дефинишемо прикључке порта C PC0 до PC3 као излазе, а PC4 до PC7 као улазе.

Уколико је потребно, намена појединог прикључка може бити промењена у току рада. Програм који следи пример је рада са прикључком 0 порта C без утицаја на намену осталих прикључака.

Дефинисање прикључка PC0 као излаза

```
INITPC0 LDAA    DDRC
          ORAA    #$01
          STAA    DDRC
          RTS
```

Постављање високог нивоа на прикључак PC0

```
SETPC0   LDAA    PORTC
          ORAA    #$01
          STAA    PORTC
          RTS
```

Постављање ниског нивоа на прикључак PC0

```
SETPC0   CLR     PC0
          LDAA    #PORTC
          ANDA    #$FE
          STAA    PORTC
          RTS
```

9.3 Серијски комуникациони интерфејс

Серијски комуникациони интерфејс (SCI¹) користи се за асинхрони серијски пренос података и састоји се из предајног и пријемног дела. Предајни део серијског комуникационог интерфејса (SCI) укључује TxD прикључак као излаз података TTL нивоа. Предајни део има 10. или 11- битни померачки регистар, коме програм не може директно да приступи (слика 9.3). Овај померачки регистар потпуно је одвојен од пријемног померачког регистра.

Да би се послао податак користећи серијски комуникациони интерфејс (SCI) податак се уписује у предајни регистар података (SCDR²). Предајни регистар података је само за уписивање³, што значи да софтвер може само да уписује у њега (чиме покреће нову предају), али не може да га чита. На тај начин могуће је да предајни и пријемни (који је само за читавање⁴) регистар података буду на истој адреси иако су то физички два одвојена регистра.

Четири контролна бита утичу на предају:

- уписивање податка 1 у TE⁵ контролни бит омогућује⁶ рад предајника;
- постављањем контролног бита SBK⁷ шаље се блок од 10 или 11 нула (зависи од формата речи);
- постављањем TIE⁸ контролног бита дозвољава се генерисање прекида;
- контролни бит TCIE⁹ омогућује да TC¹⁰ флег генерише прекид.

Поред контролних постоје и два бита стања (статус бита) која се постављају у зависности од стања и активности предајника:

¹ *Serial Communications Interface*

² *Serial Communication Data Register*

³ *Write only*

⁴ *Read only*

⁵ *Transmit Enable*

⁶ *Enable*

⁷ *Send Break*

⁸ *Transmit Interrupt Enable*

⁹ *Transmit Complete Enable*

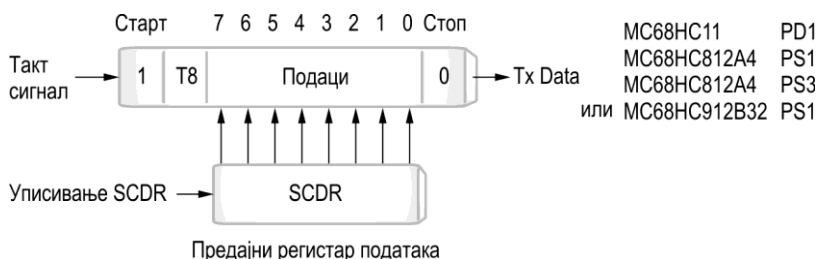
¹⁰ *Transmit Complete*

9. Микроконтролер MC68HC11

- бит TDRE¹ је постављен када је предајни регистар података SCDR празан. Бит TDRE брише се уписивањем податка у предајни регистар података SCDR;
- бит TC се поставља када предајни померачки регистар заврши са слањем податка. Бит TC се брише читавањем, а затим уписивањем податка у регистар података SCDR.

Када се нови податак упише у регистар података SCDR, он се копира у предајни померачки регистар. Затим се додаје СТАРТ бит, T8 (ако је M=1) и СТОП елемент, и почиње слање податка померањем једног бита за другим (бит по бит), брзином дефинисаном податком у регистру бодске брзине².

Пријемни део серијског комуникационог интерфејса (SCI) укључује RxD прикључак као улаз података TTL нивоа и 10-битни или 11-битни померачки регистар, коме програм не може директно приступити (слика 9.3). Примљени податак се учитава из пријемног регистра података (SCDR³) пошто је у питању иста адреса као код предајног регистра података, иако су у питању два физички одвојена регистра.



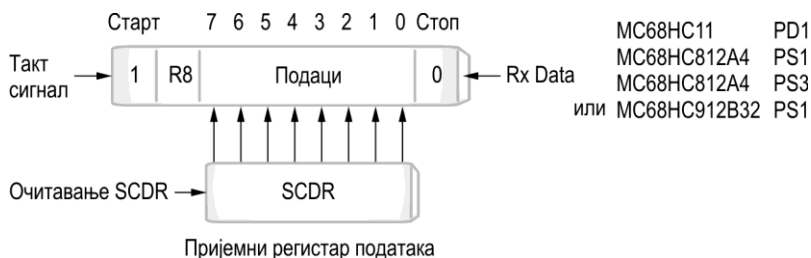
Слика 9.3 Предајни регистар података и померачки регистар реализују серијски пренос⁴

¹ Transmit Data Register Empty

² Baud rate

³ Serial Communication Data Register

⁴ Бит T8 у SCI управљачком регистру (SCCR1) понаша се као додатни бит вишег реда (девети бит) предајног регистра. Девети бит се користи само уколико је M бит у SCCR1 један од оних који одређује формат 9-битног SCCR1 карактера. M бит такође контролише дужину карактера прекида и неактивности (idle). Битови R8 и WAKE у SCCR1 односе се на SCI пријемник.



Слика 9.4. Пријемни померачки регистар реализује пријемни серијски интерфејс

Четири контролна бита утичу на пријем. Уписивање податка 1 у RE¹ контролни бит селекује² пријемник. Уколико је постављен контролни бит RWU³ тада ће активност на пријемној линији укључити рачунар у случају да се он налази у неком од режима за уштеду енергије⁴. Поред тога ту су два контролна бита која дефинишу начин генерисања прекида:

- контролни бит RIE⁵ омогућава да RDRF флег генерише захтев за прекид;
- контролни бит ILIE⁶ омогућава да IDLE флег генерише захтев за прекид.

Поред тога постоји пет статусних бита који се постављају у зависности од стања и активности пријемника:

- RDRF флег се поставља када је доступан нови примљени податак. RDRF бит се брише очитавањем RDRF флега, а затим очитавањем регистра података SCDR;
- IDLE⁷ флег се поставља када пријемна линија постане неактивна. IDLE бит се брише очитавањем IDLE флега, а затим очитавањем регистра података SCDR;

¹ Receiver Enable

² Enable

³ Receiver Wake Up

⁴ Low power или Sleep mode

⁵ Receiver Interrupt Enable

⁶ Idle Line Interrupt Enable

⁷ Idle Line dEct

- OR¹ флег се поставља када дође до губљења примљеног податка, пошто је стигао нови податак а претходни није очитан. OR бит се брише читавањем OR флега, а затим читавањем регистра података (SCDR);
- NF² флег се поставља када на пријемној линији постоји сметња (шум). NF бит се брише читавањем NF флега а затим читавањем регистра података (SCDR). У пријемнику се сваки примљени бит одмерава³ три пута и NF флег је обрисан ако може да се донесе неспорна одлука. Ако у било којој групи од три узорка постоји неслагање NF флег се поставља. Узрок грешке (постављен NF бит) може бити стварни шум на линији, али често је узрок и несагласност брзина на предаји и пријему.
- FE⁴ се поставља када није примљен исправан стоп бит. FE бит се брише читавањем FE флега, а затим читавањем регистра података SCDR. Грешка рама (FE бит постављен) често је, такође, проузрокована нескладом брзина на предаји и пријему.

Када пријемник детектује силазну ивицу на пријемном прикључку (почетак старт бита) он изврши померање следећих 10 или 11 битова податка који пристижу на RxD прикључак. СТАРТ бит и СТОП елемент се уклањају, уз проверу да ли постоји грешка услед сметње или грешка рама. Осам битова податка пребацује се у регистар података (SCDR). Ако постоји (ако је M=1) девети бит се пребацује у R8, и поставља се RDRF флег. Уколико већ постоји податак у регистру података (SCDR) када је померачки регистар завршио сачекаће се да софтвер чита претходни податак. Грешка услед губитка података јавља се када је један примљени податак у регистру података (SCDR), други у пријемном померачком регистру, а трећи стиже на пријемни прикључак RxD.

У табели 9.1 приказани су портови који су важни за имплементацију функција серијског комуникационог интерфејса (SCI⁵).

¹ *Overrun*

² *Noise*

³ *Sample* – одмерак, узорак, примерак...

⁴ *Framing Error*

⁵ *Serial Communication Interface*

Адреса	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	Назив
\$102B	TCLR	SCP2	SCP1	SCP0	RCKB	SCR2	SCR1	SCR1	BAUD
\$102D	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK	SCCR2
\$102E	TDRE	TC	RDRF	IDLE	OR	NF	FE	0	SCSR
\$102F	R7/T7	R6/T6	R5/T5	R4/T4	R3/T3	R2/T2	R1/T1	R0/T0	SCDR

Табела 9.1 Портови серијског интерфејса MC68HC11

BAUD регистар се користи за дефинисање линијске брзине преноса података. Он одређује вредности два целобројна фактора којима се дели учестаност E- такт сигнала процесора да би се добила одговарајућа вредност линијске брзине (табела 9.2)

Два бита SCP1 и SCP0 дефинишу фактор прескалера. SCP2 додаје дељење са 39 у прескалеру, али постоји само код неких модела породице MC68HC11. Три бита SCR2, SCR1 и SCR0 дефинишу други фактор дељења, а линијска брзина се може израчунати коришћењем израза:

$$R_{SCI} = E / (16 * P * BR)$$

SCP1	SCP1	Прескалер ¹ , P
0	0	1
0	1	3
1	0	4
1	1	13

SCR2	SCR1	SCR0	Фактор дељења ² , BR
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Табела 9.2 Одређивање линијске (бодске брзине) код микроконтролера MC68HC11

¹ Prescaler² Divider

BAUD	Линијска брзина [b/s]
\$30	9600
\$31	4800
\$32	2400
\$33	1200

Табела 9.3 Вредности BAUD регистра за неке уобичајене брзине

У табели 9.3 дате су вредности BAUD регистра за неке уобичајене брзине преноса у случају када је такт-сигнал процесора $E = 2\text{MHz}$.

SCCR2 контролни регистар садржи битове којима се укључују и искључују предајник и пријемник SCI (TE и RE), и битове којима се омогућавају или забрањују одговарајући прекиди.

SCSR је статусни регистар који може бити само прочитан (његов садржај се не може софтверски изменити уписивањем у регистар). Флегови које садржи овај регистар описани су у претходном тексту.

Софтвер за слање и пријем података мора да обезбеди механизме за синхронизацију рада софтвера и хардвера. Податак се читава тек када је комплетно примљен. Такође нови податак се шаље тек када је претходни послат. Асемблерски програм који следи приказује како се реализује функција серијског преноса у режиму условног улаза/излаза.

```

;Inicijalizacija MC6811 SCI
;Ulaz: nema; Izlaz: nema
SCI_Init
        LDAA    #$0C
        STAA    SCCR2        ;selektovanje1 SCI
        LDAA    #0
        STAA    BAUD        ;125000b/s
        RTS
;Citanje 8 bita iz serijskog porta
;Vraca bajt podatka u akumulator A
RDFR EQU $20
SCI_InChar
        LDAA    SCSR        ;status

```

¹ Enable

```

        ANDA    #RDFR
        BEQ     SCI_InChar    ;ceka RDFR
        LDAA    SCDR          ;cita bajt
        RTS

                                ;Upis 8 bita na serijski port
                                ;Ulaz: bajt u akumulatoru A

TDRE EQU $80
SCI_OutChar
        LDAB    SCSR          ;status
        ANDB    #TDRE
        BEQ     SCI_OutChar    ;ceka TDRE
        STAA    SCDR          ;pise bajt
        RTS

```

9.4 Синхрони периферијски интерфејс

Типично синхрони периферијски интерфејс (SPI¹) користи се за повезивање додатних улазно/излазних уређаја или периферија са микроконтролером. Основна разлика између серијског периферијског интерфејса, који користи асинхрони протокол, и оног који користи синхрони протокол јесте начин на који се ради са такт сигналом. Два уређаја која комуницирају користећи асинхрони серијски интерфејс (SCI²) раде на истој учестаности али користе два одвојена (несинхронизована) такт сигнала. Два уређаја (модула) који комуницирају користећи синхрони серијски интерфејс (SPI) користе исти такт сигнал. Типично примарни³ (главни) уређај генерише такт сигнал, а секундарни⁴ (пратећи) користи тај такт сигнал за слање и пријем података. Синхрони периферијски интерфејс (SPI) произвођача *Motorola* има четири улазно/излазне линије. Селектовање секундарног уређаја⁵ (SS) је опциони контролни сигнал од примарног до секундарног уређаја, који указује да је канал активан. Друга линија SCIK

¹ *Synchronous Peripheral Interface*

² *Serial Communication Interface*

³ *Master*

⁴ *Slave*

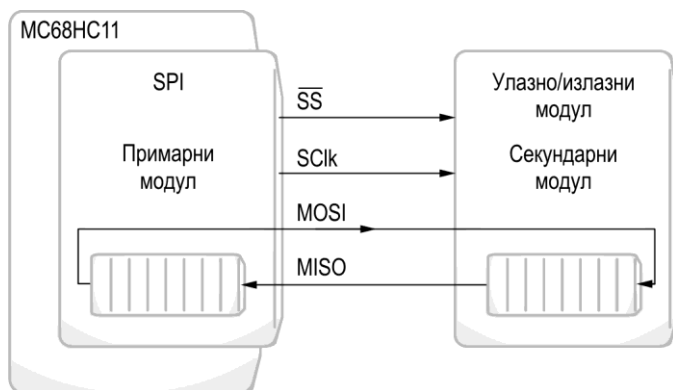
⁵ *Slave Select*

9. Микроконтролер MC68HC11

је такт сигнал који генерише примарни уређај. Линијом MOSI¹ податаке шаље примарни а прима секундарни уређај. Линијом MISO² податке шаље секундарни а прима примарни уређај. Да би систем исправно радио предајни уређај користи једну ивицу такт сигнала да би променио стање на излазу, док пријемни уређај користи следећу ивицу да би прочитао податак.

Мада се синхрони периферијски интерфејс (SPI) може користити за међусобно повезивање микроконтролера најчешће се користи за везу микроконтролера са екстерним модулима. У том случају синхрони периферијски интерфејс (SPI) у микроконтролеру ради као примарни, а спољни (екстерни) модули су секундарни. У синхронном периферијском интерфејсу (SPI) осмобитни регистар података (SPDR) у примарном уређају и осмобитни регистар података у секундарном уређају повезани су и формирају дистрибуирани шеснаестобитни регистар. Слика 9.5 приказује везу између микроконтролера и једног екстерног модула.

Селектовање секундарног модула ($\overline{SS}$) користи се да активира екстерни улазно/излазни модул, док SClk помера податке у оба померачка регистра. Када се врши пренос података овај шеснаестобитни регистар серијски се помера за осам битова помоћу SClk такт сигнала из примарног модула, тако да ефективно долази до замене података између примарног



Слика 9.5 Синхрони периферијски интерфејс између микроконтролера и улазно/излазног модула

¹ Master out Slave in

² Master in Slave out

ри секундарног модула. Подаци уписани у SPDR регистар примарног модула пребачени су у секундарни. Пошто је такт сигнал заједнички, ако се промени учестаност Е-такт сигнала промениће се брзина преноса, али ће синхрони периферијски интерфејс (SPI) и даље радити исправно.

Порт синхроног периферијског интерфејса код микронтролера MC68HC11 користи четири прикључка PD5=SS, PD4=SClk, PD3=MOSI и PD2=MISO. Када је MC68HC11 примарна станица треба (коришћењем DDRD регистра) дефинисати да PD5, PD4 и PD3 буду излази а PD2 улаз. У табели 9.4 приказани су портови које користи синхрони периферијски интерфејс (SPI).

Контролни регистар SPCR садржи конфигурационе битове који одређују начин рада.

- SPIE¹ бит служи да се дозволи прекид;
- SPE² активира синхрони периферијски интерфејс (SPI);
- DWOM бит у SPCR регистру користи се за постављање свих шест прикључака порта D у режим рада са отвореним дрејном, или их дефинише као нормалне CMOS излазе:
 - Када је DWOM бит нула тада су PD5- PD0 стандардни CMOS сигнали. Стандардни CMOS излаз има два стања: ниско (логичка нула, 0V) или високо (логичка јединица, +5V),
 - Када је DWOM бит један, тада је сваки прикључак порта D отворени дрејн³. Отворени дрејн значи да излаз прикључка може бити логичка нула, или пливајући⁴, када се понаша као да је откачен од остатка кола;
- MSTR је бит за селектовање примарног начина рада⁵. Када је

Адреса	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀	Назив
\$1009	0	0	DDRD5	DDRD4	DDRD3	DDRD2	DDRD1	DDRD0	DDRD
\$1028	SPIE	SPE	DWOM	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
\$1029	SPIF	WCOL	0	MODF	0	0	0	0	SPSR
\$102A	Bit 7	6	5	4	3	2	1	Bit 0	SPDR

Табела 9.4 Портови синхроног серијског интерфејса MC68HC11

¹ Serial Peripheral Interrupt Enable

² Serial Peripheral System Enable

³ Open drain

⁴ Three state

⁵ Master Mode Select

SPR1	SPR0	Делитељ	Брзина преноса	Битски интервал
0	0	2	1 MHz	1 μ s
0	1	4	500 kHz	2 μ s
1	0	16	125 kHz	8 μ s
1	1	32	62,5 kHz	16 μ s

Табела 9. 5. Избор брзине преноса за синхрони серијски порт

овај бит постављен (уписано 1) MC68HC11 ради као примарна станица;

- CPOL и CPHA битови дефинишу поларитет и фазу такт сигнала;
- Контролни битови SPR1 и SPR0 одређују брзину преноса. У табели 9.5 могу се видети брзине преноса у случају када је учестаност Е-такт сигнала 2MHz.

Регистар SPSR (*SPI Status Register*) садржи статус битове синхроног периферијског интерфејса (SPI).

- SPIF је бит који показује да је пренос завршен¹. SPIF се брише читавањем SPSR, а затим приступом SPDR регистру.
- WCOL² бит указује на колизију при упису. Он се поставља ако се у SPDR упише податак док је пренос у току. Бит WCOL се брише читавањем SPSR, а затим приступом SPDR регистру.
- MODF³ бит се поставља уколико $\overline{SS}$ линија буде споља спуштена на нулу док је MSTR=1. Пошто је ово случај колизије у систему појава ове грешке прекида рад SPI.

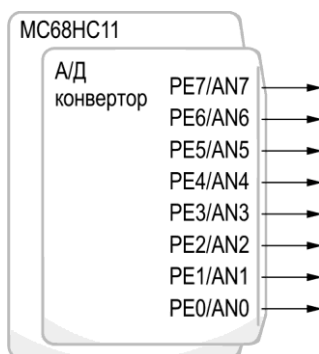
9.5 Аналого/дигитални конвертори

Микроконтролери породице MC68HC11 имају уграђене осмоканалне, осмобитне аналого/дигиталне конверторе, на чији улаз може да се доведе аналогни сигнал у опсегу од 0 до 5V, и који као излаз дају дигиталну вредност између 0 и 255 (слика 9.6). Конвертор може да обави једну секвенцу конверзије и тада стане, или да континуално обавља А/Д конверзију. Такође, може да се обавља конверзија на

¹ SPI Transfer Complete Flag

² Write Collision

³ Mode fault



Слика 9.6. A/Д конвертор код MC68HC11 користи прикључке порта Е једном каналу више пута, или да се врши конверзија на више канала истовремено.

У табели 9.6 приказана је вредност дигиталног излаза у зависности од улазног напона за неке вредности улазног напона.

Аналогоно/дигитални систем код микроконтролера MC68HC11 се укључује постављањем бита ADPU у OPTION регистру. Када је бит ADPU нула порт Е ради као обичан паралелни улазни порт. Контролни бит CSEL служи за избор такт сигнала за напонску пумпу. Бит CSEL треба да буде 0 за нормалне фреквенције Е-такт сигнала. Уколико је фреквенција Е-такт сигнала испод 750kHz бит CSEL треба да буде 1. У табели 9.7 приказани су портови које користи систем за A/Д конверзију.

ADCTL битови 5-0 у ADCTL регистру користе се за дефинисање начина рада A/Д конвертора, и почетак конверзије. Свака секвенца од четири конверзије захтева 128 циклуса Е-такт сигнала да би се комплетирали. Када конвертор ради у режиму појединачних конверзија секвенца конверзије се започиње уписивањем податка у ADCTL.

Аналогни улазни напон (V)	Дигитални излаз
0.00	%00000000 \$00
0.02	%00000001 \$01
2.50	%10000000 \$80
3.75	%11000000 \$C0
5.00	%11111111 \$FF

Табела 9.6. Зависност дигиталног излаза од улазног напона

9. Микроконтролер MC68HC11

Адреса	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀	Назив
\$100A	PE7	PE6	PE5	PE4	PE3	PE2	PE1	PE0	PORTE
\$1030	CCF	0	SCAN	MULT	CD	CC	CB	CA	ADCTL
\$1031	Bit 7	6	5	4	3	2	1	Bit 0	ADR1
\$1032	Bit 7	6	5	4	3	2	1	Bit 0	ADR2
\$1033	Bit 7	6	5	4	3	2	1	Bit 0	ADR3
\$1034	Bit 7	6	5	4	3	2	1	Bit 0	ADR4
\$1039	ADPU	CSEL	IRQE	DLY	CME	0	CR 1	CR 0	OPTION

Табела 9.7 Портови система за А/Д конверзију

CCF¹ је бит који може само да се читава и означава да ли је конверзија завршена. Нула указује да је конверзија у току, а 1 да је завршена. Бит CCF се брише уписивањем податка у ADCTL (односно покретањем конверзије). Када је бит CCF=1, ADR1, ADR2, ADR3 и ADR4 садрже исправне податке (резултате конверзије).

SCAN је контролни бит којим се дефинише да ли ће се обавити једна конверзија, или ће се конверзија обављати континуално. Када је SCAN =0 А/Д конвертор обави једну секвенцу конверзије и тада се зауставља. Када је SCAN =1 конверзија се обавља континуално.

MULT је контролни бит којим се одређује да ли ће се вршити једноканална или вишеканаална конверзија. У бит MULT уписујемо 0 када желимо конверзију само на једном каналу. Када желимо да се конверзија обавља на скупу канала (0, 1, 2, 3) или (4, 5, 6, 7) у MULT уписујемо 1. У мулти-канал начину рада (MULT=1) мултиплексерски

CD, CC, CB, CA	Канал
0000	PE0 аналогни канал 0
0001	PE1 аналогни канал 1
0010	PE2 аналогни канал 2
0011	PE3 аналогни канал 3
0100	PE4 аналогни канал 4
0101	PE5 аналогни канал 5
0110	PE6 аналогни канал 6
0111	PE7 аналогни канал 7

Табела 9.8. Контрола мултиплексера

¹ Conversion Complete Flag

контролни битови CD и CC специфицирају групу канала, а мултиплексерски контролни битови CB и CA се игноришу. Битови CD, CC, CB, CA дефинишу на ком каналу се врши конверзија, као што је приказано у табели 9.8.

Пример за рад на једном каналу је следећи програм:

```

;*****Inicijalizacija ADC*****
; Ulaz: nema    Izlaz: nema
                ADC_Init
                LDAB    #$80
                STAB     OPTION    ;selektuje1 A/D konvertor
                RTS
;*****Uzimanje odbirka*****
;Ulaz: U akumulator B je redni broj kanala
;Izlaz: U akumulator B je vrednost odbirka
ADC_In
                STAB     ADCTL
WAIT           LDAB     ADCTL
                ANDB     #$80
                BEQ      WAIT    ;ceka se CCF
                LDAB     ADR1
                RTS

```

9.6 Систем часовника микроконтролера MC68HC11

Основа система часовника код микроконтролера MC68HC11 је шеснаестобитни бројач који се инкрементира у једнаким интервалима, а софтверски је могуће прочитати његов садржај у било ком тренутку. Овај бројач може се користити за генерисање импулса, четвртки, или ширински модулисаног сигнала. Може се употребити за мерење периода, ширине импулса или учестаности долазног сигнала.

У табели 9. 9 приказани су портови часовника које користи сам бројач.

TCNT је шеснаестобитни бројач који непрекидно броји и инкрементира се учестаношћу која зависи од вредности Е-такт сигнала и вредности два

¹ Enable

9. Микроконтролер MC68HC11

Адреса	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	Назив
\$100E	Bit 15	14	13	12	11	10	9	Bit 8	TCNT
\$100F	Bit 7	6	5	4	3L	2	1	Bit 0	TCNT
\$1024	TOI	RTII	PAOVI	PAII	0	0	PR1	PR0	TMSK2
\$1025	TOF	RTIF	PAOVF	PAIF	0	0	0	0	TFLG2

Табела 9.9 Портови часовника

прескалерска бита (PR1 и PR0) у TMSK2 регистру, као што је приказано у табели 9.10. Сваки пут када бројач одброји до краја (промена са \$FFFF на 0) поставља се TOF бит (флег) у TFLG2 регистру. Уколико је маска TOI у TMSK2 регистру постављена биће генерисан захтев за прекид. Битови у TFLG2 регистру бришу се уписивањем јединице на позицију бита који желимо да обришемо. На пример TOF флег се брише уписивањем вредности \$80 у TFLG2.

Следећи програм показује како се може реализовати тачно дефинисано кашњење помоћу часовника. Параметар којим се дефинише кашњење (дефинисано бројем такт сигнала часовника) може бити било који број од 1 до 65535.

```

; U akumulatoru D je vreme cekanja u ciklusima
WAIT      ADDD      TCNT      ;vreme kraja
WLOOP     CPD       TCNT      ;endT<TCNT
          BPL       WLOOP
          RTS

```

PR1	PR0	Фактор прескалера	Фреквенција кристала		
			223 Hz	8 MHz	4 MHz
			резолюција/опсер		
0	0	1	477ns/31,25ms	500ns/32,77ms	1µs/65,54ms
0	1	4	1,91µs/125ms	2µs/131,1ms	4µs/262,1ms
1	0	8	3,81µs/250ms	4µs/262,1ms	8µs/524,3ms
1	1	16	7,63 µs/0,05s	8µs/524,3ms	16µs/1,049s
			2,1MHz	2MHz	1MHz
			E такт сигнал		

Табела 9.10 Такт бројача (резолюција) и време до премашаја (опсер) у зависности од фреквенције кристала и вредности PR1 и PR0

9.7 ПИТАЊА И ЗАДАЦИ

1. Микроконтролер има свој сопствени, уграђени UART. Који се од прикључака портова могу користити као асинхрони серијски комуникациони интерфејс између микроконтролера улазно/излазних модула?
2. Написати програмску секвенцу која ће селектовати SCI секцију микроконтролера MC68HC11 за асинхрону серијску комуникацију.
3. Описати функцију DDR регистра микроконтролера MC68HC11.
4. Написати програмску секвенцу која ће иницијализовати битове 7, 4, 3, 0 порта C као излазне а битове 6, 5, 2, 1 као улазне поставити све портове. Поставити све битове порта D као излазне.
5. Написати програмску секвенцу која ће иницијализовати битове 5, 4, 1, 0 порта C као излазне а битове 7, 6, 3, 2 као улазне. Поставити све портове. Поставити све битове порта D као улазне, а све битове порта B као излазне.
6. Написати програмску секвенцу која ће иницијализовати бит 1 порта D као улазни. Сви остали DDRD битови остају непромењени.
7. Какав утицај има стање DWOM бита регистра SPCR на порт D.
8. Описати функцију бита RIE регистра SCCR2.
9. Описати функцију бита RDRF регистра SCCR2.
10. Написати програмску секвенцу која проверава да ли јесте или није детектована грешка рама за примљени карактер који се тренутно налази у RDR регистру.
11. Написати програмску секвенцу која брише TDRE флег у SCSR регистру.
12. Који је највећи децимални број који часовник микроконтролера MC68HC11 може да достигне?
 - а) Показати како се TCNT бројач може поново покренути (ресетовати) да би започео са бројањем од \$0000
 - б) Дати пример како се може очитати садржај TCNT бројача
13. Претпоставити да је учестаност кристала кварца MC68HC11 8MHz. Одредити следеће:
 - а) Колико дуго треба TCNT бројачу да поново почне бројање¹ уколико

¹ Recycle

у TMSK2 регистру флегови имају следеће вредности:

I. PR0=1, PR1=0

II. PR0= PR1=1

- б) Шта доводи до тога да се TOF бит у TFLG2 регистру поставља на вредност 1?
 - в) Написати програмску секвенцу која ће само обрисати флаг TOF и PAOVF у TFLG2 регистру.
14. Написати програмску секвенцу која ће одмерити канал 2 А/Д конвертора четири пута, израчунати средњу вредност четири одмерка, и вратити резултат у акумулатор А.
15. Написати програмску секвенцу која ће одмерити свих осам канала А/Д конвертора, израчунати минимум и максимум тих осам одмерака и вратити опсег (минимум-максимум) у акумулатор А.

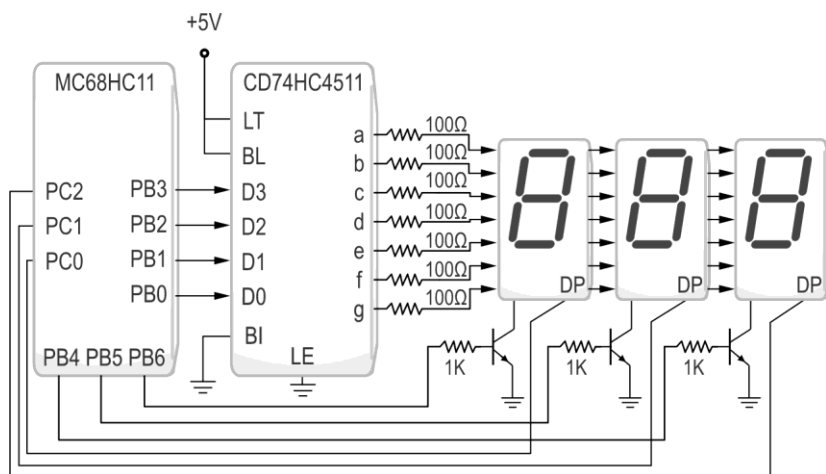
10. Примена микроконтролера серије MC68HC11

У овом поглављу биће детаљно анализирани примери како се микроконтролер MC68HC11 може употребити у реализацији једноставних и сложенијих уређаја различите намене.

10.1 Повезивање екрана са три седмосегментна LED показивача и микроконтролера

Често се показује потреба за повезивањем екрана за приказ бројева на микрорачунар. Блок шема једног од начина реализације овог повезивања уз коришћење микроконтролера MC68HC11 дата је на слици 12.1.

Подаци које микроконтролер шаље на LED показивач су BCD



Слика 10.1 Повезивање екрана састављеног од седмосегментних LED показивача и микроконтролера MC68HC11

10. Примена микроконтролера серије MC68HC11

кодирани. Свака цифра LED показивача повезује се са 7 линија леч декодера¹ и једном од три линије са С паралелног порта микроконтролера. Цифре које се приказују могу бити само децималне. LED показивач ради у режиму освежавања цифара. У једном временском интервалу светле сегменти само једног седмосегментног показивача. Временски интервал у коме светли један показивач је приближно 5ms и реализује се софтверском петљом за кашњење. Цифра коју треба приказати избацује се на линије порта В (PB3-PB0). Селекција седмосегментног показивача који у том тренутку светли врши се преко линија порта В (PB4-PB6). Ако је потребно да светли седмосегментни показивач највеће тежине на линију PB6 треба избацити логичку јединицу а на линије PB5 и PB4 логичке нуле. Када треба да светли децимална тачка испред цифре најмање тежине треба избацити логичку јединицу на порт PC2 а на линије PC1 и PC0 логичке нуле. У табели 10.1 представљени су примери исписивања на седмосегментни LED показивач у зависности од садржаја на линијама порта В (PB3-PB0).

4-битни податак на линијама PB3-PB0	Приказ на екрану
\$0	0
\$1	1
\$9	9

Табела 10.1 Пример приказан на екрану

У табели 10.2 приказан је садржај који треба избацити на линије PB6, PB5, PB4 да би светлела жељена цифра.

Податак на линијама PB6, PB5 и PB4	Цифра која светли
100	Цифра највеће тежине
010	Средња цифра
001	Цифра најмање тежине

Табела 10.2 Избор цифре која светли

¹ BCD на седам сегмената

Могуће је додати и децималну тачку. У табели 10.3 представљене су одговарајуће вредности линија порта С и приказ на LED показивачу уз претпоставку да је податак \$321.

Податак на линијама PC2 PC1 PC0	Приказ на LED показивачу
000	+321
001	+32.1
010	+3.21
100	+.321

Табела 10.3 Пример приказа децималне тачке на LED показивачу

Програм 10.1 Програм за повезивање микроконтролера и LED показивача

Пример програма који на LED показивачу приказује податак 321.

```

PORTA      EQU  $1000
PORTC      EQU  $1003
PORTB      EQU  1004
DDRC       EQU  $1007
            ORG  $0
CIFRA3     RMB  1      ;cifra najveće težine
CIFRA2     RMB  1      ;srednja cifra
CIFRA1     RMB  1      ;cifra najmanje težine
            ORG  $$FF
STACK      RMB  1
; Početak programa je na adresi $B600
            ORG  $B600
; Definisanje pokazivača stek memorije
            LDS  #STACK
            JSR  LED_init
            LDAA #$3
            STAA CIFRA3
            LDAA #$2
            STAA CIFRA2

```

```

                                LDAA  #$1
                                STAA  CIFRA1
START                          LDAA  CIFRA3
                                ANDA  #$0F
                                ORAA  #$40
                                STAA  PORTB
                                JSR   Kasni5ms
                                LDAA  CIFRA2
                                ANDA  #$0F
                                ORAA  #$20
                                STAA  PORTB
                                JSR   Kasni5ms
                                LDAA  CIFRA1
                                ANDA  #$0F
                                ORAA  #$10
                                STAA  PORTB
                                JSR   Kasni5ms
                                BRA   START

; Inicijalizacija LED
; Ulaz: nema (none)
; Izlaz: nema (none)
LED_init                      LDAA  #$FF
                                STAA  DDRB
                                STAA  DDRC
                                RTS

; Petlja za kašnjenje sa trajanjem od 5ms. Instrukciski ciklus je 0,5μs
; Izlaz: nema (none)
Kasni5ms                      LDX   #$500
PET                            DEX
                                NOP
                                NOP
                                BNE  PET
                                RTS
                                ORG  $FFFE
                                FDB  START

```

10.2 Управљање једносмерним мотором

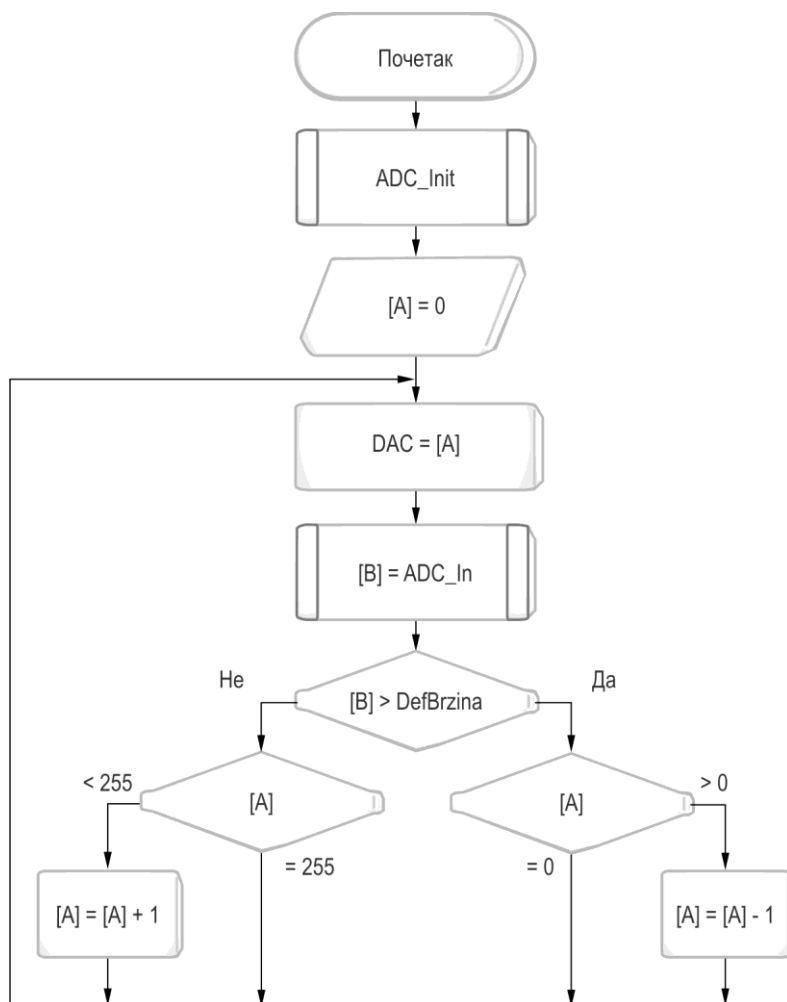
За управљање једносмерним мотором (DC¹) може се употребити микроконтролер МС68НС11, 8-битни Д/А конвертор и тахометар (сензор) којим се мери брзина мотора. Д/А конвертор је повезан са 8-битним излазним портом. Опсег промене напона доведеног на мотор је од минималне вредности (0) до максималне вредности која се добија када се у Д/А конвертор упише 255. Сензор мери брзину мотора, и као излаз даје напон пропорционалан брзини. На пример, за сензор РО2211 вредност излазног напона је у опсегу 0V до 5V. Уколико се употреби сензор који даје други опсег напона требало би убацити степен за нормализацију који ће обезбедити да напон на улазу у А/Д конвертор буде у опсегу од 0V до 5V, тако да податак о брзини буде вредност од 0 до 255. На слици 10.2 представљен је блок дијаграм система за управљање мотором.

Намена овог управљачког система је да управља брзином мотора. Са DefBrzina означена је жељена брзина мотора. Инкрементални управљачки алгоритам једноставно додаје или одузима константу од вредности акумулатора А, која дефинише брзину (вредност за Д/А конвертор) у зависности од знака грешке (слика 10.3, програм 10.2). Другим речима ако је брзина превише мала вредност акумулатора А се увећава, а ако је брзина велика, вредност акумулатора А се умањује. Битно је изабрати одговарајућу брзину којом се инкременталан управљачки софтвер извршава. Уколико се не извршава довољно често, систем неће реаговати на промене које се



Слика 10.2 Блок дијаграм система за управљање брзином мотора

¹ Direct Current



Слика 10.3 Дијаграм тока контролера брзине који користи инкрементално управљање

дешавају на објекту којим се управља или на промене DefBrzina. Код овог инкременталног контролера додајемо или одузимамо 1 од управљачке величине. Уколико је вредност већа од 1 одзив је бржи али може да дође до осциловања. Аналогно/дигитални конвертор користи се за мерење

брзине мотора, а дигитално/аналогни конвертор за контролу снаге мотора. Може се приметити да је решење са инкременталним контролером доста добро са становишта тачности и стабилности за широк број апликација. Недостатак овог решења је у томе што је одзив¹ овог контролера спор. У реалном систему, после Д/А конвертора морао би да постоји још један степен (драјвери) који би омогућио већу снагу, али то не би утицало на софтверску реализацију (програм је идентичан).

Програм 10.2 Асемблерски програм за инкрементални контролер

```

PORTB      EQU   $1004
ADCTL      EQU   $1030
ADR1       EQU   $1031
OPTION     EQU   $1039
           ORG   $0
DefBrzina  RMB   1
           ORG   $B600
MAIN       LDS   #$FF
; Upravljanje aktuatomom
           JSR   ADC_Init
           LDAA  #0
; Izlaz ka aktuatoru
PETLJA     STAA  PORTB
           LDAB  #2
; Očitavanje senzora
           JSR   ADC_In
; Željena brzina
           CMPB  #DefBrzina
           BLO   SPORO
; Već je na minimumu
BRZO       CMPA  #0
           BEQ   PETLJA
; Smanjenje brzine

```

¹ Response time

```

                                DECA
                                BRA  PETLJA
; Već je na maksimumu
SPORO                          CMPA  #$FF
                                BEQ  PETLJA
; Povećanje brzine
                                INCA
                                BRA  PETLJA
                                ORG  $FFFE
                                FDB  MAIN
; Potprogram za inicijalizaciju ADC
; Ulaz: nema
; Izlaz: nema
                                ADC_Init
                                LDAB  #$80
; Uključuje A/D konvertor
                                STAB  OPTION
                                RTS
; Potprogram za uzimanje odbirka
; Ulaz: u akumulatoru B je redni broj kanala
; Izlaz: u akumulatoru B je vrednost odbirka
                                ADC_In
; Početak konverzije
                                STAB  ADCTL
WAIT                            LDAB  ADCTL
                                ANDB  #$80
; Provera da li je konverzija završena
                                BEQ  WAIT
                                LDAB  ADR1
                                RTS

```

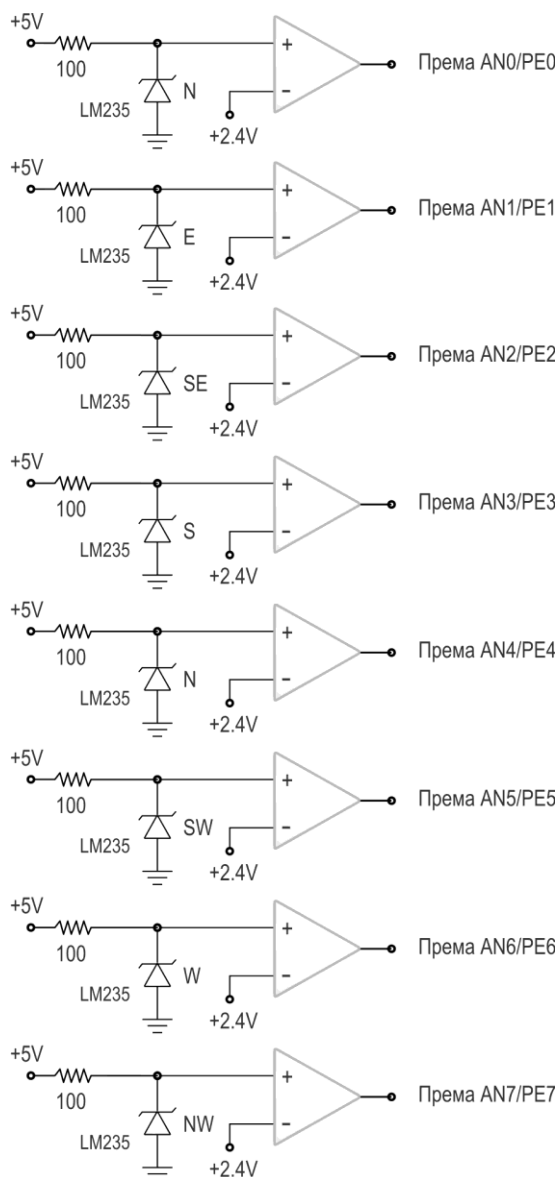
10.3 Полупроводнички ветроказ

У примеру који следи приказана је једна реализација ветроказа са могућношћу даљинског читавања. Ветроказ са могућношћу даљинског

очитавања могуће је реализовати тако што ће се класичан ветроказ повезати са прецизним потенциометром који може да ротира 360° . Вредност очитана са потенциометра може да се проследи до места где треба приказати смер ветра. Проблеми могу да настану када је влажност вазуа велика а температуре су ниске. Тада уређај може да се замрзне и да не показује прави смер ветра. Познат је случај када је ветроказ реализован на овај начин цео месец показивао да ветар дува у истом смеру - све док није довољно отоплило да систем поново проради. Пошто је реч о механичком уређају практично је немогуће учинити га идеално водонепропусним, што после неког времена доводи до блокаде у раду. Очигледно је да би право решење био ветроказ без покретних делова. Идеја је да се искористи особина ветра да хлади површину преко које дува (струјање ваздуха одводи топлоту). Замислимо да имамо преграду у правцу исток – запад, и са сваке стране преграде грејач и температурни сензор. Јасно је да ће, у случају да дува северни ветар сензор са северне стране преграде бити осетно хладнији од сензора са јужне стране, и обрнуто у случају да дува јужни ветар сензор са јужне стране биће хладнији. Уколико је преграда постављена у смеру север - југ моћи ћемо да одредимо да ли дува источни или западни ветар. На исти начин може да се одреди да ли дува североисточни, југозападни, северозападни или југоисточни ветар. Ако поставимо систем од четири кутије отворене са стране и постављене под заклон од сунца, са преградама под углом од 45° , моћи ћемо да одредимо смер дувања ветра тако што ћемо наћи који је од осам сензора најтоплији. Одређивање који је сензор најтоплији може да се обави и коришћењем дискретних компоненти (матрица напонских компаратора и пратећих кола), али у том случају сензори морају да буду скоро идеални, а цена једног таквог сензора вишеструко премашује цену микроконтролера МС68НС11. Коришћењем микроконтролера МС68НС11 можемо да искористимо његову „памет“ и уз статистичку анализу добијемо тачан резултат са јефтиним, комерцијално доступним, сензорима. Установљено је да су и код најјефтинијих сензора средње карактеристике три сензора скоро идентичне средњим карактеристикама било која три друга сензора.

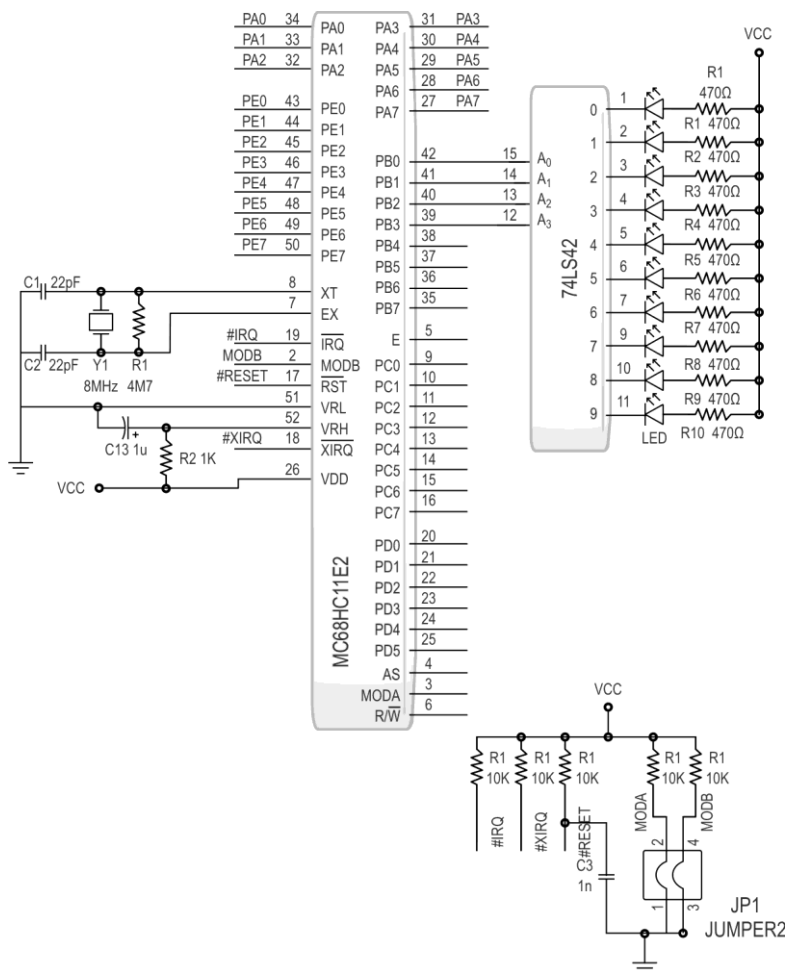
Мада су температурни сензори интегрисана кола са више од 15 транзистора (на пример сензори серија LM135/235/335 произвођача National Semiconductor) они раде као зенер диоде са пробојним напонам директно

10. Примена микроконтролера серије MC68HC11



Слика 10.4 Систем сензора

пропорционалним апсолутној температури ($+10\text{mV}/^\circ\text{K}$). Када се мери апсолутна температура интерно загревање изазвано реверзном струјом кроз сензор није пожељно, тако да ови сензори нормално раде са малим струјама (обично испод 1mA). Међутим, нама треба загревање тако да ће сензори радити са струјама блиским максимално дозвољеној. Због интерног



Слика 10.5 Електрична шема полупроводничког ветроказа са микроконтролером MC68HC11A8

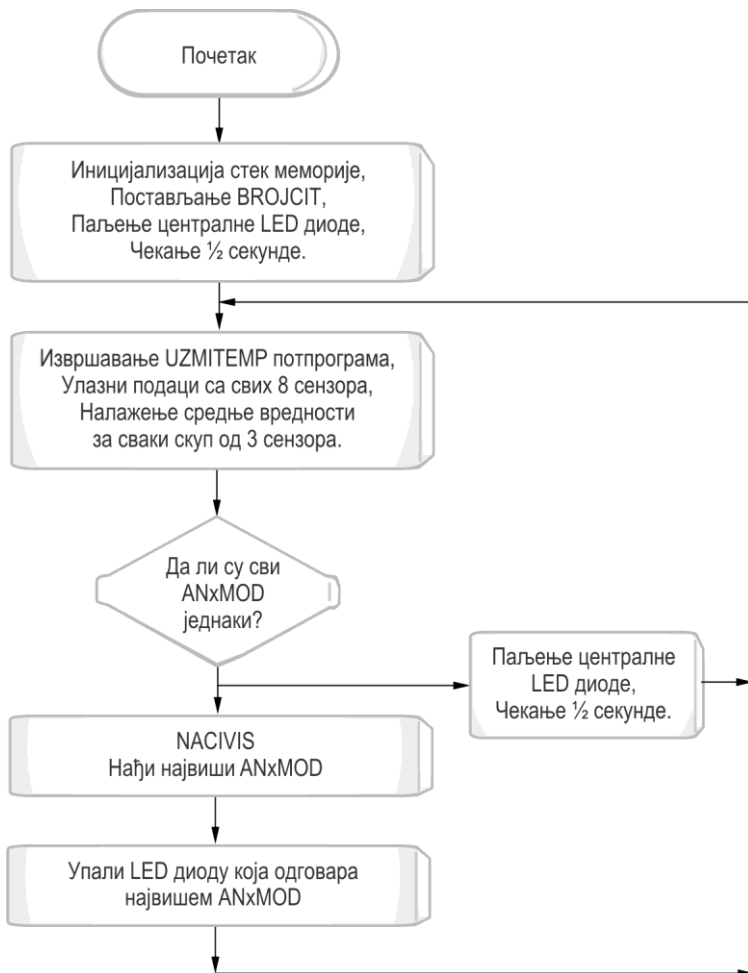
загревања сензор у заветрини биће најтоплији, а два суседна сензора биће нешто хладнија. Када ветар дува остали сензори биће значајно хладнији. Излази сензора се воде на операционе појачаваче који обезбеђују промену напона у опсегу 0-5V (пун опсег рада интегрисаног А/Д конвертора). Поједностављена шема овог склопа приказана је на слици 10.4

За приказивање резултата користи се екран који се састоји од 9 LED диода и декодера BCD кода у десет сигнала. Осам LED диода је постављено у круг да показује одговарајући смер ветра (север, североисток, исток итд.). Девета LED диода је постављена у центар и показује стање потпуно без ветра, екстремно високу или ниску температуру или стање грешке.

Пошто је предвиђено да се комплетан програм смести у EEPROM на интегрисаном колу користи се микроконтролер MC68HC11A8 који ради као једно интегрисано коло¹ (слика 10.5). Ограничење у величини програма диктира писање програма у асемблеру.

Први корак у писању програма је дефинисање стек меморије уписивањем одговарајуће адресе у показивач стек меморије. Други корак је дефинисање променљиве BROJCIT која дефинише после колико читавања сваког сензора се налази средња вредност тих читавања, и та вредност се памти као читана вредност за тај сензор. Затим се пали централна LED диода на $\frac{1}{2}$ секунде. Потом учитавамо свих осам температура са сензора, а затим налазимо средњу вредност за свака три суседна сензора. Ове средње вредности се уписују у осам двобајтних променљивих названих AN0MOD, AN1MOD, AN2MOD,..., AN7MOD. У наставку програма одређује се да ли су све ANxMOD температуре једнаке, и ако јесу пали се централна LED диода на $\frac{1}{2}$ секунде. Ово се може догодити ако уопште нема ветра, или ако постоји комбинација екстремно високе температуре (преко 45°C) уз врло слаб ветар. Уколико ANxMOD температуре нису једнаке одређује се која је највиша. Када је одређена највиша вредност, користећи порт В пали се LED диода која одговара централном сензору три сензорске комбинације. На пример, уколико програм одреди да је средња температура сензора за север, североисток и исток највиша упалиће се LED диода за североисток. Програм се затим враћа на поновно читавање свих осам сензора. У наставку дат је поједностављени дијаграм тока и листинг програма.

¹ Single chip mode



Слика 10.6 Дијаграм тока програма полупроводничких ветроказ

Дијаграм тока и листинг програма полупроводничких ветроказ**Програм 10.3 Полупроводнички ветроказ**

```

; ADRESE:
; koristi se interni RAM 0-$FF
; i interni EEPROM $B600-$B7FF

```

10. Примена микроконтролера серије MC68HC11

; LED1-LED8 kontroliše niži nibl PORTA B (\$1004)
; sledeći heksadecimalni brojevi uključuju
; respektivne LED diode; x znači nije bitno.
; sever--led1--\$x7
; si-----led2--\$x6
; istok---led3--\$x5
; ji-----led4--\$x4
; jug----led5--\$x3
; jz-----led6--\$x2
; zapad-led7--\$x1
; sz-----led8--\$x0

*Centralna LED dioda ukazuje na stanje kada nema vetra

*ili je temperatura suviše visoka ili niska

*CENT_LED----CENTER LED ----\$x8

PORTA	EQU	\$1000
PORTB	EQU	\$1004
PORTE	EQU	\$100A
T1C1	EQU	\$1010
TCTL2	EQU	\$1021
TFLG1	EQU	\$1023
ADCTL	EQU	\$1030
DATA	EQU	\$1031
ADR2	EQU	\$1032
ADR3	EQU	\$1033
ADR4	EQU	\$1034
OPTION	EQU	\$1039
COPRST	EQU	\$103A
PPROG	EQU	\$103B
HPRIO	EQU	\$103C
MT_LED1	EQU	\$10
AUTOADJ	EQU	\$01
N_LED	EQU	\$07
NE_LED	EQU	\$06
E_LED	EQU	\$05
SE_LED	EQU	\$04

S_LED	EQU	\$03
SW_LED	EQU	\$02
W_LED	EQU	\$01
NW_LED	EQU	\$00
CENT_LED	EQU	\$08
CLR_LEDS	EQU	\$09
	ORG	\$00
	RMB	50
STACK	RMB	1
TEMP1	RMB	40
NEXTD	RMB	1
AN0	RMB	1
AN1	RMB	1
AN2	RMB	1
AN3	RMB	1
AN4	RMB	1
AN5	RMB	1
AN6	RMB	1
AN7	RMB	1
AN0MOD	RMB	2
AN1MOD	RMB	2
AN2MOD	RMB	2
AN3MOD	RMB	2
AN4MOD	RMB	2
AN5MOD	RMB	2
AN6MOD	RMB	2
AN7MOD	RMB	2
ANN0	RMB	2
ANN1	RMB	2
ANN2	RMB	2
ANN3	RMB	2
ANN4	RMB	2
ANN5	RMB	2
ANN6	RMB	2
ANN7	RMB	2
BROJSRED	RMB	1

10. Примена микроконтролера серије MC68HC11

```
COUNT      RMB  1
AVGTH      RMB  1
AVGTL      RMB  1
BROJCIT    RMB  2
CUMTEMPH   RMB  1
CUMTEMPL   RMB  1
; Početak programa je na adresi $B600
          ORG  $B600
; Definisanje pokazivača stek memorije—veoma važno
LDS        #STACK
; Definisanje da se obavi 30 očitavanja senzora svaki put kada se pozove
; potprogram UZMITEMP
START      LDAA  #$30
          STAA  BROJCIT+1
; Najpre--palimo centralnu LED diodu i držimo upaljenu 1/2 sekunde--500 ms
          BSR   CENTRLED
          JSR   DLY500
; Proveravamo da li su sve ANxMOD temperature jednake ukoliko nisu
; izvršava se deo programa NACIVIS u kome se određuje najviša temperatura
NOVICIK    JSR   UZMITEMP
          LDAA  AN0MOD+1
          CMPA  AN1MOD+1
          BNE   NACIVIS
          CMPA  AN2MOD+1
          BNE   NACIVIS
          CMPA  AN3MOD+1
          BNE   NACIVIS
          CMPA  AN4MOD+1
          BNE   NACIVIS
          CMPA  AN5MOD+1
          BNE   NACIVIS
          CMPA  AN6MOD+1
          BNE   NACIVIS
          CMPA  AN7MOD+1
          BNE   NACIVIS
```

; Ukoliko su sve ANxMOD temperature jednake pali se centralna LED dioda
 ; i držimo upaljen 1/2 sekunde, zatim pokreće novo merenje

```

                                BSR  CENTRLED
                                JSR  DLY500
                                BRA  NOVICIK
CENTRLED  PSHB
                                LDAB #CENT_LED
                                STAB PORTB
                                PULB
                                RTS
NACIVIS   LDAB #N_LED
                                LDAA AN0MOD+1
                                CMPA AN1MOD+1

```

; Ako je AN1 veće nego AN0 to je novi podatak za poređenje

```

                                BHI  CKAN1
                                LDAB #NE_LED
                                LDAA AN1MOD+1
CKAN1     CMPA AN2MOD+1

```

; Ako je AN2 veće nego prethodni to je novi podatak za poređenje

```

                                BHI  CKAN2
                                LDAB #E_LED
                                LDAA AN2MOD+1
CKAN2     CMPA AN3MOD+1

```

; Ako je AN3 veće nego prethodni to je novi podatak za poređenje

```

                                BHI  CKAN3
                                LDAB #SE_LED
                                LDAA AN3MOD+1
CKAN3     CMPA AN4MOD+1

```

; Ako je AN4 veće nego prethodni to je novi podatak za poređenje

```

                                BHI  CKAN4
                                LDAB #S_LED
                                LDAA AN4MOD+1
CKAN4     CMPA AN5MOD+1

```

; Ako je AN5 veće nego prethodni to je novi podatak za poređenje


```

                                BHI    CKAN5
                                LDAB   #SW_LED
                                LDAA   AN5MOD+1
CKAN5                          CMPA   AN6MOD+1
; Ako je AN6 veće nego prethodni to je novi podatak za poređenje
                                BHI    CKAN6
                                LDAB   #W_LED
                                LDAA   AN6MOD+1
CKAN6                          CMPA   AN7MOD+1
; Ako je AN7 veće nego prethodni to je najveća vrednost
                                BHI    REPEAT
                                LDAB   #NW_LED
; U akumulatoru B je vrednost koja odgovara senzoru sa najvećom temperaturom
REPEAT                         STAB   PORTB
NEWJMP                         JMP    NOVIK
; Kraj programa   slede potprogrami
; Potprogram za proveru da li je gotova A/D konverzija
CCFCHK                         PSHA
CCFCHK1                        LDAA   ADCTL
                                ANDA   #%10000000
                                BEQ    CCFCHK1
                                PULA
                                RTS
; Potprogram za kašnjenje od 20 milisekundi za takt E=2MHz,to je 40k ciklusa
DLY20                          PSHX
                                LDX    #$6663
SUB1                           DEX
                                BNE    SUB1
                                PULX
                                RTS
; Kašnjenje od 1/2 sekunde
DLY500                         PSHB
                                LDAB   #$25
SUB2                           BSR    DLY20
                                DECB
                                BNE    SUB2

```

```

                                PULB
                                RTS
; Postavljanje A/D konvertora
UZMITEMP      LDX  #OPTION
                                BSET  0,X %11000000
; Briše NEXTD i puni indeks registar
                                CLR   NEXTD
                                LDX   #AN0
NEXTDATA      BSR   DLY20
NEXTDREG      LDAA  NEXTD
; Početak A/D konverzije
                                STAA  ADCTL
; Čeka kraj A/D konverzije
                                BSR   CCFCHK
                                PSHX
                                LDX   #TEMP1
                                PSHB
                                LDAB  BROJCIT+1
OK            STAB  BROJSRED
                                STAB  COUNT
                                CLR   CUMTEMPH
; Zapamti MSB akumulatora D
                                PSHA
AVG           LDAA  NEXTD
; Početak A/D ponovo -- i dalje ukazuje na AN0
                                STAA  ADCTL
                                JSR   CCFCHK
                                JSR   DLY20
                                LDAB  DATA
                                STAB  0,X
                                INX
                                DEC   BROJSRED
                                BNE   AVG
; Sada je 'BROJSRED' očitavanja smešteno na lokacije od TEMP1 dalje
; treba sabrati sve podatke i podeliti sa COUNT

```

```

; Najpre se briše akumulator D
      CLRA
      CLRB
; Briše se bit najveće težine (MSB) od BROJCIT
      STAA  BROJCIT
; Ukazuje se na prvi podatak
      LDX  #TEMP1
; Sabiranje
ADDAGAIN  ADDB  0,X
          STAB  CUMTEMPL
          BCC  NOCARRY
          INC  CUMTEMPH
NOCARRY   INX
          DEC  COUNT
          BNE  ADDAGAIN
; Sada CUMTEMP(H/L) sadrži sve akumulirane podatke sabrane 'COUNT' puta
          LDD  CUMTEMPH
; Load Index Register X sa BROJCIT-1
          LDX  #BROJCIT
          DEX
; Deljenje CUMTEMP sa (BROJCIT-1)
          IDIV
; INDEX Registar sadrži srednju vrednost
          STX  AVGTH
          PULA
          PULB
          PULX
; AVGTЛ sadrži srednju temperaturu
          LDAB AVGTЛ
; sada akumulator B sadrži srednju temperaturu za ANX
          STAB 0,X
          CPX  #AN7
          BEQ  AVG_OF_3
          INX
          INC  NEXTD

```

```

        BRA    NEXTDATA
; Postavi sve MSB od ANNX=0; smesti ANX na LSB od ANNX
AVG_OF_3    CLRA
            STAA  ANN0
            STAA  ANN1
            STAA  ANN2
            STAA  ANN3
            STAA  ANN4
            STAA  ANN5
            STAA  ANN6
            STAA  ANN7
            LDAA  AN0
            STAA  ANN0+1
            LDAA  AN1
            STAA  ANN1+1
            LDAA  AN2
            STAA  ANN2+1
            LDAA  AN3
            STAA  ANN3+1
            LDAA  AN4
            STAA  ANN4+1
            LDAA  AN5
            STAA  ANN5+1
            LDAA  AN6
            STAA  ANN6+1
            LDAA  AN7
            STAA  ANN7+1
; Saberi ANN0, ANN1, ANN2 i podeli sa 3; smesti rezultat na AN1MOD
            LDX   #3
            LDD   ANN0
            ADDD  ANN1
            ADDD  ANN2
            IDIV
            STX   AN1MOD
; Saberi ANN1, ANN2, ANN3 i podeli sa 3; smesti rezultat na AN2MOD
            LDX   #3

```

```

LDD  ANN1
ADDD ANN2
ADDD ANN3
IDIV
STX  AN2MOD
; Saberi ANN2, ANN3, ANN4 i podeli sa 3; smesti rezultat na AN3MOD
LDX  #3
LDD  ANN2
ADDD ANN3
ADDD ANN4
IDIV
STX  AN3MOD
; Saberi ANN3, ANN4, ANN5 i podeli sa 3; smesti rezultat na AN4MOD
LDX  #3
LDD  ANN3
ADDD ANN4
ADDD ANN5
IDIV
STX  AN4MOD
; Saberi ANN4, ANN5, ANN6 i podeli sa 3; smesti rezultat na AN5MOD
LDX  #3
LDD  ANN4
ADDD ANN5
ADDD ANN6
IDIV
STX  AN5MOD
; Saberi ANN5, ANN6, ANN7 i podeli sa 3; smesti rezultat na AN6MOD
LDX  #3
LDD  ANN5
ADDD ANN6
ADDD ANN7
IDIV
STX  AN6MOD
* Saberi ANN6, ANN7, ANN0 i podeli sa 3; smesti rezultat na AN7MOD
LDX  #3
LDD  ANN6

```

```

        ADDD ANN7
        ADDD ANN0
        IDIV
        STX  AN7MOD
* Saberi ANN7, ANN0, ANN1 i podeli sa 3; smesti rezultat na AN0MOD
        LDX  #3
        LDD  ANN7
        ADDD ANN0
        ADDD ANN1
        IDIV
        STX  AN0MOD
RETURN    RTS
        END

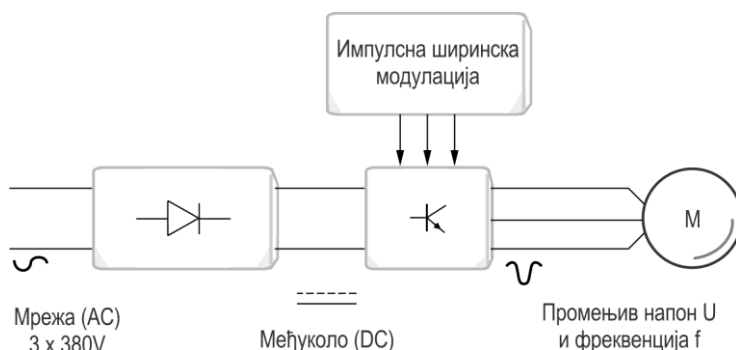
```

10.4 Једна реализација управљања трофазним асинхроним мотором

У примеру који следи описана је једна реализација управљања трофазним асинхроним мотором коришћењем микроконтролера MC68HC11. За контролу брзине обртања асинхроног мотора користе се инвертори. Једна од техника за регулацију фреквенције је импулсно-ширинска модулација (PWM¹). Инвертор претвара једносмерни напон у наизменични, а погодним избором периода дозволе провођења једносмерног напона добијају се правоугаони импулси одређене ширине и учестаности. Овом методом могуће је регулисати и вредност напона који ће се довести на мотор, јер мотор захтева константан обртни моменат, који зависи од односа квадрата напона и фреквенције. Употребом филтара може се издвојити синусоидни напон потребан за исправан рад асинхроног мотора.

Треба напоменути да се при коришћењу уређаја за промену брзине асинхроног мотора мора водити рачуна о односу напон/учестаност (V/Hz). Овај однос мора бити константан да не би дошло до прегоривања намотаја због превелике струје.

¹ Pulse Width Modulation



Слика 10.7 Блок шема фреквентне регулације трофазног асинхроног мотора

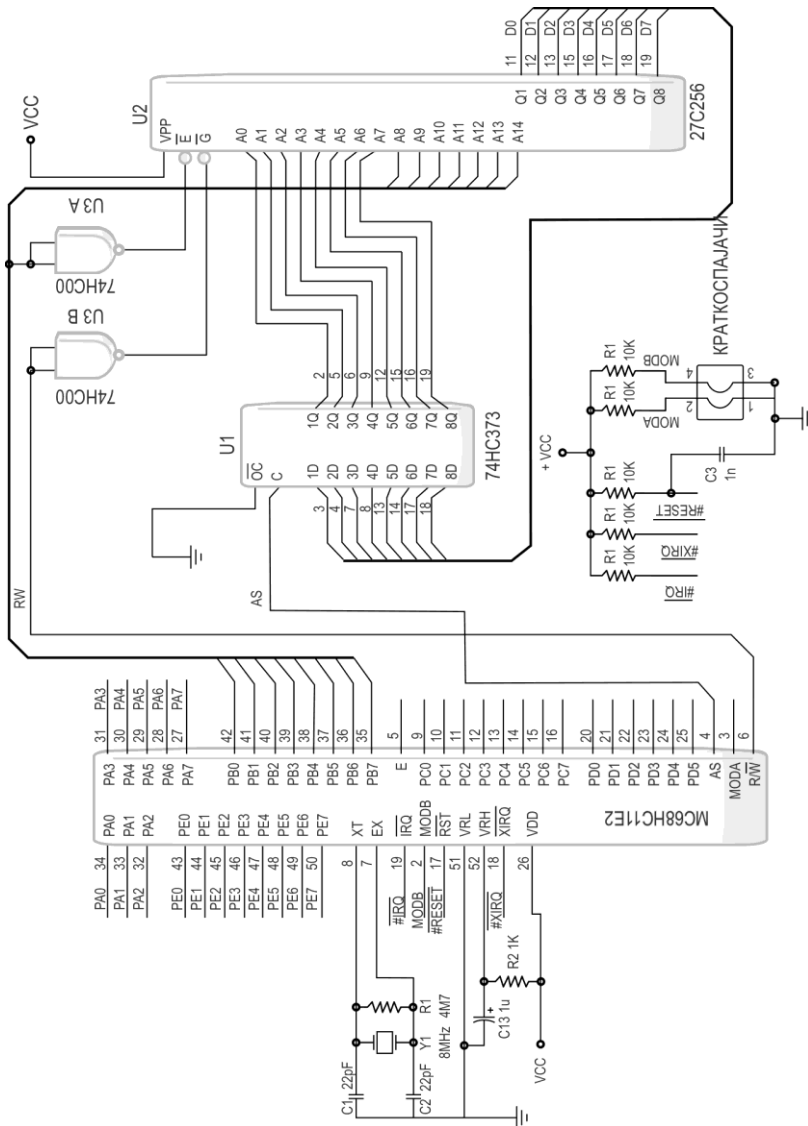
Као пример узмемо мотор 220V, 50Hz. Однос је 4,4. Ако овај мотор мора да ради са половином брзине (25Hz), онда напон мора да се смањи на 110V. Ако ово није урађено потећи ће двоструко већа струја и брзо уништити мотор. Овај однос напона и фреквенције мора да се прати при свим брзинама мотора.

На слици 10.7 приказана је блок шема фреквентне регулације трофазног асинхроног мотора. Блок означен са “импулсно-ширинска модулација” реализован је са микроконтролером MC68HC11 и треба да обезбеди генерисање три фазно померена синусна сигнала за угао од 120° променљиве учестаности од 0 до 50Hz. Поред тога треба обезбедити да количник максималне амплитуде и радне учестаности буде константа. Електрична шема процесорске плоче дата је на слици 10.8.

Да би програм могао да ради у реалном времену унапред су израчунате вредности амплитуде синусног сигнала у зависности од учестаности и смештене су у меморију. Како се ради о великом броју податка процесорској плочи је додата спољашња EPROM меморија (слика 10.8).

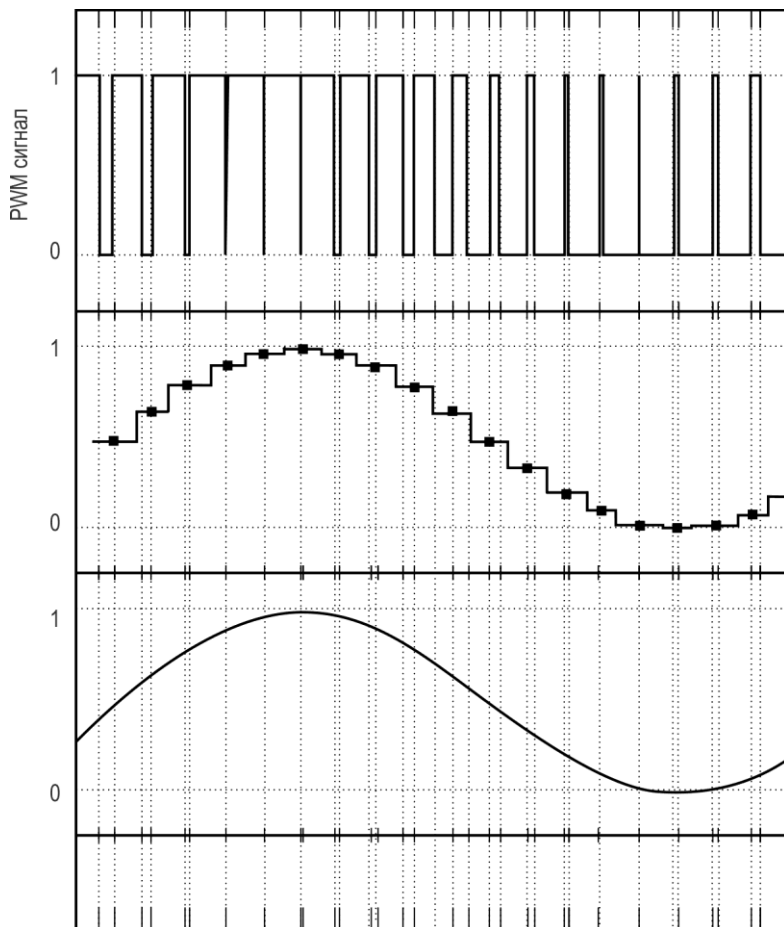
Генерисање синусног сигнала

На слици 10.9 приказан је принцип генерисања синусног сигнала коришћењем PWM модулације. За генерисање једне периоде синусног сигнала користи се 19 једнаких подинтервала у којима се генеришу



Слика 10.8. Електрична шема процесорске плоче

импульси различитих фактора испуне¹. Амплитуда сигнала је функција фактора испуне. Вредност фактора испуне за синусни сигнал може се израчунати коришћењем формуле $[1+\sin(\omega t)]/2$. Фактори испуне за 19 подинтервала за пример са слике 10.9 дати су табеларно (табела 10.4).



Слика 10.9 Генерисање синусног сигнала коришћењем PWM модулације

¹ Однос времена трајања јединице у односу на укупно време трајања подинтервала.

Редни број подинтервала	Фактор испуне	Трајање импулса у μs за синусни сигнал учестаности 50Hz
0	0.5	526
1	0.66	694
2	0.8	842
3	0.91	957
4	0.98	1031
5	0.99	1042
6	0.95	1000
7	0.86	905
8	0.73	768
9	0.58	610
10	0.41	431
11	0.26	273
12	0.13	136
13	0.04	42
14	0	0
15	0.01	10
16	0.08	84
17	0.19	200
18	0.33	347

Табела 10.4 Фактори испуне за 19 подинтервала за пример са слике 10.9

Трајање периоде сигнала израчунава се као производ броја подинтервала и времена трајања једног подинтервала. Претпоставимо да желимо да генеришемо синусни сигнал учестаности 50Hz. Време трајања једног подинтервала добија се тако што се време трајања периоде сигнала подели бројем подинтервала. У нашем примеру трајање једног подинтервала је $20\text{ms}/19$, односно $1,05\text{ ms}$. Да бисмо генерисали синусни сигнал потребно је сваких $1,05\text{ms}$ повећати редни број подинтервала за један и генерисати импулс чије је трајање дефинисано редним бројем подинтервала. Када редни број подинтервала постане већи од 18 поново се креће од редног броја 0.

Генерисање оваквог сигнала може се остварити применом микроконтролера MC68HC11. За генерисање се користи часовник и неки од регистара у које се уписује унапред одређена вредност са којом желимо да упоредимо тренутно стање часовника (компер регистар). За генерисање ћемо користити компер регистар OC5 коме је придружена линија PA3 порта A. Софтверски се мора дефинисати да линија PA3 има функцију излаза часовника. Стање излазне линије PA3 мења се када се вредност часовника изједначи са вредношћу компер регистра. На PA3 линију директан утицај има и регистар OC1 за упоређивање (компер) коме у иницијализацији периферија треба дозволити генерисање прекида. Постављањем одговарајуће вредности у OC1 регистар дефинише се време трајања подинтервала.

Претпоставимо да је такт сигнал са којим ради часовник $1\mu s$. Да бисмо остварили трајање подопсега од $1,05ms$ морамо у OC1 регистар уписати вредност која је за 1050 већа од тренутне вредности часовника. После 1050 периода такт сигнала вредност часовника ће се изједначити са вредношћу уписаном у регистар OC1 што ће изазвати извршавање потпрограма за обраду прекида и постављање PA3 на активан ниво.

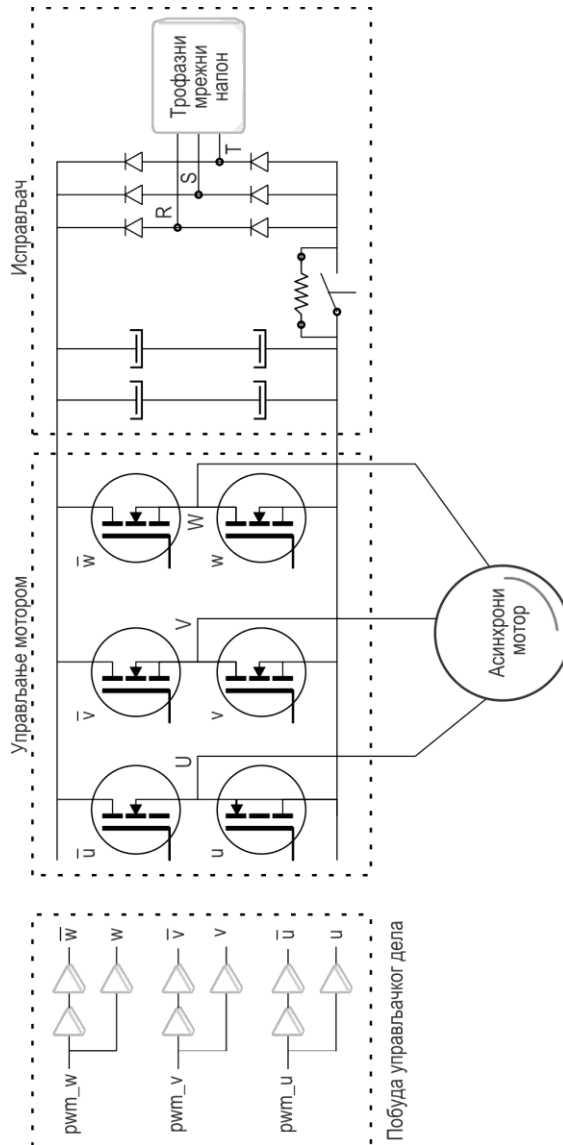
У потпрограму за обраду прекида треба реализовати програм који обезбеђује кружно читавање вредности времена трајања импулса из меморије (времена трајања дата су у табели 10.4), додавање тренутне вредности часовника на очитане вредности и смештање добијеног збира у OC5 регистар.

Када се вредност часовника изједначи са вредношћу OC5 регистра PA3 линија се пребацује на ниво логичке 0.

Користећи прекид и унапред израчунате вредности обезбеђује се рад у реалном времену.

Претпоставимо да желимо да генеришемо синусни сигнал двоструко мање учестаности (25Hz). Алгоритам који је описан не би се мењао, једино би се предефинисане вредности за трајање подинтервала и трајање импулса помножиле са 2.

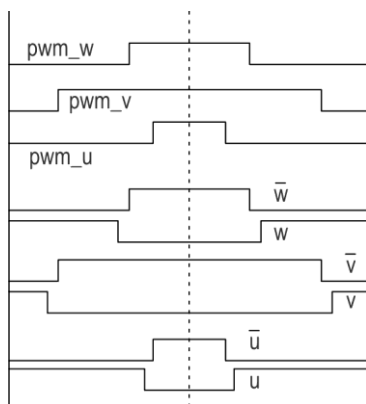
Микроконтролер треба да обезбеди три линије на којима се генеришу сигнали чија се учестаност може мењати у опсегу од 0 до 50Hz, фактор испуне од 0 до 100% и фазни помак између генерисаних сигнала од 120° . За генерисање оваквих сигнала користе се часовник и регистри поређења OC3,



Слика 10.10 Енергетски модул за побуду трофазног асинхронног мотора

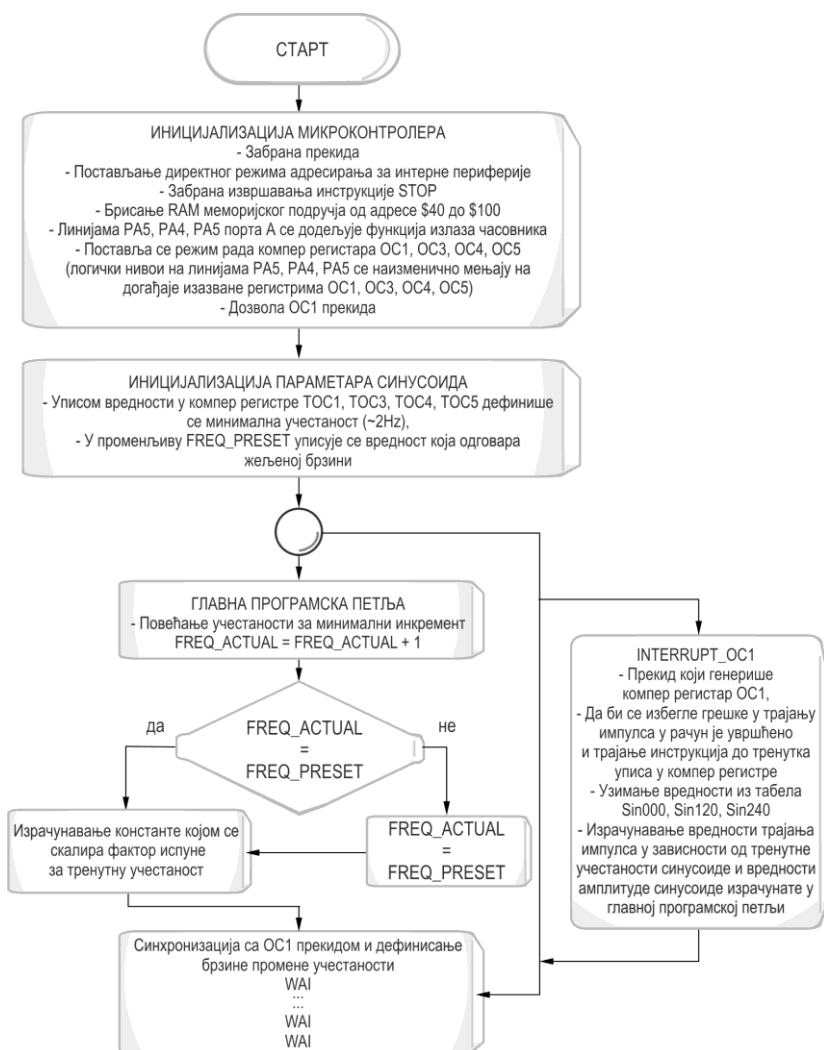
OC4, OC5 којима су придружене линије порта A PA5, PA4, PA3 респективно. Софтверски се мора дефинисати да ови прикључци имају функцију излаза часовника. Стање излазних линија мења се када се вредност одговарајућег компер регистра изједначи са вредношћу часовника. На све три линије директан утицај има компер регистар OC1. Постављањем одговарајуће вредности у OC1 регистар дефинише се учестаност сва три сигнала. Постављене вредности у регистре OC3, OC4 и OC5 дефинишу амплитуде и фазни помак између сигнала генерисаних на линијама PA5, PA4, PA3. Вредности које треба сместити у регистре OC3, OC4 и OC5 одговарају амплитудама синусног сигнала и не би могле да се израчунају у реалном времену. Због тога су оне унапред израчунате и смештене табеларно у EPROM (27C256) меморију. Очитавају се коришћењем индексног адресирања. Линије PA5, PA4, PA3 доводе се на улазе `pwm_w`, `pwm_v`, `pwm_u` енергетског модула за побуду трофазног асинхроног мотора приказаног на слици 10.10.

На слици 10.11 приказани су таласни облици сигнала које треба генерисати на линијама PA5, PA4, PA3. На процесорској плочи налази се RS232 интерфејс коло. Коришћењем серијске комуникације може се остварити комуникација и управљање посредством персоналног рачунара. Дијаграм тока (слика 10.12) и програм за генерисање таласних облика сигнала потребних за управљање брзином трофазног асинхроног мотора дати су у наставку овог одељка.



Слика 10.11 Таласни облици сигнала

Дијаграм тока и листинг програма за управљање трофазним асинхроним мотором



Слика 10.12 Дијаграм тока за програм 10.4

Програм 10.4 Програм за управљање трофазним асинхроним мотором

```

; Izlazna frekvencija je u opsegu od 0Hz do 50Hz
; Tip programske memorije je EPROM memorija 27C256
PORTA      EQU   $0000
TCNT       EQU   $000E
TOC1       EQU   $0016
TOC2       EQU   $0018
TOC3       EQU   $001A
TOC4       EQU   $001C
TOC5       EQU   $001E
TCTL1      EQU   $0020
TMSK1      EQU   $0022
TFLG1      EQU   $0023
OPTION     EQU   $0039
INIT       EQU   $003D
;Promenjive su smeštene u RAM području
          ORG   $0040

; 5000 => 50 Hz
FREQ_PRESET RMB   2
; od 0 do 5000 => 0Hz do 50 Hz
FREQ_ACTUAL RMB   2
; Napon na motoru
V_MOT      RMB   1
; Pomoćna promenljiva
PWM        RMB   2
; Pokazivač na tabelu sinusnih vrednosti
PTR_SINUS  RMB   4
; Pokazivač se uvećava za ovu vrednost
PTR_ADD    RMB   2
; Pomoćna promenljiva
TEMP      RMB   2
; Program je smešten u EPROM područje
          ORG   $8000

```

```
;Inicijalizacija mikrokontrolera i RAM memorije
START      CLR   INIT+$1000
           SEI
           CLR   TMSK1
; Dozvoljava nadgledanje takta A/D konvertora
           LDAA  #$9B
; Postavlja kašnjenje registra za upoređivanje (COP)
           STAA  OPTION
           TPA
; Zabranjena STOP instrukcija
           ORAA  #$80
           TAP
; Postavljanje pokazivača stek memorije na vrednost $FF
           LDS   #$FF
; Brisanje RAM memorije
           LDX   #$40
CLEARRAM   CLR   0,X
           INX
           CPX   #$0100
           BLO   CLEARRAM
           LDAA  PORTA
           ANDA  #%11000111
           STAA  PORTA
; Naizmenični komper režim rada
           LDAA  #$15
; Postavljanje OC3, OC4 и OC5 u TCTL1 u aktivno stanje
           STAA  TCTL1
; Postavljanje vrednosti OC1 (prekid je omogućen) u registru TMSK1
           LDAA  #$80
           STAA  TMSK1
; Postavljanje vrednosti registra za upoređivanje
           LDD   #$FF69
           STD   TOC1
           LDD   #$FF40
           STD   TOC3
```



```

        STD  TOC4
        STD  TOC5
        LDD  #SIN000
        STD  PTR_SINUS
        LDAA #$90
        STAA V_MOT
; Početak rada sa 5000 => 50 Hz
        LDD  #$5000
        STD  FREQ_PRESET
        LDAA TCNT
        ADDA #$12
        STAA TOC1
        CLI
        WAI
; Glavna programska petlja
; Blok I: vreme trajanja 7x128 µsec = 896 µsec
PWM3_LOOP  WAI
            LDD  FREQ_ACTUAL
            ADDD #1
            STD  FREQ_ACTUAL
            CPD  FREQ_PRESET
            BLT  RAMP_OK
            LDD  FREQ_PRESET
            STD  FREQ_ACTUAL
RAMP_OK    WAI
            LDD  FREQ_ACTUAL
            LDX  #$4200
            IDIV
            STX  TEMP
            WAI
            LDX  #$4200
            FDIV
            WAI
            XGDX
            TAB
            LDAA TEMP+1

```

```

                                STD  PTR_ADD
                                WAI
                                WAI
                                WAI
;Blok II: vreme trajanja 7x128 µsec = 896 µsec
                                WAI
                                WAI
                                LDD  FREQ_ACTUAL
                                CPD  #$4200
                                BLO  DIV_OKAY
                                WAI
; Maksimalna modulacija (faktor ispune PWM signala 100%)
                                LDAB #$255
                                BRA  UF_OK
DIV_OKAY  LDX  #$4200
                                WAI
                                FDIV
                                XGDX
                                TAB
UF_OK     WAI
                                LDX  #UF_TABLE
                                ABX
                                LDAB 0, X
                                SUBB #$10
                                BCC  SAVE_V_MOT
                                CLR B
SAVE_V_MOT CLR B
                                STAB V_MOT
                                WAI
; Blok III: vreme trajanja 7x128µsec = 896 µsec
                                WAI
                                WAI
                                WAI
                                WAI
                                WAI
```

```

WAI
WAI
JMP  PWM3_LOOP

;Prekidna rutina
; Za svaku granu unutar prekidne rutine izbalansirano je vreme trajanja
INTERRUPT_OC1 NOP
; Izračunavanje parametara faze W
LDAA  TOC3+1
LDAB  TOC4+1
NEGA
NEGB
STAA  TOC3+1
STAB  TOC4+1
NEG  TOC5+1
NOP
NOP
NOP
NOP
NOP
LDAA  TOC3+1
LDY   PTR_SINUS
LDX   PTR_SINUS
CBA
BCS  SETPWM_1
CMPB #$222
BCC  SETPWM_2
LDD  PTR_SINUS+1
ADDD PTR_ADD
STAB PTR_SINUS+2
LDAB #$186
STAB TFLG1
CMPA #$10
BCS  PR1
STAA PTR_SINUS+1

```

```
PR1      LDAB  V_MOT
          ABX
          LDAA  TOC1
          INCA
          STAA  TOC1
          LDAB  0,X
          ADDB  0,X
          LSRB
          STD   PWM
          LDAB  !62,Y
          ADDB  !62,X
          LSRB
          STD   TOC4
          LDAB  !124,Y
          ADDB  !124,X
          LSRB
          LDX   PWM
          STD   TOC5
          STX   TOC3
          RTI
;Izračunavanje parametara faze napona U
SETPWM_1  CMPA  #$222
          BCC   SETPWM_2
          LDD   PTR_SINUS+1
          ADDD  PTR_ADD
          STAB  PTR_SINUS+2
          LDAB  #$186
          STAB  TFLG1
          CMPA  #$10
          BCS   PR2
          STAA  PTR_SINUS+1
PR2      LDAB  V_MOT
```

```

ABX
LDAA TOC1
INCA
STAA TOC1
LDAB !124,Y
ADDB !124, X
LSRB
STD PWM
LDAB 0,Y
ADDB 0, X
LSRB
STD TOC3
LDAB 62,Y
ADDB 62, X
LSRB
LDX PWM
STD TOC4
STX TOC5
RTI
SETPWM_2 LDD PTR_SINUS+1
          ADDD PTR_ADD
          STAB PTR_SINUS+2
          LDAB #$186
          STAB TFLG1
          CMPA #$10
          BCS PR3
          STAA PTR_SINUS+1
PR3       LDAB V_MOT
          ABX
          LDAA TOC1
          INCA
          STAA TOC1

```

LDAB 62,Y
 ADDB 62,X
 LSRB
 STD PWM
 LDAB 124,Y
 ADDB 124,X
 LSRB
 STD TOC5
 LDAB 0,Y
 ADDB 0,X
 LSRB
 LDX PWM
 STD TOC3
 STX TOC4
 RTI

; Tabela diskretnih vrednosti sisnusoidea pomerenih za 120 stepeni

; Prva adresa tabele mora biti xx00

	ORG	\$F800
Sin000	FCB	064,067,071,074,078,081,084,087,090,093
	FCB	096,099,101,104,106,108,110,112,113,115
	FCB	116,118,119,119,120,121,121,122,122,122
	FCB	122,122,122,122,122,122,122,122,121,121
	FCB	121,121,121,120,120,120,120,120,120,120
	FCB	120,121,121,121,121,121,122,122,122,122
	FCB	122,122
Sin120	FCB	122,122,122,122,122,121,121,120,119,119
	FCB	118,116,115,113,112,110,108,106,104,101
	FCB	099,096,093,090,087,084,081,078,074,071
	FCB	067,064,061,057,054,050,047,044,041,038
	FCB	035,032,029,027,024,022,020,018,016,015
	FCB	013,012,010,009,009,008,007,007,006,006
	FCB	006,006

10. Примена микроконтролера серије MC68HC11

Sin240	FCB	006,006,006,006,006,006,006,007,007,007
	FCB	007,007,008,008,008,008,008,008,008,008
	FCB	007,007,007,007,007,006,006,006,006,006
	FCB	006,006,006,006,006,006,007,007,008,009
	FCB	009,010,012,013,015,016,018,020,022,024
	FCB	027,029,032,035,038,041,044,047,050,054
	FCB	057,061
Sin360	FCB	064,067,071,074,078,081,084,087,090,093
	FCB	096,099,101,104,106,108,110,112,113,115
	FCB	116,118,119,119,120,121,121,122,122,122
	FCB	122,122,122,122,122,122,122,122,121,121
	FCB	11,121,121,120,120,120,120,120,120,120
	FCB	120,121,121,121,121,121,122,122,122,122
	FCB	122,122
	FCB	122,122,122,122,122,121,121,120,119,119
	FCB	118,116,115,113,112,110,108,106,104,101
	FCB	099,096,093,090,087,084,081,078,074,071
	FCB	067,064,061,057,054,050,047,044,041,038
	FCB	035,032,029,027,024,022,020,018,016,015
	FCB	013,012,010,009,009,008,007,007,006,006
	FCB	006,006
	FCB	006,006,006,006,006,006,006,007,007,007
	FCB	007,007,008,008,008,008,008,008,008,008
	FCB	007,007,007,007,007,006,006,006,006,006
	FCB	006,006,006,006,006,006,007,007,008,009
	FCB	009,010,012,013,015,016,018,020,022,024
	FCB	027,029,032,035,038,041,044,047,050,054
	FCB	057,061
	FCB	064,067,071,074,078,081,084,087,090,093
	FCB	096,099,101,104,106,108,110,112,113,115
	FCB	116,118,119,119,120,121,121,122,122,122

FCB 122,122,122,122,122,122,122,121,121
 FCB 121,121,121,120,120,120,120,120,120
 FCB 120,121,121,121,121,121,122,122,122
 FCB 122,122

; Tabela vrednosti napona sinusoide u funkciji učestanosti

	ORG	\$FC00
UF_TABLE	FCB	093,093,093,092,092,092,091,091,091,091
	FCB	090,090,090,090,089,089,089,088,088
	FCB	088,088,087,087,087,087,086,086,086,086
	FCB	085,085,085,085,084,084,084,084,083,083
	FCB	083,083,082,082,082,082,081,081,081,081 ; 5
	FCB	080,080,080,080,079,079,079,079,078,078 ; 6
	FCB	078,078,077,077,077,076,076,076,076,075 ; 7
	FCB	075,075,075,074,074,074,074,073,073,073 ; 8
	FCB	073,072,072,072,071,071,071,071,070,070 ; 9
	FCB	070,070,069,069,069,068,068,068,068,067 ;10
	FCB	067,067,067,066,066,066,065,065,065,065 ;11
	FCB	064,064,064,063,063,063,063,062,062,062 ;12
	FCB	061,061,061,061,060,060,060,059,059,059 ;13
	FCB	058,058,058,058,057,057,057,056,056,056 ;14
	FCB	055,055,055,054,054,054,054,053,053,053 ;15
	FCB	052,052,052,051,051,051,050,050,050,049 ;16
	FCB	049,049,048,048,048,047,047,046,046,046 ;17
	FCB	045,045,045,044,044,044,043,043,042,042 ;18
	FCB	042,041,041,040,040,040,039,039,038,038 ;19
	FCB	038,037,037,036,036,035,035,035,034,034 ;20
	FCB	033,033,032,032,031,031,030,030,029,029 ;21
	FCB	028,028,027,026,026,025,025,024,023,023 ;22
	FCB	022,021,021,020,019,018,017,016,015,014 ;23
	FCB	013,012,011,009,007,004,000,000,000,000 ;24
	FCB	000,000,000,000,000,000,000,000,000,000 ;25

10. Примена микроконтролера серије MC68HC11

```
FCB  000,000,000,000,000,000
ORG  $FFE8
FDB  INTERRUPT_OC1
ORG  $FFFE
FDB  START
```

11. Комплетан скуп инструкција MC68HC11

У овом поглављу биће приказан преглед свих инструкција MC68HC11.
За сваку инструкцију, биће дат :

- текстуални опис;
- симболичка ознака (мнемоник);
- табела која показује симболичку ознаку инструкције, начин адресирања, операциони код, број такт циклуса потребних за извршавање инструкције, број бајтова операционог кода и
- ознаке како инструкција утиче на различите битове регистра услова.

У даљем тексту је листа ознака које се користе у овом поглављу.

а) Оператори

- [] = садржај регистра приказан је између заграда
- ← = је пренет у
- ↑ = скинут (очитан) из стек меморије
- ↓ = гурнут (уписан) у стек меморију
- = логичко И
- ⊕ = аритметичка операција сабирања
- ⊕ = ексклузивно ИЛИ
- * = операција множења
- :
- = аритметичка операција одузимања

б) Регистри

- A = акумулатор А
- B = акумулатор В
- A_x = акумулатор А или В

D	=	двоструки акумулатор D – акумулатор A је здружен са акумулатором B, а бајт веће тежине се налази у акумулатору A
CCR	=	регистар услова
IX	=	индекс регистар X, 16 битова
IXH	=	виши бајт индекс регистра X
IXL	=	нижи бајт индекс регистра X
	=	индекс регистар Y, 16 битова
IYH	=	виши бајт индекс регистра Y
IYL	=	нижи бајт индекс регистра Y
PC	=	програмски бројач, 16 битова
PCH	=	виши бајт програмског бројача
PCL	=	нижи бајт програмског бројача
SP	=	показивач стек меморије, 16 битова
SPH	=	виши бајт показивача стек меморије
SPL	=	нижи бајт показивача стек меморије

в) Меморија и адресирање

M	=	меморијска локација која је дата адресом операнда, 1 бајт
M+1	=	меморијска локација која је дата адресом операнда + 1
Rel	=	релативно одступање
(opr)	=	операнд
(msk)	=	маска која се користи у инструкцијама које раде на нивоу једног бита
(rel)	=	померај који се користи код инструкција гранања

г) Битови (флегови) регистра услова

S	=	бит (флег) који онемогућава извршавање STOP инструкције (б ₇)
X	=	бит (флег) забране немаскирајућег прекида (б ₆)
H	=	бит (флег) половине преноса (б ₅)
I	=	бит (флег) маскирања прекида (б ₄)
N	=	бит (флег) који указује да је резултат операције негативни број (б ₃)
Z	=	бит (флег) који указује да је резултат операције нула (б ₂)
V	=	бит (флег) који указује да је резултат операције ван опсега (б ₁)
C	=	бит (флег) преноса (б ₀)

д) Листа симбола који се користе са битовима

- = операција нема утицаја на битове регистра услова
- 0 = бит је постављен на 0
- 1 = бит је постављен на 1
- Δ = бит је обрисан или постављен у складу са резултатом операције
- $\downarrow$ = бит може променити стање са 1 на 0, или остати на 0 или 1 као резултат операције, али не може променити стање са 0 на 1

ABA – Add Accumulator B to Accumulator A

Опис: Додаје садржај акумулатора В садржају акумулатора А и смешта резултат у акумулатор А. Садржај акумулатора В остаје непромењен.

Операција:

$$A \leftarrow [A] + [B]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
ABA	Имплицитно	1B	1	2	-	-	Δ	-	Δ	Δ	Δ	Δ

ABX – Add Accumulator B to Index Register X

Опис: Додаје 8-битни садржај акумулатора В садржају индекса регистра X (IX), узимајући у обзир могући пренос из нижег бајта индекса регистра. Резултат се смешта у индекс регистар X (IX). Садржај акумулатора В остаје непромењен. Не постоји еквивалентна инструкција која додаје садржај акумулатора А индекс регистру X.

Операција:

$$IX \leftarrow [IX] + [B]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
ABX	Имплицитно	3A	1	3	-	-	-	-	-	-	-	-

ABY – Add Accumulator B to Index Register Y

Опис: Додаје 8-битни садржај акумулатора В садржају индекса регистра Y (IY), узимајући у обзир могући пренос из нижег бајта индекса регистра.

11. Комплетан скуп инструкција MC68HC11

Резултат се смешта у индекс регистар Y (IY). Садржај акумулатора В остаје непромењен. Не постоји еквивалентна инструкција која додаје садржај акумулатора А индекс регистру Y.

Операција:

$$IY \leftarrow [IY] + [B]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
ABY	Имплицитно	18 3A	2	4	-	-	-	-	-	-	-	-

ADC – Add with Carry

Опис: Вредност С бита регистра услова додаје се на збир садржаја акумулатора A_x и садржаја меморијске локације М. Резултат се смешта у акумулатор A_x .

Операција:

$$A_x \leftarrow [A_x] + [M] + [C]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
ADCA (opr)	Непосредно	89	2	2	-	-	Δ	-	Δ	Δ	Δ	Δ
	Директно	99	2	3	-	-	Δ	-	Δ	Δ	Δ	Δ
	Проширено	B9	3	4	-	-	Δ	-	Δ	Δ	Δ	Δ
	Индексно X	A9	2	4	-	-	Δ	-	Δ	Δ	Δ	Δ
ADCB (opr)	Непосредно	C9	2	2	-	-	Δ	-	Δ	Δ	Δ	Δ
	Директно	D9	2	3	-	-	Δ	-	Δ	Δ	Δ	Δ
	Проширено	F9	3	4	-	-	Δ	-	Δ	Δ	Δ	Δ
	Индексно X	E9	2	4	-	-	Δ	-	Δ	Δ	Δ	Δ
	Индексно Y	18 E9	3	5	-	-	Δ	-	Δ	Δ	Δ	Δ

ADD – Add without Carry

Опис: Додаје садржај меморијске локације М садржају акумулатора A_x и смешта резултат у акумулатор A_x .

Операција:

$$A_x \leftarrow [A_x] + [M]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
ADDA (opr)	Непосредно	8B	2	2	-	-	Δ	-	Δ	Δ	Δ	Δ
	Директно	9B	2	3	-	-	Δ	-	Δ	Δ	Δ	Δ
	Проширено	BB	3	4	-	-	Δ	-	Δ	Δ	Δ	Δ
	Индексно X	AB	2	4	-	-	Δ	-	Δ	Δ	Δ	Δ
	Индексно Y	18 AB	3	5	-	-	Δ	-	Δ	Δ	Δ	Δ
ADDB (opr)	Непосредно	CB	2	2	-	-	Δ	-	Δ	Δ	Δ	Δ
	Директно	DB	2	3	-	-	Δ	-	Δ	Δ	Δ	Δ
	Проширено	FB	3	4	-	-	Δ	-	Δ	Δ	Δ	Δ
	Индексно X	EB	2	4	-	-	Δ	-	Δ	Δ	Δ	Δ
	Индексно Y	18 EB	3	5	-	-	Δ	-	Δ	Δ	Δ	Δ

ADDD – Add Double Accumulator

Опис: Додаје шеснаестобитни садржај меморијских локација M (виши бајт) и M+1 (нижи бајт) садржају акумулатора D и смешта резултат у акумулатор D. У акумулатор A се смешта виши бајт 16-битног акумулатора D.

Операција:

$$D \leftarrow [D] + [M : M+1]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
ADDD (opr)	Непосредно	C3	3	4	-	-	-	-	Δ	Δ	Δ	Δ
	Директно	D3	2	5	-	-	-	-	Δ	Δ	Δ	Δ
	Проширено	F3	3	6	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно X	E3	2	6	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно Y	18 E3	3	7	-	-	-	-	Δ	Δ	Δ	Δ

AND – Logical AND

Опис: Извршава се логичка И операција између садржаја акумулатора A_x и садржаја меморијске локације M. Резултат се смешта у акумулатор A_x .

Операција:

$$A_x \leftarrow [A_x] \bullet [M]$$

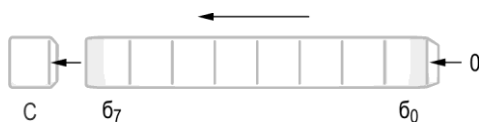
11. Комплетан скуп инструкција MC68HC11

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
ANDA (opr)	Непосредно	84	2	2	-	-	-	-	Δ	Δ	0	-
	Директно	94	2	3	-	-	-	-	Δ	Δ	0	-
	Проширено	B4	3	4	-	-	-	-	Δ	Δ	0	-
	Индексно X	A4	2	4	-	-	-	-	Δ	Δ	0	-
	Индексно Y	18 A4	3	5	-	-	-	-	Δ	Δ	0	-
ANDB (opr)	Непосредно	C4	2	2	-	-	-	-	Δ	Δ	0	-
	Директно	D4	2	3	-	-	-	-	Δ	Δ	0	-
	Проширено	F4	3	4	-	-	-	-	Δ	Δ	0	-
	Индексно X	E4	2	4	-	-	-	-	Δ	Δ	0	-
	Индексно Y	18 E4	3	5	-	-	-	-	Δ	Δ	0	-

ASL – Arithmetic Shift Left (исто као LSL)

Опис: Инструкција врши померање свих битава акумулатора A_x или меморијске локације М једно место улево. Бит са нулте позиције (b_0) поставља се на 0. У С флег регистра услова смешта се бит највеће тежине (b_7) акумулатора A_x или садржаја меморијске локације М.

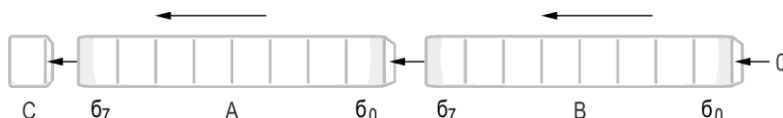
Операција:



Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
ASL (opr)	Проширено	78	3	6	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно X	68	2	6	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно Y	18 68	3	7	-	-	-	-	Δ	Δ	Δ	Δ
ASLA	Имплицитно	48	1	2	-	-	-	-	Δ	Δ	Δ	Δ
ASLB	Имплицитно	58	1	2	-	-	-	-	Δ	Δ	Δ	Δ

ASLD – Arithmetic Shift Left Double Accumulator (исто као LSLD)

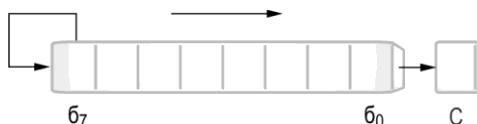
Опис: Инструкција врши померање свих битава акумулатора D једно место улево. Бит са нулте позиције (b_0) поставља се на 0. У С флег регистра услова смешта се бит највеће тежине (b_{15}) акумулатора D.

Операција:

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
ASLD	Имплицитно	05	1	3	-	-	-	-	Δ	Δ	Δ	Δ

ASR – Arithmetic Shift Right

Опис: Инструкција врши померање свих битова акумулатора A_x или меморијске локације M за једно место удесно. У C бит регистра услова смешта се бит најмање тежине (b_0) A_x или садржаја меморијске локације M .

Операција:

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
ASR (opr)	Проширено	77	3	6	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно X	67	2	6	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно Y	18 67	3	7	-	-	-	-	Δ	Δ	Δ	Δ
ASRA	Имплицитно	47	1	2	-	-	-	-	Δ	Δ	Δ	Δ
ASRB	Имплицитно	57	1	2	-	-	-	-	Δ	Δ	Δ	Δ

За бајт података 10110101 биће:

	Подаци из меморије	C бит
Пре ASR	10110101	Δ
После ASR	11011010	1

11. Комплетан скуп инструкција MC68HC11

BCC – *Branch if Carry Clear* (исто као **BHS**)

Опис: Тестира се садржај С бита регистра услова и уколико је садржај једнак нули долази до гранања.

Операција:

$$PC \leftarrow [PC] + \$0002 + Rel \quad \text{If } (C) = 0$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
BCC (rel)	Релативно	24	2	3	-	-	-	-	-	-	-	-

BCLR – *Clear Bit(s) in Memory*

Опис: Брише битове меморијске локације М. Битови које треба обрисати у масци се означавају са 1. Сви остали битови меморијске локације М не мењају своју вредност.

Операција:

$$M \leftarrow [M] \bullet [PC + 2]$$

$$M \leftarrow [M] \bullet [PC + 3] \text{ (за индексно, Y регистар)}$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
BCLR (opr)	Директно	15	3	6	-	-	-	-	Δ	Δ	0	-
(msk)	Индексно X	1D	3	7	-	-	-	-	Δ	Δ	0	-
	Индексно Y	18 1D	4	8	-	-	-	-	Δ	Δ	0	-

BCS – *Branch if Carry Set* (исто као **BLO**)

Опис: Тестира се садржај С бита регистра услова, и уколико је садржај једнак јединици долази до гранања.

Операција:

$$PC \leftarrow [PC] + \$0002 + Rel \quad \text{If } (C) = 1$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
BCS (rel)	Релативно	25	2	3	-	-	-	-	-	-	-	-

BEQ – *Branch if Equal to Zero*

Опис: Тестира се садржај Z бита регистра услова, и уколико је садржај једнак јединици долази до гранања.

Операција:

$$PC \leftarrow [PC] + \$0002 + Rel \quad \text{If } (Z) = 1$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	0	1	2	3	4	5	6	7
BEQ (rel)	Релативно	27	2	3	-	-	-	-	-	-	-	-

BGE – Branch if Grater or Equal to Zero

Опис: Ако се BGE инструкција извршава одмах након извршавања било које од следећих инструкција, CBA, CMP (акумулатори A, B, или D), CP (индекс регистра X или Y), SBA, SUB (акумулатори A, B, или D), до гранања ће доћи само ако је означени број у акумулатору A_x већи или једнак означеном броју са меморијске локације M.

Операција:

$$PC \leftarrow [PC] + \$0002 + Rel \quad \text{If } \{(N) \oplus (V)\} = 1$$

тј. If $[A_x] \geq [M]$ (други комплемент - бројеви са знаком)

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	0	1	2	3	4	5	6	7
BGE (rel)	Релативно	2C	2	3	-	-	-	-	-	-	-	-

BGT – Branch if Grater than Zero

Опис: Ако се BGT инструкција извршава одмах након извршавања било које од следећих инструкција, CBA, CMP (акумулатори A, B, или D), CP (индекс регистра X или Y), SBA, SUB (акумулатори A, B, или D), до гранања ће доћи само ако је означени број у акумулатору A_x већи од означеног броја са меморијске локације M.

Операција:

$$PC \leftarrow [PC] + \$0002 + Rel \quad \text{If } (Z) + \{(N) \oplus (V)\} = 0$$

тј. If $[A_x] > [M]$ (други комплемент бројева са знаком)

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	0	1	2	3	4	5	6	7
BGT (rel)	Релативно	2E	2	3	-	-	-	-	-	-	-	-

BHI – Branch if Higher

Опис: Ако се BHI инструкција извршава одмах након извршавања било које од следећих инструкција, CBA, CMP (акумулатори A, B, или D), CP (индекс регистри X или Y), SBA, SUB (акумулатори A, B, или D), гранање ће се извршити само ако је бинарни број без знака у акумулатору A_x већи од бинарног броја без знака са меморијске локације M.

Операција:

$$PC \leftarrow [PC] + \$0002 + Rel \quad \text{If } \{(C) + (Z)\} = 0$$

$$\text{If } [A_x] > [M] \quad (\text{бинарни бројеви без знака})$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
BHI (rel)	Релативно	22	2	3	-	-	-	-	-	-	-	-

BHS – Branch if Higher or Same (исто као BCC)

Опис: Ако се BHS инструкција извршава одмах након извршавања било које од следећих инструкција, CBA, CMP (акумулатори A, B, или D), CP (индекс регистри X или Y), SBA, SUB (акумулатори A, B, или D), до гранања ће доћи само ако је бинарни број без знака у акумулатору A_x већи или једнак бинарном броју без знака са меморијске локације M.

Операција:

$$PC \leftarrow [PC] + \$0002 + Rel \quad \text{If } (C) = 0$$

$$\text{If } [A_x] \geq [M] \quad (\text{бинарни бројеви без знака})$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
BHS (rel)	Релативно	24	2	3	-	-	-	-	-	-	-	-

BIT – Bit Test

Опис: Логичка И операција изводи се бит по бит, између садржаја меморијске локације M и акумулатора AX. Резултат утиче само на Z и N битове. Садржај акумулатора и меморијске локације остаје неизмењен.

Операција:

$$[A_x] \bullet [M]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
BITA (opr)	Непосредно	85	2	2	-	-	-	-	Δ	Δ	0	-
	Директно	95	2	3	-	-	-	-	Δ	Δ	0	-
	Проширено	B5	3	4	-	-	-	-	Δ	Δ	0	-
	Индексно X	A5	2	4	-	-	-	-	Δ	Δ	0	-
	Индексно Y	18 A5	3	5	-	-	-	-	Δ	Δ	0	-
BITB (opr)	Непосредно	C5	2	2	-	-	-	-	Δ	Δ	0	-
	Директно	D5	2	3	-	-	-	-	Δ	Δ	0	-
	Проширено	F5	3	4	-	-	-	-	Δ	Δ	0	-
	Индексно X	E5	2	4	-	-	-	-	Δ	Δ	0	-
	Индексно Y	18 E5	3	5	-	-	-	-	Δ	Δ	0	-

BLE – Branch if Less or Equal to Zero

Опис: Ако се BLE инструкција извршава одмах након извршавања било које од следећих инструкција, CBA, CMP (акумулатори A, B, или D), CP (индекс регистри X или Y), SBA, SUB (акумулатори A, B, или D), до гранања ће доћи само ако је означени број у акумулатору A_x мањи, или једнак, од означеног броја са меморијске локације M.

Операција:

$$PC \leftarrow [PC] + \$0002 + Rel \quad \text{If } (Z) + \{(N) \oplus (V)\} = 1$$

тј. If $[A_x] \leq [M]$ (други комплемент бројева са знаком)

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
BLE (rel)	Релативно	2F	2	3	-	-	-	-	-	-	-	-

BLO – Branch if Lower (исто као BCS)

Опис: Ако се BLO инструкција извршава одмах након извршавања било које од следећих инструкција, CBA, CMP (акумулатори A, B, или D), CP (индекс регистри X или Y), SBA, SUB (акумулатори A, B, или D), до гранања ће доћи само ако је бинарни број без знака у акумулатору A_x мањи од бинарног броја без знака са меморијске локације M.

Операција:

$$PC \leftarrow [PC] + \$0002 + Rel \quad \text{If } (C) = 1$$

тј. If $[A_x] < [M]$ (бинарни бројеви без знака)

11. Комплетан скуп инструкција MC68HC11

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	0	1	2	3	4	5	6	7
BLO (rel)	Релативно	25	2	3	-	-	-	-	-	-	-	-

BLS – *Branch if Lower or Same*

Опис: Ако се BLS инструкција извршава одмах након извршавања било које од следећих инструкција, CBA, CMP (A, B, или D), CP (X или Y), SBA, SUB (A, B, или D), до гранања ће доћи само ако је бинарни број без знака у акумулатору A_x мањи или једнак бинарном броју без знака са меморијске локације M.

Операција:

$PC \leftarrow [PC] + \$0002 + Rel \quad \text{If } \{(C) + (Z)\} = 1$
 тј. $\text{If } [A_x] \leq [M] \quad (\text{бинарни бројеви без знака})$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	0	1	2	3	4	5	6	7
BLS (rel)	Релативно	23	2	3	-	-	-	-	-	-	-	-

BLT – *Branch if Less than Zero*

Опис: Ако се BLT инструкција извршава одмах након извршавања било које од следећих инструкција, CBA, CMP (A, B, или D), CP (X или Y), SBA, SUB (A, B, или D), до гранања ће доћи само ако је означени број у акумулатору A_x мањи од означеног броја са меморијске локације M.

Операција:

$PC \leftarrow [PC] + \$0002 + Rel \quad \text{If } \{(N) \oplus (V)\} = 1$
 тј. $\text{If } [A_x] < [M] \quad (\text{други комплемент бројева са знаком})$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	0	1	2	3	4	5	6	7
BLT (rel)	Релативно	2D	2	3	-	-	-	-	-	-	-	-

BMI – *Branch if Minus*

Опис: Извршавање програма се наставља (грана) са адресе која се одређује додавањем бајта одступања на PC, само ако је резултат негативан. У супротном, наставља се са извршавањем следеће инструкције у секвенци. Резултат је бинарни број са знаком. Ова

BRA – *Branch Always*

Опис: Извршавање програма увек се наставља (грана) са адресе која се одређује додавањем бајта одступања на PC.

Операција:

$$PC \leftarrow [PC] + \$0002 + Rel$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
BRA (rel)	Релативно	20	2	3	-	-	-	-	-	-	-	-

BRCLR – *Branch if Bit(s) Clear*

Опис: Инструкција извршава логичку И операцију између садржаја меморијске локације М и маске која је приложена уз инструкцију. До гранања долази само ако је резултат операције једнак нули.

Операција:

$$PC \leftarrow [PC] + \$0004 + Rel \quad \text{If } \{[M] \bullet [PC + 2]\} = 0$$

$$PC \leftarrow [PC] + \$0005 + Rel \quad \text{If } [M] \bullet [PC + 3] = 0 \text{ (Y регистар)}$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
BRCL r (opr)	Директно	13	4	6	-	-	-	-	-	-	-	-
(msk)	Индексно X	1F	4	7	-	-	-	-	-	-	-	-
(rel)	Индексно Y	18 1F	5	8	-	-	-	-	-	-	-	-

BRN – *Branch Never*

Опис: Никада не долази до гранања. У пракси ова инструкција може да се употреби као 2-бајтна NOP¹, којој је потребно 3 циклуса за извршавање. Инструкција је уврштена у инструкцијски скуп као комплементарна инструкција BRA инструкцији. Такође је корисна током откривања грешака у програму² како би се поништили ефекти неке друге инструкције гранања без промене бајта одступања.

Операција:

$$PC \leftarrow (PC) + \$0002$$

¹ No operation

² Debuging

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
BRN (rel)	Релативно	21	2	3	-	-	-	-	-	-	-	-

BRSET – *Branch if Bit(s) Set*

Опис: Инструкција извршава логичку И операцију између инвертованог садржаја меморијске локације М и маске која је приложена уз инструкцију. До гранања долази само ако је резултат операције једнак нули.

Операција:

$$PC \leftarrow [PC] + \$0004 + Rel \quad \text{If } \{[] \bullet [PC + 2]\} = 0$$

$$PC \leftarrow [PC] + \$0005 + Rel \quad \text{If } \{[] \bullet [PC + 3]\} = 0 \text{ (Y регистар)}$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
BRSE Г (opr)	Директно	12	4	6	-	-	-	-	-	-	-	-
(msk)	Индексно X	1E	4	7	-	-	-	-	-	-	-	-
(rel)	Индексно Y	18 1E	5	8	-	-	-	-	-	-	-	-

BSET – *Set Bit(s) in Memory*

Опис: Поставља битове у меморијској локацији М. Битови које треба сетовати означавају се у масци са 1. Сви остали битови меморијске локације М задржавају тренутно стање.

Операција:

$$M \leftarrow [M] + [PC + 2]$$

$$M \leftarrow [M] + [PC + 3] \text{ (Y регистар)}$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
BSET (opr)	Директно	14	3	6	-	-	-	-	Δ	Δ	0	-
(msk)	Индексно X	1C	3	7	-	-	-	-	Δ	Δ	0	-
	Индексно Y	18 1C	4	8	-	-	-	-	Δ	Δ	0	-

BSR – *Branch to Subroutine*

Опис: Садржај програмског бројача (PC) инкрементира се за 2 (ово је повратна адреса): бајт мање тежине програмског бројача (PC) уписује се у стек меморију. Показивач стек меморије декрементира се за један. Затим

11. Комплетан скуп инструкција MC68HC11

се виши бајт програмског бројача (PC) уписује у стек меморију. Након тога се показивач меморије опет декрементира за један. Извршавање програма се наставља (грانا) са адресе која је одређена одступањем.

Операција:

$PC \leftarrow [PC] + \$0002$	програмски бројач (pc) се поставља на повратну адресу
$\downarrow [PCL]$	нижи бајт се уписује у стек меморију
$SP \leftarrow [SP] - \$0001$	декрементира се показивач стек меморије (sp)
$\downarrow [PCH]$	виши бајт се уписује у стек меморију
$SP \leftarrow [SP] - \$0001$	декрементира се показивач стек меморије (sp)
$PC \leftarrow [PC] + Rel$	поставља се адреса захтеваног потпрограма

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	0	1	2	3	4	5	6	7
BSR (rel)	Релативно	8D	2	6	-	-	-	-	-	-	-	-

BVC – Branch if Overflow Clear

Опис: Извршавање програма се наставља (грана) са адресе која се одређује додавањем бајта одступања на програмски бројач (PC), само ако је бит премашења постављен на 0 (V=0). У супротном, наставља се са извршавањем следеће инструкције у програмској секвенци.

Операција:

$$PC \leftarrow [PC] + \$0002 + Rel \quad \text{If } (V) = 0$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	0	1	2	3	4	5	6	7
BVC (rel)	Релативно	28	2	3	-	-	-	-	-	-	-	-

BVS – Branch if Overflow Set

Опис: Извршавање програма се наставља (грана) са адресе која се одређује додавањем бајта одступања на PC, само ако је бит премашења постављен на 1 (V=1). У супротном, наставља се са извршавањем следеће инструкције у секвенци.

Операција:

$$PC \leftarrow [PC] + \$0002 + Rel \quad \text{If } (V) = 1$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	С	Х	Н	І	Н	З	В	С
BVS (rel)	Релативно	29	2	3	-	-	-	-	-	-	-	-

CBA – *Compare Accumulators*

Опис: Упоређује се садржај акумулатора А са садржајем акумулатора В. Резултат се користи само да би се поставили N, Z и V битови. Садржаји акумулатора остају неизмењени.

Операција:

$$[A] - [B]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	С	Х	Н	І	Н	З	В	С
CBA	Имплицитно	11	1	2	-	-	-	-	Δ	Δ	Δ	Δ

CLC – *Clear Carry*

Опис: С бит регистра услова поставља се на 0. Други битови остају непромењени.

Операција:

$$C \text{ бит} \leftarrow 0$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	С	Х	Н	І	Н	З	В	С
CLC	Имплицитно	0C	1	2	-	-	-	-	-	-	-	0

CLI – *Clear Interrupt Mask*

Опис: І бит регистра услова поставља се на 0. Овим поступком се омогућавају прекиди. Други битови регистра услова остају непромењени.

Операција:

$$I \text{ бит} \leftarrow 0$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
CLI	Имплицитно	0E	1	2	-	-	-	0	-	-	-	-

CLR – Clear**Опис:** Брише се садржај акумулатора или меморијске локације M.**Операција:**

$$A_x \leftarrow 0 \quad \text{или} \quad M \leftarrow 0$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
CLR (opr)	Проширено	7F	3	6	-	-	-	-	0	1	0	0
	Индексно X	6F	2	6	-	-	-	-	0	1	0	0
	Индексно Y	18 6 ⁻	3	7	-	-	-	-	0	1	0	0
CLRA	Имплицитно	4F	1	2	-	-	-	-	0	1	0	0
CLRB	Имплицитно	5F	1	2	-	-	-	-	0	1	0	0

CLV – Clear 2's-Complement Overflow Bit

Опис: Бит премашења регистра услова поставља се на 0 (V=0).
Инструкција нема утицај на остале битове регистра услова.

Операција:

$$V \text{ бит} \leftarrow 0$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
CLV	Имплицитно	0A	1	2	-	-	-	-	-	-	0	-

CMP – Compare

Опис: Садржај акумулатора A_x упоређује се са садржајем меморијске локације M и зависно од резултата мењају се N, Z, V и C битови регистра услова (податак из меморије одузима се од садржаја селектованог акумулатора, а садржаји акумулатора и меморијске локације остају непромењени).

Операција:

$$[A_x] - [M]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
CMPA (opr)	Непосредно	81	2	2	-	-	-	-	Δ	Δ	Δ	Δ
	Директно	91	2	3	-	-	-	-	Δ	Δ	Δ	Δ

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
	Проширено	B1	3	4	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно X	A1	2	4	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно Y	18 A1	3	5	-	-	-	-	Δ	Δ	Δ	Δ
CMPB (org)	Непосредно	C1	2	2	-	-	-	-	Δ	Δ	Δ	Δ
	Директно	D1	2	3	-	-	-	-	Δ	Δ	Δ	Δ
	Проширено	F1	3	4	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно X	E1	2	4	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно Y	18 E1	3	5	-	-	-	-	Δ	Δ	Δ	Δ

COM – Complement

Опис: Инструкција комплементира (инвертује) сваки бит садржаја акумулатора A_x или садржаја меморијске локације M. Резултат ће утицати на N и Z битове, C бит се поставља на 1, а V бит се поставља на 0.

Операција:

$$A_x \leftarrow [ACCX] = \$FF - [A_x] \quad \text{или} \quad M \leftarrow [M] = \$FF - [M]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
COM (org)	Проширено	73	3	6	-	-	-	-	Δ	Δ	0	1
	Индексно X	63	2	6	-	-	-	-	Δ	Δ	0	1
	Индексно Y	18 63	3	7	-	-	-	-	Δ	Δ	0	1
COMA	Имплицитно	43	1	2	-	-	-	-	Δ	Δ	0	1
COMB	Имплицитно	53	1	2	-	-	-	-	Δ	Δ	0	1

CPD – Compare Double Accumulator

Опис: Инструкција упоређује садржај акумулатора D са 16-битном вредности која се налази на задатим адресама и мења одговарајуће битове регистра услова. Упоређивање се изводи одузимањем 16-битне вредности (M: M+1) од акумулатора D. Садржај акумулатора и меморијских локација остаје непромењен.

Операција:

$$[D] - [M: M+1]$$

11. Комплетан скуп инструкција MC68HC11

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
CPD (opr)	Непосредно	1A 83	4	5	-	-	-	-	Δ	Δ	Δ	Δ
	Директно	1A 93	3	6	-	-	-	-	Δ	Δ	Δ	Δ
	Проширено	1A B3	4	7	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно X	1A A3	3	7	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно Y	CDA 3	3	7	-	-	-	-	Δ	Δ	Δ	Δ

CPX – Compare Index Register X

Опис: Инструкција упоређује садржај индекса регистра X са 16-битном вредношћу која се налази на задатим адресама и мења одговарајуће битове регистра услова. Упоређивање се изводи одузимањем 16-битне вредности (M: M+1) од индекса регистра X. Садржај индекса регистра и меморијских локација остаје непромењен.

X регистар је 16-битни (два бајта). 16-битни операнд који се одузима од X добија се са две суседне меморијске локације почевши од адресе операнда. На пример, инструкција CPX \$B000 ће довести до тога да процесор уради следеће:

- Повезују се бајтови смештени на адресама \$B000 и \$B001 у 16-битну реч, где је бајт са адресе \$B000 бајт веће тежине.
- Одузима се ова 16-битна реч од садржаја регистра X. Знак је одређен 16. битом резултата.
- Мења се статус N, Z и V бита у зависности од резултата одузимања.
- С бит се поставља на 1 уколико је апсолутна вредност садржаја меморијских локација већа од апсолутне вредности индекса регистра, у супротном поставља се на 0.

Операција:

$$[X] - [M: M+1]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
CPX (opr)	Непосредно	8C	3	4	-	-	-	-	Δ	Δ	Δ	Δ
	Директно	9C	2	5	-	-	-	-	Δ	Δ	Δ	Δ
	Проширено	BC	3	6	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно X	AC	2	6	-	-	-	-	Δ	Δ	Δ	Δ

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
	Индексно Y	CD AC	3	7	-	-	-	-	Δ	Δ	Δ	Δ

CPY – Compare Index Register Y

Опис: Инструкција упоређује садржај индекс регистра Y са 16-битном вредности која се налази на задатим адресама и мења одговарајуће битове регистра услова (као у случају CPX). Упоређивање се изводи одузимањем 16-битне вредности (M: M+1) од индекс регистра Y. Садржај индекс регистра и меморијских локација остаје непромењен.

Операција:

$$[Y] - [M: M+1]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
CPY (opg)	Непосредно	18 8C	4	5	-	-	-	-	Δ	Δ	Δ	Δ
	Директно	18 9C	3	6	-	-	-	-	Δ	Δ	Δ	Δ
	Проширено	18 BC	4	7	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно X	1A AC	3	7	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно Y	18 AC	3	7	-	-	-	-	Δ	Δ	Δ	Δ

DAA – Decimal Adjust

Опис: Ова инструкција претвара садржај акумулатора A, после операције сабирања два BCD броја, у тачан BCD формат. DAA инструкција увек следи иза инструкција ABA, ADDA или ADCA које извршавају операције са BCD бројевима. DAA инструкција извршава корекцију која је потребна када се бинарно сабирање користи за сабирање BCD вредности.

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
DAA	Имплицитно	19	1	2	-	-	-	-	Δ	Δ	?	Δ

DEC – Decrement

Опис: Од садржаја акумулатора A_x или M одузима се 1 (декрементира се садржај). N, Z и V битови регистра услова се мењају у складу са резултатом. Ова инструкција нема утицај на бит C.

Операција:

$$A_x \leftarrow [A_x] - \$01 \text{ или } M \leftarrow [M] - \$01$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
DEC (opr)	Проширено	7A	3	6	-	-	-	-	Δ	Δ	Δ	-
	Индексно X	6A	2	6	-	-	-	-	Δ	Δ	Δ	-
	Индексно Y	18 6A	3	7	-	-	-	-	Δ	Δ	Δ	-
DECA	Имплицитно	4A	1	2	-	-	-	-	Δ	Δ	Δ	-
DECB	Имплицитно	5A	1	2	-	-	-	-	Δ	Δ	Δ	-

DES – Decrement Stack Pointer

Опис: Декрементира се садржај показивача стек меморије. Инструкција нема утицаја на битове регистра услова.

Операција:

$$SP \leftarrow [SP] - \$0001$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
DES	Имплицитно	34	1	3	-	-	-	-	-	-	-	-

DEX – Decrement Index Register X

Опис: Декрементира се садржај индекса регистра X. Ова инструкција утиче само на Z бит регистра услова.

Операција:

$$IX \leftarrow [IX] - \$0001$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
DEX	Имплицитно	09	1	3	-	-	-	-	-	Δ	-	-

DEY – Decrement Index Register Y

Опис: Декрементира се садржај индекса регистра Y. Ова инструкција утиче само на Z бит регистра услова.

Операција:

$$IY \leftarrow [IY] - \$0001$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
DEY	Имплицитно	18 09	1	4	-	-	-	-	-	Δ	-	-

EOR – Exclusive-OR

Опис: Извршава се ексклузивна ИЛИ операција између податка из меморије M, бит по бит, и садржаја селектованог акумулатора. Резултат остаје у акумулатору.

Операција:

$$A_x \leftarrow [A_x] \oplus [M]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
EORA (opr)	Непосредно	88	2	2	-	-	-	-	Δ	Δ	0	-
	Директно	98	2	3	-	-	-	-	Δ	Δ	0	-
	Проширено	B8	3	4	-	-	-	-	Δ	Δ	0	-
	Индексно X	A8	2	4	-	-	-	-	Δ	Δ	0	-
	Индексно Y	18 A8	3	5	-	-	-	-	Δ	Δ	0	-
EORB (opr)	Непосредно	C8	2	2	-	-	-	-	Δ	Δ	0	-
	Директно	D8	2	3	-	-	-	-	Δ	Δ	0	-
	Проширено	F8	3	4	-	-	-	-	Δ	Δ	0	-
	Индексно X	E8	2	4	-	-	-	-	Δ	Δ	0	-
	Индексно Y	18 E8	3	5	-	-	-	-	Δ	Δ	0	-

FDIV – Fractional Divide

Опис: Инструкција извршава нецелобројно дељење бројева без знака, 16-битни број који се налази у акумулатору D дели се 16-битним имениоцем који се налази у индекс регистру X. Након извршавања ове инструкције долази до промене одговарајућих битоа регистра услова. Количник се смешта у индекс регистар X, док се остатак смешта у акумулатор D. Претпоставља се да је бројилац мањи од имениоца, у случају премашења (именилац је мањи или једнак бројиоцу) или у случају дељења са нулом, количник се поставља на вредност \$FFFF док је остатак неодређен.

Операција:

$$[D] / [IX]; IX \leftarrow \text{количник}, D \leftarrow \text{остатак}$$

11. Комплетан скуп инструкција MC68HC11

Мнемон.....	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
FDIV	Имплицитно	03	1	41	-	-	-	-	-	Δ	Δ	Δ

IDIV – Integer Divide

Опис: Инструкција извршава целобројно дељење бројева без знака, 16-битни број који се налази у акумулатору D дели се 16-битним имениоцем који се налази у индекс регистру X. Извршавањем ове инструкције долази до промене одговарајућих битова регистра услова. Количник се смешта у индекс регистар X, док се остатак смешта у акумулатор D. Претпоставља се да је бројилац већи од имениоца. У случају дељења са нулом, количник се поставља на вредност \$FFFF док је остатак неодређен.

Операција:

$[D] / [IX]; IX \leftarrow \text{количник}, D \leftarrow \text{остатак}$

Мнемон.....	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
IDIV	Имплицитно	02	1	41	-	-	-	-	-	Δ	0	Δ

INC – Increment

Опис: На садржај A_x или M додаје се 1 (инкрементира се садржај). N, Z и V битови регистра услова мењају се у складу са резултатом. Ова инструкција нема утицај на бит C.

Операција:

$A_x \leftarrow [A_x] + \$01$ или $M \leftarrow [M] + \$01$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
INC (opr)	Проширено	7C	3	6	-	-	-	-	Δ	Δ	Δ	-
	Индексно X	6C	2	6	-	-	-	-	Δ	Δ	Δ	-
	Индексно Y	18 6C	3	7	-	-	-	-	Δ	Δ	Δ	-
INCA	Имплицитно	4C	1	2	-	-	-	-	Δ	Δ	Δ	-
INCB	Имплицитно	5C	1	2	-	-	-	-	Δ	Δ	Δ	-

INS – Increment Stack Pointer

Опис: Инкрементира се садржај показивача стек меморије.

Инструкција нема утицаја на битове регистра услова.

Операција:

$$SP \leftarrow [SP] + \$0001$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
INS	Имплицитно	31	1	3	-	-	-	-	-	-	-	-

INX – Increment Index Register X

Опис: Инкрементира се садржај индекса регистра X. Ова инструкција утиче само на Z бит регистра услова.

Операција:

$$IX \leftarrow [IX] + \$0001$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
INX	Имплицитно	08	1	3	-	-	-	-	-	Δ	-	-

INY – Increment Index Register Y

Опис: Инкрементира се садржај индекса регистра Y. Ова инструкција утиче само на Z бит регистра услова.

Операција:

$$IY \leftarrow [IY] + \$0001$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
INY	Имплицитно	18 08	2	4	-	-	-	-	-	Δ	-	-

JMP – Jump

Опис: Процесор узима следећу инструкцију са адресе која је специфицирана операндом. Ова инструкција има два могућа начина адресирања и оба су илустрована у даљем тексту:

Проширено

7E	JMP \$3C77
3C	
77	

11. Комплетан скуп инструкција MC68HC11

Ова инструкција доводи до тога да се вредност операнда, \$3C77, смести у програмски бројач. Процесор затим узима следећу инструкцију са адресе \$3C77.

Индексно	
6E	JMP \$24, X
24	

Инструкција проузрокује да процесор дода одступање \$24, које је 8-битни број без знака, на садржај X регистра да би се добила адреса са које ће бити прочитана следећа инструкција. На пример, уколико је $[X] = 03A0_{16}$, онда ефективна вредност операнда постаје $03A0 + \$24 = \$03C4$. Затим се програмски бројач (PC) пуни са $03C4$, и процесор наставља са извршавањем програма од те адресе надаље.

Операција:

PC ← Ефективна адреса

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
JMP (opr)	Проширено	7E	3	3	-	-	-	-	-	-	-	-
	Индексно X	6E	2	3	-	-	-	-	-	-	-	-
	Индексно Y	18 6E	3	4	-	-	-	-	-	-	-	-

JSR – Jump to Subroutine

Опис: Садржај програмског бројача (PC) се инкрементира за 3 или за 2 у зависности од начина адресирања који се користи. Онда се програмски бројач (PC) уписује у стек меморију (бајт по бајт), тако што се прво уписује бајт мање тежине. Показивач стек меморије указује на следећу слободну локацију стек меморије. Извршавање се наставља (скаче се) са инструкцијом која је смештена на ефективној адреси. Ефективна адреса је у складу са правилима за индексно, директно или проширено адресирање.

Операција:

PC ← [PC] + \$0003	за проширено или индексно, Y адресирање или:
PC ← [PC] + \$0002	за директно или индексно, X адресирање
↓[PCL]	нижи бајт се уписује у стек меморију
SP ← [SP] - \$0001	декрементира се показивач стек меморије (SP)

$\downarrow[\text{PCN}]$	виши бајт се уписује у стек меморију
$\text{SP} \leftarrow [\text{SP}] - \$0001$	декрементира се показивач стек меморије (SP)
$\text{PC} \leftarrow \text{ефективна адреса}$	поставља се адреса захтеваног потпрограма

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
JSR (opr)	Директно	9D	2	5	-	-	-	-	-	-	-	-
	Проширено	BD	3	6	-	-	-	-	-	-	-	-
	Индексно X	AD	2	6	-	-	-	-	-	-	-	-
	Индексно Y	18 A D	3	7	-	-	-	-	-	-	-	-

LDA – Load Accumulator

Опис: Садржај меморијске локације која је специфицирана адресом операнда смешта се у жељени акумулатор. Садржај меморијске локације остаје неизмењен.

Операција:

$$A_x \leftarrow [M]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
LDAA (opr)	Непосредно	86	2	2	-	-	-	-	Δ	Δ	0	-
	Директно	96	2	3	-	-	-	-	Δ	Δ	0	-
	Проширено	B6	3	4	-	-	-	-	Δ	Δ	0	-
	Индексно X	A6	2	4	-	-	-	-	Δ	Δ	0	-
	Индексно Y	18 A6	3	5	-	-	-	-	Δ	Δ	0	-
LDAB (opr)	Непосредно	C6	2	2	-	-	-	-	Δ	Δ	0	-
	Директно	D6	2	3	-	-	-	-	Δ	Δ	0	-
	Проширено	F6	3	4	-	-	-	-	Δ	Δ	0	-
	Индексно X	E6	2	4	-	-	-	-	Δ	Δ	0	-
	Индексно Y	18 E6	3	5	-	-	-	-	Δ	Δ	0	-

LDD – Load Double Accumulator

Опис: У двоструки акумулатор D смешта се садржај меморијских локација M и M+1. Битови регистра услова мењају се у зависности од података. Податак који се налази на меморијској локацији M смешта се у акумулатор A, док се податак са меморијске локације M+1 смешта у акумулатор B.

Операција:

$$D \leftarrow [M : M+1]; A \leftarrow [M], B \leftarrow [M+1]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
LDD (opr)	Непосредно	CC	3	3	-	-	-	-	Δ	Δ	0	-
	Директно	DC	2	4	-	-	-	-	Δ	Δ	0	-
	Проширено	FC	3	5	-	-	-	-	Δ	Δ	0	-
	Индексно X	EC	2	5	-	-	-	-	Δ	Δ	0	-
	Индексно Y	18 EC	3	6	-	-	-	-	Δ	Δ	0	-

LDS – Load Stack Pointer

Опис: Смешта се 16 битова (два бајта) података из меморије у регистар показивача стек меморије (SP). 16 битова података узима се са две узастопне меморијске локације почевши од адресе операнда.

На пример, инструкција LDS \$45EA ће сместити бајт са адресе \$45EA у SPH (виши бајт показивача стек меморије - SP), и бајт са адресе \$45EB у SPL (нижи бајт показивача стек меморије - SP).

Операција:

$$SPH \leftarrow [M], SPL \leftarrow [M+1]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
LDS (opr)	Непосредно	8E	3	3	-	-	-	-	Δ	Δ	0	-
	Директно	9E	2	4	-	-	-	-	Δ	Δ	0	-
	Проширено	BE	3	5	-	-	-	-	Δ	Δ	0	-
	Индексно X	AE	2	5	-	-	-	-	Δ	Δ	0	-
	Индексно Y	18 AE	3	6	-	-	-	-	Δ	Δ	0	-

LDX – Load Index register X

Опис: Смешта се 16 битова (два бајта) података из меморије у индекс регистар X. 16 битова података узима се са две узастопне меморијске локације почевши од адресе операнда.

На пример, инструкција LDX \$45EA ће сместити бајт са адресе \$45EA у XH (виши бајт индекс регистра X), а бајт са адресе \$45EB у XL (нижи бајт индекс регистра X).

Операција:

$$IXH \leftarrow [M], IXL \leftarrow [M+1]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
LDX (opr)	Непосредно	CE	3	3	-	-	-	-	Δ	Δ	0	-
	Директно	DE	2	4	-	-	-	-	Δ	Δ	0	-
	Проширено	FE	3	5	-	-	-	-	Δ	Δ	0	-
	Индексно X	EE	2	5	-	-	-	-	Δ	Δ	0	-
	Индексно Y	CD EE	3	6	-	-	-	-	Δ	Δ	0	-

LDY – Load Index register Y

Опис: Смешта се 16 битова (два бајта) података из меморије у Y индекс регистар. 16 битова података се узима са две узастопне меморијске локације почевши од адресе операнда.

На пример, инструкција LDX \$45EA ће сместити бајт са адресе \$45EA у YH (виши бајт индекс регистра Y), а бајт са адресе \$45EB у YL (нижи бајт индекс регистра Y).

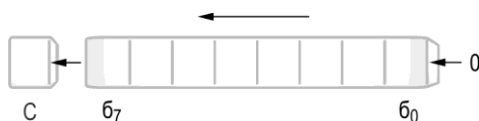
Операција:

$$IYH \leftarrow [M], IYL \leftarrow [M+1]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
LDY (opr)	Непосредно	18 CE	4	4	-	-	-	-	Δ	Δ	0	-
	Директно	18 DE	3	5	-	-	-	-	Δ	Δ	0	-
	Проширено	18 FE	4	6	-	-	-	-	Δ	Δ	0	-
	Индексно X	1A EE	3	6	-	-	-	-	Δ	Δ	0	-
	Индексно Y	18 EE	3	6	-	-	-	-	Δ	Δ	0	-

LSL – Logical Shift Left (исто као ASL)

Опис: Ова инструкција врши померај свих битова акумулатора A_x или меморијске локације за једно место улево. У бит најмање тежине се смешта 0. У C бит регистра услова поставља се бит највеће тежине.

Операција:

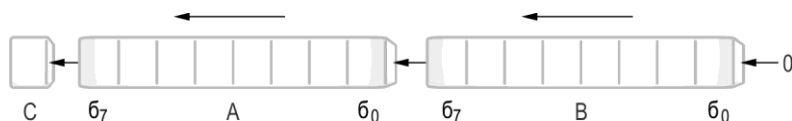
11. Комплетан скуп инструкција MC68HC11

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
LSL (opr)	Проширено	78	3	6	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно X	68	2	6	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно Y	18 68	3	7	-	-	-	-	Δ	Δ	Δ	Δ
LSLA	Имплицитно	48	1	2	-	-	-	-	Δ	Δ	Δ	Δ
LSLB	Имплицитно	58	1	2	-	-	-	-	Δ	Δ	Δ	Δ

LSLD – Logical Shift Left Double Accumulator (исто као ASLD)

Опис: Инструкција помера све битове акумулатора D за једно место улево. Бит 0 поставља се на 0. У бит C регистра услова смешта се бит највеће тежине из акумулатора D.

Операција:

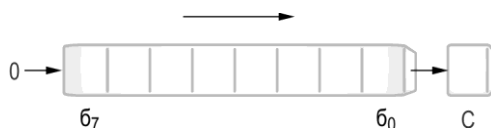


Мнемони	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
LSLD	Имплицитно	05	1	3	-	-	-	-	Δ	Δ	Δ	Δ

LSR – Logical Shift Right

Опис: Ова инструкција врши померај свих битава акумулатора A_x или меморијске локације за једно место удесно. У бит највеће тежине се смешта 0. У C бит регистра услова поставља се бит најмање тежине.

Операција:



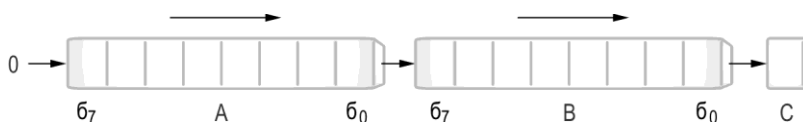
Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
LSR (opr)	Проширено	74	3	6	-	-	-	-	0	Δ	Δ	Δ
	Индексно X	64	2	6	-	-	-	-	0	Δ	Δ	Δ

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
	Индексно Y	18 64	3	7	-	-	-	-	0	Δ	Δ	Δ
LSRA	Имплицитно	44	1	2	-	-	-	-	0	Δ	Δ	Δ
LSRB	Имплицитно	54	1	2	-	-	-	-	0	Δ	Δ	Δ

LSRD – Logical Shift Right Double Accumulator

Опис: Инструкција помера све битове акумулатора D за једно место удесно. Бит 15 (бит највеће тежине у акумулатору A) поставља се на 0. У бит C регистра услова смешта се бит најмање тежине из акумулатора D (бит најмање тежине у акумулатору B).

Операција:



Мнемони...	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
LSRD	Имплицитно	04	1	3	-	-	-	-	0	Δ	Δ	Δ

MUL – Multiply Unsigned

Опис: Инструкција врши множење неозначених (без знака) 8-битног садржаја акумулатора A и 8-битног садржаја акумулатора B. Резултат се смешта у акумулатор D. С бит регистра услова омогућава заокруживање бајта највеће тежине преко секвенце инструкција: MUL, ADCA #0.

Операција:

$$D \leftarrow [A] \times [B]$$

Мнемон...	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
MUL	Имплицитно	3D	1	10	-	-	-	-	-	-	-	Δ

NEG – Negate

Опис: Од садржаја акумулатора AX или селектване меморијске локације M формира се други комплемент. Суштински конвертује

11. Комплетан скуп инструкција MC68HC11

позитивне у негативне бројеве и обрнуто. У случају вредности \$80 не долази до промене.

Операција:

$$A_x \leftarrow -[A_x] = \$00 - [A_x] \quad \text{или} \quad M \leftarrow -[M] = \$00 - [M]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
NEG (opr)	Проширено	70	3	6	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно X	60	2	6	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно Y	18 60	3	7	-	-	-	-	Δ	Δ	Δ	Δ
NEGA	Имплицитно	40	1	2	-	-	-	-	Δ	Δ	Δ	Δ
NEGB	Имплицитно	50	1	2	-	-	-	-	Δ	Δ	Δ	Δ

NOP – No Operation

Опис: Процесор не извршава никакве операције (активности) осим што инкрементира програмски бројач да би се припремио за узимање следеће инструкције. Ова инструкција нема утицаја на остале регистре и најчешће се користи за генерисање временског кашњења у програму.

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
NOP	Имплицитно	01	1	2	-	-	-	-	-	-	-	-

ORA – Inclusive-OR

Опис: Ова инструкција извршава логичку ИЛИ операцију између садржаја акумулатора A_x и садржаја меморијске локације M бит по бит. Резултат операције смешта се у акумулатор A_x .

Операција:

$$A_x \leftarrow [A_x] + [M]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
ORAA (opr)	Непосредно	8A	2	2	-	-	-	-	Δ	Δ	0	-
	Директно	9A	2	3	-	-	-	-	Δ	Δ	0	-
	Проширено	BA	3	4	-	-	-	-	Δ	Δ	0	-
	Индексно X	AA	2	4	-	-	-	-	Δ	Δ	0	-

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
	Индексно Y	18 AA	3	5	-	-	-	-	Δ	Δ	0	-
ORAB (opr)	Непосредно	CA	2	2	-	-	-	-	Δ	Δ	0	-
	Директно	DA	2	3	-	-	-	-	Δ	Δ	0	-
	Проширено	FA	3	4	-	-	-	-	Δ	Δ	0	-
	Индексно X	EA	2	4	-	-	-	-	Δ	Δ	0	-
	Индексно Y	18 EA	3	5	-	-	-	-	Δ	Δ	0	-

PSH – Push Data onto Stack

Опис: Садржај специфицираног акумулатора уписује се у стек меморију, на адресу специфицирану преко показивача стек меморије (SP). Садржај показивача стек меморије (SP) затим се декрементира.

Операција:

$$\downarrow A_x, SP \leftarrow [SP] - \$0001$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
PSHA	Имплицитно	36	1	3	-	-	-	-	-	-	-	-
PSHB	Имплицитно	37	1	3	-	-	-	-	-	-	-	-

PSHX – Push Index Register X onto Stack

Опис: Садржај индекса регистра X уписује се (смешта) у стек меморију (прво нижи бајт, па виши бајт), на адресе одређене показивачем стек меморије (SP). Садржај показивача стек меморије (SP) по завршетку ове операције декрементиран је два пута.

Операција:

$$\downarrow [IXL], SP \leftarrow [SP] - \$0001$$

$$\downarrow [IHX], SP \leftarrow [SP] - \$0001$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
PSHX	Имплицитно	3C	1	4	-	-	-	-	-	-	-	-

PSHY – Push Index Register Y onto Stack

Опис: Садржај индекса регистра Y уписује се у стек меморију

11. Комплетан скуп инструкција MC68HC11

(прво нижи бајт, па виши бајт) на адресе одређене показивачем стек меморије (SP). Садржај показивача стек меморије (SP) декрементира се два пута.

Операција:

$\downarrow[\text{IYL}], \text{SP} \leftarrow [\text{SP}] - \$0001$

$\downarrow[\text{IYH}], \text{SP} \leftarrow [\text{SP}] - \$0001$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
PSHY	Имплицитно	18 3C	2	5	-	-	-	-	-	-	-	-

PUL – Pull Data from Stack

Опис: Показивач стек меморије (SP) инкрементира се. Садржај локације одређене (специфициране) садржајем показивача стек меморије (SP) смешта се у акумулатор A_x .

Операција:

$\text{SP} \leftarrow [\text{SP}] + \$0001, \uparrow[A_x]$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
PULA	Имплицитно	32	1	4	-	-	-	-	-	-	-	-
PULB	Имплицитно	33	1	4	-	-	-	-	-	-	-	-

PULX – Pull Index Register X from Stack

Опис: Садржај индекса регистра X се скида (очитава) из стек меморије (прво виши бајт, па нижи бајт), почевши од адресе одређене садржајем показивача стек меморије (SP) увећаним за један. Садржај показивача стек меморије (SP) инкрементира се укупно два пута.

Операција:

$\text{SP} \leftarrow [\text{SP}] + \$0001, \uparrow[\text{IXH}]$

$\text{SP} \leftarrow [\text{SP}] + \$0001, \uparrow[\text{IXL}]$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
PULX	Имплицитно	38	1	5	-	-	-	-	-	-	-	-

PULY – Pull Index Register Y from Stack

Опис: Садржај индекса регистра Y читава се из стек меморије (прво виши бајт, па нижи бајт), почевши од адресе одређене показивачем стек меморије (SP) увећаним за један. Садржај показивача стек меморије (SP) инкрементира се укупно два пута.

Операција:

$$SP \leftarrow [SP] + \$0001, \uparrow[IYH]$$

$$SP \leftarrow [SP] + \$0001, \uparrow[IYL]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
PULY	Имплицитно	18 38	2	6	-	-	-	-	-	-	-	-

ROL – Rotate Left

Опис: Садржај акумулатора A_x или меморијске локације M помера се за једно место улево. На позицију бита b_0 уписује се садржај C бита регистра услова. У C бит се уписује бит највеће тежине (b_7) акумулатора A_x или меморијске локације M.

Операција:

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
ROL (opr)	Проширено	79	3	6	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно X	69	2	6	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно Y	18 69	3	7	-	-	-	-	Δ	Δ	Δ	Δ
ROLA	Имплицитно	49	1	2	-	-	-	-	Δ	Δ	Δ	Δ
ROLB	Имплицитно	59	1	2	-	-	-	-	Δ	Δ	Δ	Δ

ROR – Rotate Right

Опис: Садржај акумулатора A_x или меморијске локације M помера се за једно место удесно. У бит највеће тежине (b_7) смешта се садржај

С бита регистра услова. У С бит се смешта бит најмање тежине (b_0) акумулатора A_x или меморијске локације М.

Операција:



Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
ROR (опг)	Проширено	76	3	6	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно X	66	2	6	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно Y	18 66	3	7	-	-	-	-	Δ	Δ	Δ	Δ
RORA	Имплицитно	46	1	2	-	-	-	-	Δ	Δ	Δ	Δ
RORB	Имплицитно	56	1	2	-	-	-	-	Δ	Δ	Δ	Δ

RTI – Return from Interrupt

Опис: Процесор читава (узима) девет узастопних бајтова из стек меморије, инкрементирајући показивач стек меморије (SP) при сваком читавању. Девет читаних бајтова се уписује (смешта) у регистар услова (CCR), акумулатор В, акумулатор А, индекс регистар X (виши бајт ХН, нижи бајт XL), индекс регистар Y (виши бајт YН, нижи бајт YL), програмски бројач (виши бајт PCН и нижи бајт PCL), респективно. Овим се враћа садржај ових регистара процесора на вредности које су имали када је процесор реаговао (одговарао) на прекид. Нова вредност програмског бројача (PC) ће вратити процесор на извршавање одговарајуће инструкције у главном програму. Х бит у регистру услова може бити обрисан као резултат RTI инструкције, али неће бити постављен ако је био обрисан пре извршавања RTI инструкције.

Операција:

$SP \leftarrow [SP] + \$0001, \uparrow[CCR]$

$SP \leftarrow [SP] + \$0001, \uparrow[B]$

$SP \leftarrow [SP] + \$0001, \uparrow[A]$

$SP \leftarrow [SP] + \$0001, \uparrow[IXH]$

$SP \leftarrow [SP] + \$0001, \uparrow[IXL]$

$SP \leftarrow [SP] + \$0001, \uparrow[IYH]$
 $SP \leftarrow [SP] + \$0001, \uparrow[IYL]$
 $SP \leftarrow [SP] + \$0001, \uparrow[PCH]$
 $SP \leftarrow [SP] + \$0001, \uparrow[PCL]$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
RTI	Имплицитно	3B	1	12	Δ	$\downarrow$	Δ	Δ	Δ	Δ	Δ	Δ

RTS – Return from Subroutine

Опис: Ово би требало да буде последња инструкција било ког потпрограма. Процесор читава (скида) два бајта из стек меморије и инкрементира показивач стек меморије (SP) пре сваког читавања. Два бајта се уписују у програмски бројач (PCH и PCL, респективно). Са новом вредношћу програмског бројача процесор наставља са извршавањем инструкције у главном програму која је одмах иза инструкције JSR (скок у потпрограм).

Операција:

$SP \leftarrow [SP] + \$0001, \uparrow[PCH]$
 $SP \leftarrow [SP] + \$0001, \uparrow[PCL]$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
RTS	Имплицитно	39	1	5	-	-	-	-	-	-	-	-

SBA – Subtract Accumulators

Опис: Садржај акумулатора В одузима се од садржаја акумулатора А и резултат се смешта у акумулатор А. Садржај акумулатора В остаје непромењен.

Операција:

$A \leftarrow [A] - [B]$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
SBA	Имплицитно	10	1	2	-	-	-	-	Δ	Δ	Δ	Δ

SBC – *Subtract with Carry*

Опис: Одузима се садржај меморијске локације и вредност С бита од акумулатора A_x , а резултат остаје у акумулатору.

Операција:

$$A_x \leftarrow [A_x] - [M] - [C]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
SBCA (opr)	Непосредно	82	2	2	-	-	-	-	Δ	Δ	Δ	Δ
	Директно	92	2	3	-	-	-	-	Δ	Δ	Δ	Δ
	Проширено	B2	3	4	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно X	A2	2	4	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно Y	18 A2	3	5	-	-	-	-	Δ	Δ	Δ	Δ
SBCB (opr)	Непосредно	C2	2	2	-	-	-	-	Δ	Δ	Δ	Δ
	Директно	D2	2	3	-	-	-	-	Δ	Δ	Δ	Δ
	Проширено	F2	3	4	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно X	E2	2	4	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно Y	18 E2	3	5	-	-	-	-	Δ	Δ	Δ	Δ

SEC – *Set Carry*

Опис: Поставља се С бит регистра услова на 1. Нема утицаја на друге битове.

Операција:

$$C \text{ бит} \leftarrow 1$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
SEC	Имплицитно	0D	1	2	-	-	-	-	-	-	-	1

SEI – *Set Interrupt Mask*

Опис: Поставља се I бит регистра услова на 1. Када је I бит постављен на 1, процесор ће реаговати само на немаскиране прекиде као и софтверски прекид (SWI). Нема утицаја на друге битове регистра услова.

Операција:

$$I \text{ бит} \leftarrow 1$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
SEI	Имплицитно	0F	1	2	-	-	-	1	-	-	-	-

SEV – Set 2's-Complement Overflow Bit

Опис: Поставља се V бит регистра услова на 1. Нема утицаја на друге битове.

Операција:

$$V \text{ бит} \leftarrow 1$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
SEV	Имплицитно	0B	1	2	-	-	-	-	-	-	1	-

STA – Store Accumulator

Опис: Садржај акумулатора A_x уписује се у меморијску локацију M. Садржај акумулатора A_x остаје непромењен.

Операција:

$$M \leftarrow [A_x]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
STAA (opr)	Директно	97	2	3	-	-	-	-	Δ	Δ	0	-
	Проширено	B7	3	4	-	-	-	-	Δ	Δ	0	-
	Индексно X	A7	2	4	-	-	-	-	Δ	Δ	0	-
	Индексно Y	18 A7	3	5	-	-	-	-	Δ	Δ	0	-
STAB (opr)	Директно	D7	2	3	-	-	-	-	Δ	Δ	0	-
	Проширено	F7	3	4	-	-	-	-	Δ	Δ	0	-
	Индексно X	E7	2	4	-	-	-	-	Δ	Δ	0	-
	Индексно Y	18 E7	3	5	-	-	-	-	Δ	Δ	0	-

STD – Store Double Accumulator

Опис: Садржај акумулатора D смешта се у меморију. Садржај акумулатора D остаје непромењен.

Операција:

$$M: M+1 \leftarrow [D], M \leftarrow [A], M+1 \leftarrow [B]$$

11. Комплетан скуп инструкција MC68HC11

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
STD (opr)	Директно	DD	2	4	-	-	-	-	Δ	Δ	0	-
	Проширено	FD	3	5	-	-	-	-	Δ	Δ	0	-
	Индексно X	ED	2	5	-	-	-	-	Δ	Δ	0	-
	Индексно Y	18 ED	3	6	-	-	-	-	Δ	Δ	0	-

STOP – Stop Processing

Опис: Ако је S бит регистра услова постављен на вредност 1 STOP инструкција је онемогућена и има исту улогу као NOP инструкција. Ако је бит S регистра услова обрисан извршавање STOP инструкције проузрокује заустављање свих системских такт сигнала. Систем прелази у режим минималне потрошње енергије. Регистри процесора остају непромењени, а улазно/излазни прикључци (пинови) остају такође непромењени. Повратак из STOP режима може се постићи сигнаlima RESET, немаскирајућим прекидом $\overline{XIRQ}$ или са немаскираним маскирајућим прекидом $\overline{IRQ}$. Ако се повратак из STOP режима постиже прекидним сигналом $\overline{XIRQ}$, и ако је X бит регистра услова обрисан, извршавање програма биће настављено реаговањем на $\overline{XIRQ}$ прекид (смештање података у стек меморију и извршавање потпрограма за обраду $\overline{XIRQ}$ прекида). Ако је X бит регистра услова постављен $\overline{XIRQ}$ прекид је маскиран, извршавање програма биће настављено инструкцијама које следе након STOP инструкције.

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
STOP	Имплицитно	CF	1	2	-	-	-	-	-	-	-	-

STS – Store Stack Pointer

Опис: Бајт веће тежине показивача стек меморије (SP) смешта се у меморију на адресу специфицирану у програму, а бајт мање тежине показивача стек меморије смешта на следећу меморијску локацију.

Операција:

$M \leftarrow [SPH], M+1 \leftarrow [SPL]$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
STS (opr)	Директно	9F	2	4	-	-	-	-	Δ	Δ	0	-
	Проширено	BF	3	5	-	-	-	-	Δ	Δ	0	-
	Индексно X	AF	2	5	-	-	-	-	Δ	Δ	0	-
	Индексно Y	18 AF	3	6	-	-	-	-	Δ	Δ	0	-

STX – Store Index Register X

Опис: Бајт веће тежине индекс регистра X смешта се у меморију на адресу специфицирану у програму, а бајт мање тежине индекс регистра X смешта се у следећу меморијску локацију.

Операција:

$$M \leftarrow [IXH], M+1 \leftarrow [IXL]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
STX (opr)	Директно	DF	2	4	-	-	-	-	Δ	Δ	0	-
	Проширено	FF	3	5	-	-	-	-	Δ	Δ	0	-
	Индексно X	EF	2	5	-	-	-	-	Δ	Δ	0	-
	Индексно Y	CD EF	3	6	-	-	-	-	Δ	Δ	0	-

STY – Store Index Register Y

Опис: Бајт веће тежине индекс регистра Y смешта се у меморију на адресу специфицирану у програму, а бајт мање тежине индекс регистра Y смешта се у следећу меморијску локацију.

Операција:

$$M \leftarrow [IYH], M+1 \leftarrow [IYL]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
STY (opr)	Директно	18 DF	3	5	-	-	-	-	Δ	Δ	0	-
	Проширено	18 FF	4	6	-	-	-	-	Δ	Δ	0	-
	Индексно X	1A EF	3	6	-	-	-	-	Δ	Δ	0	-
	Индексно Y	18 EF	3	6	-	-	-	-	Δ	Δ	0	-

SUB – Subtract

Опис: Одузима се садржај меморијске локације од акумулатора A_x , а резултат се смешта у акумулатор A_x . Код одузимања С бит регистра услова представља бит позајмице.

Операција:

$$A_x \leftarrow [A_x] - [M]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
SUBA (opr)	Непосредно	80	2	2	-	-	-	-	Δ	Δ	Δ	Δ
	Директно	90	2	3	-	-	-	-	Δ	Δ	Δ	Δ
	Проширено	B0	3	4	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно X	A0	2	4	-	-	-	-	Δ	Δ	Δ	Δ
SUBB (opr)	Индексно Y	18 A0	3	5	-	-	-	-	Δ	Δ	Δ	Δ
	Непосредно	C0	2	2	-	-	-	-	Δ	Δ	Δ	Δ
	Директно	D0	2	3	-	-	-	-	Δ	Δ	Δ	Δ
	Проширено	F0	3	4	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно X	E0	2	4	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно Y	18 E0	3	5	-	-	-	-	Δ	Δ	Δ	Δ

SUBD – Subtract Double Accumulator

Опис: Одузима се садржај меморијских локација M: M+1 од садржаја акумулатора D, а резултат се смешта у акумулатор D. Код одузимања С бит регистра услова представља бит позајмице.

Операција:

$$D \leftarrow [D] - [M: M+1]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
SUBD (opr)	Непосредно	83	2	4	-	-	-	-	Δ	Δ	Δ	Δ
	Директно	93	2	5	-	-	-	-	Δ	Δ	Δ	Δ
	Проширено	B3	3	6	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно X	A3	2	6	-	-	-	-	Δ	Δ	Δ	Δ
	Индексно Y	18 A3	3	7	-	-	-	-	Δ	Δ	Δ	Δ

SWI – Software Interrupt

Опис: Програмски бројач (PC) се инкрементира за један.

Програмски бројач (PC), индекс регистри Y и X, акумулатори A и B и регистар услова (CCR) смештају се у стек меморију. Показивач стек меморије се декрементира после сваког бајта који се упише у стек меморију. I бит регистра услова се поставља на 1. У програмски бројач (PC) затим се уписује адреса коју садржи SWI вектор, и извршавање инструкција се наставља од те локације. Извршење ове инструкције се не може спречити I битом.

Операција:

$SP \leftarrow [SP] + \$0001$
 $\downarrow[PCL], SP \leftarrow [SP] - \$0001$
 $\downarrow[PCH], SP \leftarrow [SP] - \$0001$
 $\downarrow[IYL], SP \leftarrow [SP] - \$0001$
 $\downarrow[IYH], SP \leftarrow [SP] - \$0001$
 $\downarrow[IXL], SP \leftarrow [SP] - \$0001$
 $\downarrow[IXH], SP \leftarrow [SP] - \$0001$
 $\downarrow[A], SP \leftarrow [SP] - \$0001$
 $\downarrow[B], SP \leftarrow [SP] - \$0001$
 $\downarrow[CCR], SP \leftarrow [SP] - \$0001$
 $I \leftarrow 1, PC \leftarrow [SWI \text{ вектор}]$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
SWI	Имплицитно	3F	1	14	-	-	-	1	-	-	-	-

TAB - Transfer from Accumulator A to Accumulator B

Опис: Преноси се садржај акумулатора A у акумулатор B. Претходни садржај акумулатора B се губи, док садржај акумулатора A остаје непромењен.

Операција:

$B \leftarrow [A]$

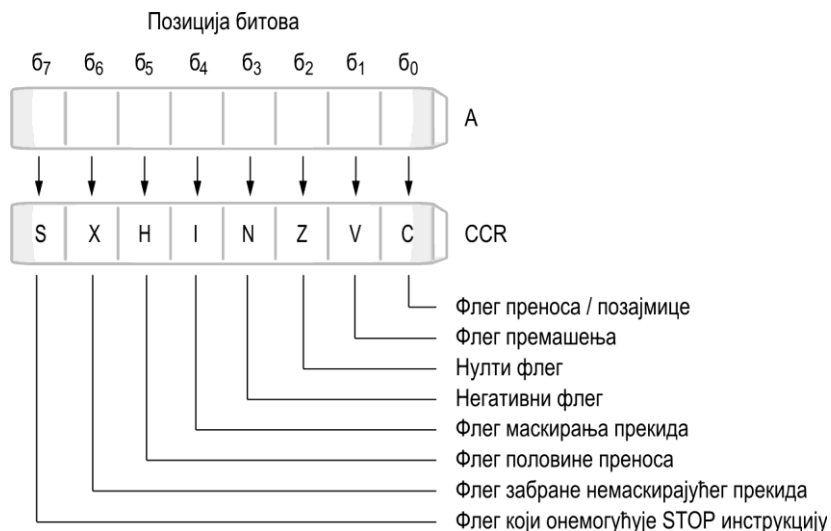
Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
TAB	Имплицитно	16	1	2	-	-	-	-	Δ	Δ	0	-

TAP - *Transfer from Accumulator A to Condition Code Register*

Опис: Садржај акумулатора А преноси се (уписује) у регистар услова (CCR). Садржај акумулатора А се не мења. Х бит у регистру услова може бити обрисан као последица TAP инструкције, али не може бити постављен на вредност 1 ако је био обрисан пре TAP инструкције.

Операција:

$$CCR \leftarrow [A]$$



Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
TAP	Имплицитно	06	1	2	Δ	↓	Δ	Δ	Δ	Δ	Δ	Δ

TBA - *Transfer from Accumulator B to Accumulator A*

Опис: Преноси се садржај акумулатора В у акумулатор А. Претходни садржај акумулатора А се губи, док садржај акумулатора В остаје непромењен.

Операција:

$$A \leftarrow [B]$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
TBA	Имплицитно	17	1	2	-	-	-	-	Δ	Δ	0	-

TEST – *Test Operation (Test Mode Only)*

Опис: Ова инструкција доводи до континуираног инкрементирање програмског бројача (PC). Може се извршавати само у тест режиму рада. Да би се прекинуло извршавање ове инструкције процесор мора бити ресетован. Извршавање програма је одложено за време извршавања ове инструкције. Када процесор није у тест режиму рада овај операциони кођ није дозвољен.

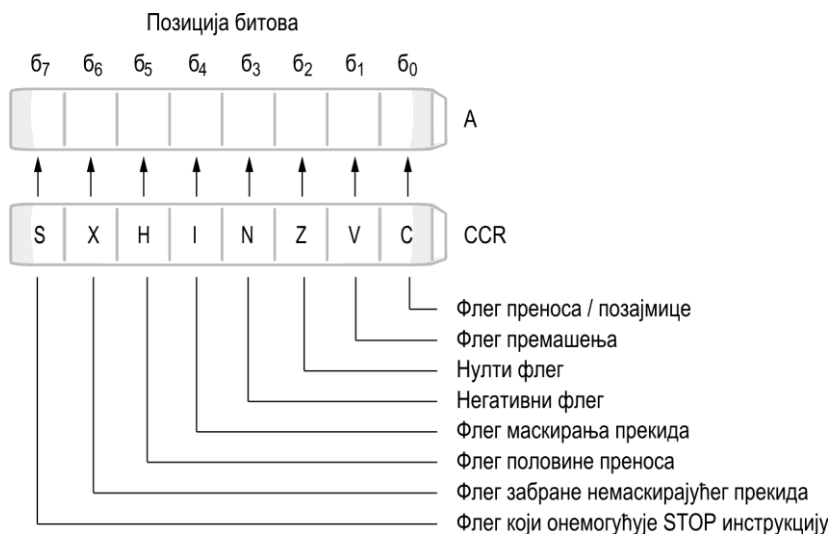
Мнемони...	Начин адресирања	Оп. кођ	Број бајтова	Број циклуса	С	А	И	Т	Н	З	В	С
TEST	Имплицитно	00	1	*	-	-	-	-	-	-	-	-

TPA - *Transfer from Condition Code Register to Accumulator A*

Опис: Садржај регистра услова (CCR) смешта се у акумулатор А. Садржај регистра услова се не мења.

Операција:

$$A \leftarrow [CCR]$$



Мнемон...	Начин адресирања	Оп. кођ	Број бајтова	Број циклуса	С	А	И	Т	Н	З	В	С
TPA	Имплицитно	07	1	2	-	-	-	-	-	-	-	-

TST – Test

Опис: Од садржаја акумулатора A_x или меморијске локације M се одузима \$00, и битови регистра услова се постављају на одговарајуће вредности. Инструкција не мења садржај акумулатора A_x као ни садржај меморијске локације.

Операција:

$$[A_x] - \$00 \text{ или } [M] - \$00$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
TST (opr)	Проширено	7D	3	6	-	-	-	-	Δ	Δ	0	0
	Индексно X	D	2	6	-	-	-	-	Δ	Δ	0	0
	Индексно Y	18 6D	3	7	-	-	-	-	Δ	Δ	0	0
TSTA	Имплицитно	4D	1	2	-	-	-	-	Δ	Δ	0	0
TSTB	Имплицитно	5D	1	2	-	-	-	-	Δ	Δ	0	0

TSX - Transfer from Stack Pointer to Index Register X

Опис: У индекс регистар X смешта се садржај стек показивача увећан за један. Садржај стек показивача остаје непромењен. После TSX инструкције индекс регистар X указује на последњу вредност која је смештена у стек меморији.

Операција:

$$IX \leftarrow [SP] + \$0001$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
TSX	Имплицитно	30	1	3	-	-	-	-	-	-	-	-

TSY - Transfer from Stack Pointer to Index Register Y

Опис: У индекс регистар Y смешта се садржај стек показивача увећан за један. Садржај стек показивача остаје непромењен. После TSY инструкције, индекс регистар Y указује на последњу вредност која је смештена у стек меморију.

Операција:

$$IY \leftarrow [SP] + \$0001$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
TSY	Имплицитно	18 30	2	4	-	-	-	-	-	-	-	-

TXS - *Transfer from Index Register X to Stack Pointer*

Опис: У показивач стек меморије смешта се садржај индекса регистра X умањен за један. Садржај индекса регистра X остаје непромењен.

Операција:

$$SP \leftarrow [IX] - \$0001$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
TXS	Имплицитно	35	1	3	-	-	-	-	-	-	-	-

TYS - *Transfer from Index Register Y to Stack Pointer*

Опис: У показивач стек меморије смешта се садржај индекса регистра Y умањен за један. Садржај индекса регистра Y остаје непромењен.

Операција:

$$SP \leftarrow [IY] - \$0001$$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
TYS	Имплицитно	18 35	2	4	-	-	-	-	-	-	-	-

WAI – *Wait for Interrupt*

Опис: Програмски бројач (PC) се инкрементира за један. Програмски бројач (PC), индекс регистри Y и X, акумулатори A и B и регистар услова (CCR) уписују се у стек меморију. Показивач стек меморије се декрементира после сваког бајта који се упише у стек меморију.

Након тога процесор прелази у стање чекања. Процесор излази из стања чекања када се појави немаскирани прекид. Чекање увек траје цео број такт циклуса.

По изласку из стања чекања процесор поставља I бит регистра услова на 1. У програмски бројач (PC) уписује се вектор (адреса) која одговара прекиду који је наступио, и извршавање инструкција се

11. Комплетан скуп инструкција MC68HC11
наставља од те локације надалје.

Иако WAI инструкција не утиче на битове регистра услова, прекид који је наступио поставља I бит регистра услова на вредност 1 (или X бит у случају када наступи XIRQ прекид).

Операција:

$SP \leftarrow [SP] + \$0001$
 $\downarrow [PCL], SP \leftarrow [SP] - \$0001$
 $\downarrow [PCH], SP \leftarrow [SP] - \$0001$
 $\downarrow [IYL], SP \leftarrow [SP] - \$0001$
 $\downarrow [IYH], SP \leftarrow [SP] - \$0001$
 $\downarrow [IXL], SP \leftarrow [SP] - \$0001$
 $\downarrow [IXH], SP \leftarrow [SP] - \$0001$
 $\downarrow [A], SP \leftarrow [SP] - \$0001$
 $\downarrow [B], SP \leftarrow [SP] - \$0001$
 $\downarrow [CCR], SP \leftarrow [SP] - \$0001$
 $I \leftarrow 1, PC \leftarrow [SWI \text{ вектор}]$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
WAI	Имплицитно	3E	1	14	-	-	-	-	-	-	-	-

XGDX – *Exchange Double Accumulator and Index Register X*

Опис: Заменјују се садржаји акумулатора D и индекс регистра X.

Операција:

$[IX] \leftrightarrow [D]$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
XGDX	Имплицитно	8F	1	3	-	-	-	-	-	-	-	-

XGDY – *Exchange Double Accumulator and Index Register Y*

Опис: Заменјују се садржаји акумулатора D и индекс регистра Y.

Операција:

$[IY] \leftrightarrow [D]$

Мнемоник	Начин адресирања	Оп. код	Број бајтова	Број циклуса	S	X	H	I	N	Z	V	C
XGDY	Имплицитно	18 8F	2	4	-	-	-	-	-	-	-	-