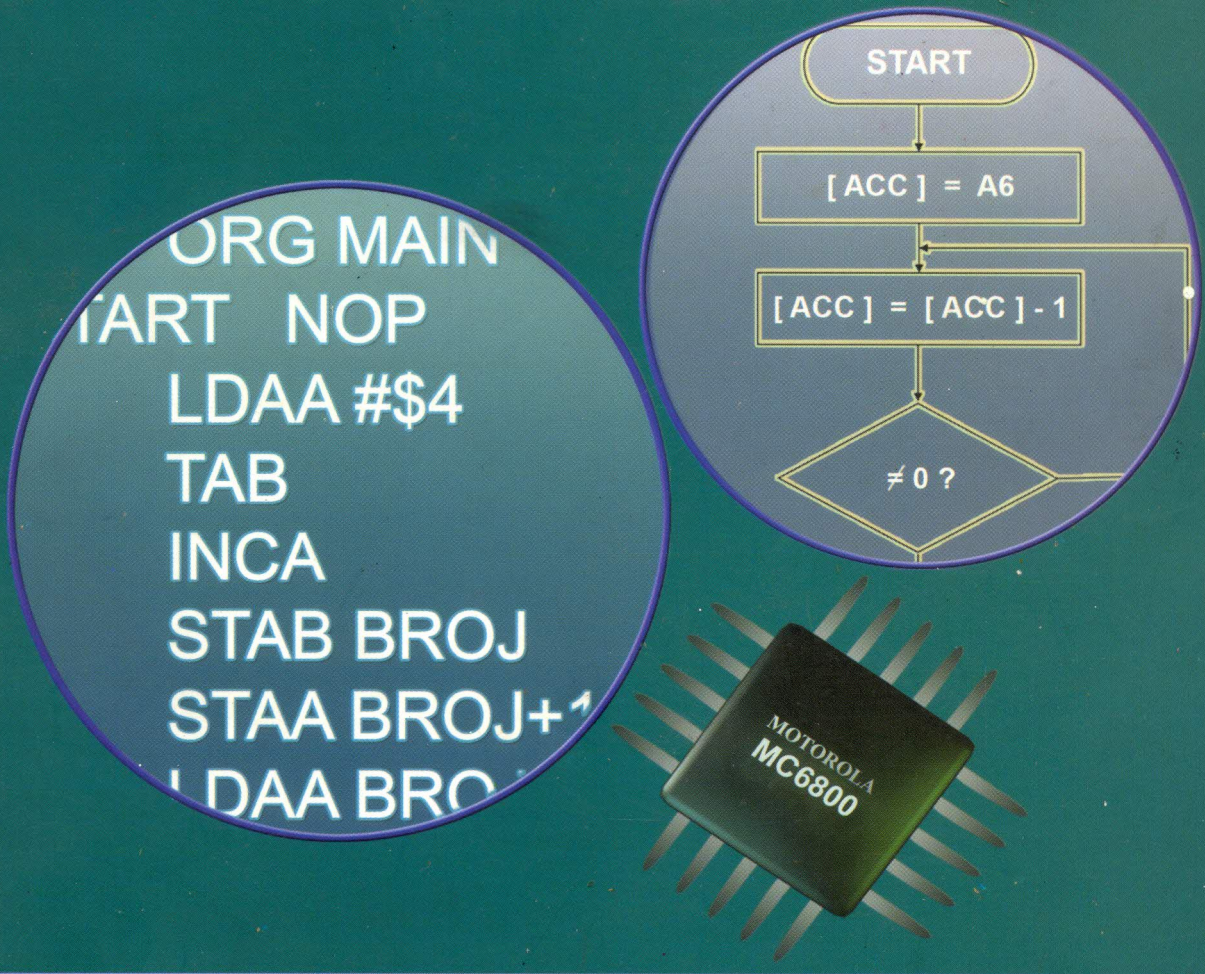
Верица Васиљевић
Бранислав Павић
Бошко Богојевић
Бошко Николић
Владимир Тадић
Александар Јосић

МИКРОРАЧУНАРИ

ЗБИРКА ЗАДАТАКА



ВИША ЕЛЕКТРОТЕХНИЧКА ШКОЛА

Верица Васиљевић
Владимир Тадић
Бошко Богојевић

Бранислав Павић
Бошко Николић
Александар Јосић

ЗБИРКА ЗАДАТАКА И ПИТАЊА

ИЗ МИКРОРАЧУНАРА

Виша електротехничка школа у Београду
2006.

Рецензенти:
Мр Борислав Хаџибабић
Мр Драгана Прокин

Издавач
Виша електротехничка школа у Београду

За издавача
Др Драгољуб Мартиновић

Дизајн
Кристијан Кук

Лектор и коректор
Анђелка Ковачевић

Рачунарска обрада
Верица Васиљевић, Бранислав Павић, Владимир Тадић
Бошко Николић, Бошко Богојевић, Александар Јосић

Штампано у 300 примерака

Наставно веће Више електротехничке школе у Београду , на својој седници од 11.1.2006. год. одобрило је издавање и примену Збирке у настави.

CIP – Каталогизација у публикацији
Народна библиотека Србије, Београд

004.382.7(075.8)(076)
004.31(075.8)(076)

ЗБИРКА задатака и питања из микрорачунара / Верица Васиљевић . . . [и др.]. - Београд : Виша електротехничка школа, 2005 (Аранђеловац : Графопак). – 227 стр. : табеле ; 25 cm

Према предговору, ово је 1. изд. – Тираж 300. – Библиографија: стр. 227.

- - Микрорачунари [Електронски извор] : збирка задатака. – 1 електронски оптички диск (CD-ROM) : слика, звук, видео ; 12 cm

ISBN 86-85081-45-9
ISBN 86-85081-47-5 (CD)
1. Васиљевић, Верица
а) Рачунари – Задаци б) Микропроцесори – Задаци
COBISS.SR – ID 128329228

Предговор

Ова збирка намењена је студентима Више електротехничке школе у Београду који су изабрали предмет Микрорачунари. Настала је са циљем да студентима помогне у овладавању рада са процесором MC6800, микроконтролером MC68HC11 као и у припреми испита из предмета Микрорачунари. Збирком су обухваћена:

- теоријска питања везана за програмирање процесора MC6800 и микроконтролера MC68HC11 на основу којих се проверава ниво припремљености студената за рад на лабораторијским вежбама, и
- решени задаци који су у последњих неколико година били на писменом делу испиту из предмета Микрорачунари.

Теоријска питања подељена су у дванаест целина (групе питања везана за конкретну лабораторијску вежбу). На почетку сваке целине налазе се питања са комплетним објашњењем решења а након тога је скуп свих питања везаних за конкретну лабораторијску вежбу коју студенти треба самостално да реше, при чему су дати одговори без објашњења.

Задаци су подељени у пет група, према областима на које се односе. Прва група је скуп примера који треба да помогну студентима у савладавању инструкционог сета. Преостале четири подгрупе су систематизовани задаци из ранијих испитних рокова. На почетку сваке подгрупе налазе се решени задаци са комплетним објашњењима. На крају сваке подгрупе налазе се задаци за вежбу. Решења ових задатака дата су на крају Збирке.

На крају Збирке дат је опис комплетног инструкцијског скупа процесора MC6800 и део инструкцијског скупа микроконтролера MC68HC11.

Да би се студентима олакшао рад код куће уз Збирку припремљен је и компакт диск који се може користити и за самостално тестирање студената преко рачунара.

Ово је прво издање Збирке задатака и питања из Микрорачунара.

Београд, 2006.

Аутори

Садржај

Објашњење инструкционог сета на примерима

1) Претпоставимо да је [ACCA]=B6$_{16}$, [ACCB]=6A$_{16}$, [0250]=37$_{16}$ и C=1. За сваку од ових инструкција, одредите операнд и C флег после извршења инструкције.

a) ASL $0250
b) ASRA
c) LSRA
d) ROL $0250
e) RORB

Решење: а) ASL $0250 извршава операцију над садржајем меморијске локације 0250. Операција се одвија на следећи начин:

C	B7	B6	B5	B4	B3	B2	B1	B0	
1	0	0	1	1	0	1	1	1	[0250] пре померања
0	0	1	1	0	1	1	1	0	[0250] после померања

Уочимо да је сваки бит операнда померен улево, са битом 7 помереним у C флег и 0 помереном у бит Б0. Тако су [0250]= 01101110$_2$=6E$_{16}$ и C=0 после операције померања улево. У извршавању инструкције, MPU смешта (пуни) операнд у ALU, извршава операцију померања операнда, и затим га смешта натраг у меморију.

	B7	B6	B5	B4	B3	B	B1	B0	C
[ACCA] пре померања	1	0	1	1	0	1	1	0	1
[ACCA] после померања	1	1	0	1	1	0	1	1	0

b) ASRA оперише над садржајем ACCA. Операција се извршава овако:

Уочимо да се сваки бит операнда помера десно. Бит 0 се помера у C флег, а бит 7 остаје непромењен. Тако су после извршавања операције [ACCA]= 11011011=DB и C=0.

c) LSRA се извршава над [ACCA]. Операција се извршава на следећи начин:

	B7	B6	B5	B4	B3	B2	B1	B0	C
[ACCA] пре померања	1	0	1	1	0	1	1	0	1
[ACCA] после померања	0	1	0	1	1	0	1	1	0

Уочимо да се сваки бит помера удесно. Бит 0 се помера у C флег а 0 се помера на место бита 7. Тако је после завршене операције [ACCA] = 01011011=5B і C=0.

d) ROL $0250 се извршава над [0250].

C	B7	B6	B5	B4	B3	B2	B1	B0	
1	0	0	1	1	0	1	1	1	[0250] пре ротације
0	0	1	1	0	1	1	1	1	[0250] после ротације

Уочимо да се сваки бит помера улево, са C флегом који се ротира у бит Б0 и битом 7 који се помера у C флег. Тако је после операције ротирања [0250]= 01101111= 6F.

e) RORB се извршава над [ACCB] .

Сваки бит се помера удесно, C флег се ротира у бит 7 и бит 0 се помера у C флег. Тако је после операције [ACCB]= 10110101= B5 і C=0.

	B7	B6	B5	B4	B3	B2	B1	B0	C
[ACCB] пре померања	0	1	1	0	1	0	1	0	1
[ACCB] после померања	1	0	1	1	0	1	0	1	0

2) Одредити [ACCA] пошто се изврши следећа инструкција: LDAA #$37, COMA.

Решење: LDAA #$37 производи [ACCA]=$00110111_2$. COMA инструкција инвертује сваки бит да би добила [ACCA] = $11001000_2 = C8_{16}$

3) MPU 6800 нема инструкцију која одузима [ACCA] од [ACCB]. Написати инструкцијску секвенцу која одузима [ACCA] од [ACCB], а резултат оставља у [ACCB].

Решење:

Инструкцијски код	Мнемоник	Коментар
40	NEGA	; Други комплемент [ACCA]
1B	ABA	; Додати [ACCB]
16	TAB	; Пренети га у [ACCB]

Прве две инструкције извршавају операцију одузимања додавањем другог комплемента ACCA на ACCB. Пошто резултат ABA инструкције остаје у ACCA, потребна је TAB инструкција за преношење резултата у ACCB.

4) Претпоставимо да је бајт података FF смештен на адресу 0250. Одредити [0250] после извршавања две узастопне INC $0250 инструкције.

Решење: Прва INC $0250 ће додати 1 на FE, па се добија FF. Следећа INC $0250 ће додати 1 на FF и добиће се 00. MPU ће игнорисати било какав пренос који је проузрокован инструкцијом за инкрементирање.

5) Одредити [ACCB] после извршавања инструкцијских секвенци: CLRB, DECB.

Решење: Резултат CLRB инструкције је [ACCB]= 00000000_2. DECB инструкција одузима 1 од [ACCB] и то додавањем другог комплемента броја 00000001 као што је илустровано:

```
  00000000 [ACCB]
+ 11111111 други комплемент од 00000001
  11111111
        └──────→ резултат смештен у ACCB
```

Уочимо да ова операција одузимања нормално доводи до позајмице, и MPU ће поставити C=1. За инструкцију декрементирања, MPU занемарује позајмљивање, и не мења C флег.

6) Испитати инструкцијску секвенцу. Одредити [ACCA] у тренутку заустављања MPU-а.

0200	86	LDAA #$6A
0201	6A	
0202	7E	JMP $0250
0203	02	
0204	50	
0205	4A	DECA
0206	3E	WAI
.		
.		
0250	4C	INCA
0251	43	COMA
0252	3E	WAI

Решење: Резултат LDAA #$6A је [ACCA]=6A. Инструкција JMP $0250 ће проузроковати да MPU скочи по следећу инструкцију на адресу 0250. Другим речима, MPU "скаче преко" адреса 0250-024F. На 0250 MPU извршава INCA инструкцију која производи [ACCA]=6A+1=6B=01101011. Затим наставља са извршавањем на локацији 0251, где прави комплемент [ACCA], и производи [ACCA]= 10010100=94_{16}. На крају MPU се зауставља на 0252.

7) У инструкцијској секвенци BCC инструкција ће довести да се MPU грана на адресу 580E уколико операција померања производи C=0. Проверити?

Адреса	Лабела	Инстр. код	Мнемоник	Коментар
57A0	START	48	ASLA	; Помери [ACCA] лево
57A1		24	BCC $580E	; Гранај се на 580E
57A2		6B		; ако је C=0
57A3	KRAJ	3E	WAI	; Заустави се ако је C=1

Решење:

Тренутни [PC]		57A3
Офсет	+	006B
Нови [PC]		580E

8) У претходном примеру променити инструкцијски код на адреси 57A2 на 94 и одредити адресу гранања.

Решење: Одступање (офсет) је негативно пошто је $94_{16}=10010100_2=-108_{10}$.

Тренутни [PC]		57A3
Офсет	+	FF94
Нови [PC]		5737

9) Посматрајмо инструкцијску секвенцу која следи. Израчунати вредност бајта одступања инструкције BEQ.

Адреса	Лабела	Инстр. код	Мнемоник	Коментар
0370	START	10	SBA	; Одузми акумулаторе
0371		27	BEQ $039C	; Гранај се на 039C
0372		??		; уколико је резултат нула
0373		97	STAA $50	; У противном, смести
0374		50		; [ACCA] у меморију на 0050
0375	KRAJ	3E	WAI	; Заустави се

Решење: Пошто MPU узме BEQ оп. код и одступање, а [PC] је 0373; ово је текући [PC]. Нови [PC] је 039C, тако да ће бити гранање унапред. Одступање се добија одузимањем на следећи начин:

Нови [PC]	039C
Текући [PC]	- 0373
Одступање	29

Бајт на адреси 0372 треба да буде 29.

10) За претходни пример израчунати одступање уколико BEQ теба да се грана на 031D.

Решење: Пошто је ово бројање уназад, нови [PC] = 031D. Треба га одузети од текућег [PC] = 0373.

Текући [PC]	0373
Нови [PC]	- 039D
	56

11) Поновити претходни пример уколико БЕQ треба да се грана на 02A3.

Решење:

Текући [PC]	0373
Нови [PC]	- 02A3
	DO

Пошто је разлика већа од 80_{16}, нови [PC] је ван опсега, тако да никакво одступање неће функционисати.

12) Модификовати претходни пример тако да се грана на 0200 у случају да је бит највеће тежине 1.

Адреса	Лабела	Инстр. код	Мнемоник	Коментар
0225	START	B6	LDAA $F700	; Напуни ACCA са

0226	F7		; улазног порта
0227	00		
0228	2B	BMI $0200	; Гранај се на 0200 ако
0229	D6		; је MSB = 1
022A	.		; У супротном настави

Решење: Није неопходно извршавати операцију померања, пошто је MSB бит-бит знака. Када се LDA инструкција изводи, N флег ће узети вредност MSB податка. Затим BMI инструкција тестира N флег и грана се уколико је он 1.

13) Коју операцију ће MPU да изврши за следећу инструкцију?

	Инструкцијски код	Асемблерски језик
Оп код →	CE	LDX #$03A7
16-битни →	03	
податак	A7	

Решење: Ова инструкција користи непосредно адресирање на шта указује симбол #. Пошто је X регистар 16-битни регистар, подаци које треба сместити у X су специфицирани преко два бајта која се налазе непосредно иза оп. кода. Тако ће ова тробајтна инструкција сместити (напунити) у X регистра податак 03A7.

14) Шта ће се десити ако се у програму JSR инструкција промени у JMP инструкцију?

Решење: Када MPU извршава JMP $C160, пуни програмски бројач са адресом C160 али у стеку не чува адресу повратног скока. Затим извршава потпрограм. Када извршава RTS, узима са стека два бајта података и смешта их у PC. Ова два бајта неће бити повратна адреса. Ово доводи да MPU одлази на непредвиђену адресу као своју следећу инструкцију. Даљи резултат је извршавање неке случајне, непредвиђене операције. Другим речима долази до потпуног ″слома″ програма.

15) Одредити садржај X регистра по завршетку следеће инструкцијске секвенце.

Адреса	Инструкцијски код	Асемблерски језик
0200	CE	LDX #$FFFE
0201	FF	
0202	FE	
0202	08	INX
0204	08	INX

Решење: Инструкција LDX #$FFFE користи непосредно адресирање да би напунила [X] са 16-битним податком $FFFE_{16}$ =1111 1111 1111 1110_2. Прва инструкција [X] ће инкрементирати на $FFFF_{16}$. Следећа ће довести [X] на 0000_{16}.

16) Пронаћи оп. код за асемблерску инструкцију ADDA #$34 и ADDB #$34.

Решење: Погледајмо мнемоник ADD. Табела иза описа инструкције даје информације и о инструкцији ADDA и о ADDB. ADDA и ADDB инструкција имају сва четири начина адресирања. Нас интересује непосредан начин адресирања због ознаке ″#″ у асемблерској инструкцији. Из табеле видимо да је оп. код за ADDA (непосредно) 8B. Тако ће се инструкција ADDA #$34 превести у два бајта 8B 34.
Проналазимо да је оп. код за ADDB (непосредно) CB. Тако се ADDB #$34 преводи у два бајта, CB 34.

17) Конвертовати LDX $A433 у код машинског језика.

Решење: Ова инструкција користи проширено адресирање да би напунила X регистар подацима. Видимо из табеле да је оп. код за мнемоник LDX и проширено адресирање FE. Тако се, LDX $A433 претвара у FE A4 33.

18) Претворити STAB $67 у машински код.

Решење: Ова инструкција користи директно адресирање пошто је адреса одређена са само две цифре (нпр. адреса операнда је у ствари $0067).

Погледајмо мнемоник STA и пронађимо у табели STAB (директно). Операциони код је дат са D7. Тако се STAB $67 преводи у D7 67.

19) 6800 MPU може да извршва одређене операције над подацима који се налазе у меморији као и над подацима који се налазе у неком од акумулатора. Једна од ових операција је операција брисања, која поставља све битове на 0 (тј. постојећи садржај меморије или акумулатора замењује са 00_{16}). Погледајмо у Прилогу I мнемоник CLR. Ово је мнемоник инструкције која брише садржај меморијске локације. Погледајмо затим мнемонике CLRA/CLRB. Ово су мнемоници за брисање ACCA и ACCB, респективно.

Решење: Оп. код за CLR (проширено адресирање) је 7F. Тако, CLR $03FF се преводи у 7F 03 FF. оп. код за CLRA (имплицитно) је 4F. Тако се, CLRA, претвара у 4F. Уочимо да је CLRA једнобајтна инструкција.

20) Претворити следеће инструкције у машински код: CLR $03FF и CLRA.

Решење: Оп. код за CLR (проширено адресирање) је 7F. Тако, CLR $03FF се преводи у 7F 03 FF. оп. код за CLRA (имплицитно) је 4F. Тако се, CLRA, претвара у 4F. Уочимо да је CLRA једнобајтна инструкција.

21) Упоредити време потребно MPU за инкрементирање ACCA са временом потребним за инкрементирање X регистра.

Решење: Погледајмо мнемоник INCA који је мнемоник за инкрементирање садржаја акумулатора A. Одговарајућа табела показује да су потребна два такт циклуса за извршавање ове инструкције. Погледајмо сада мнемоник INX, који је мнемоник за инкрементирање садржаја X регистра. Табела показује да су потребна четири такт циклуса за извршавање ове инструкције. Више циклуса је потребно за инкрементирање X регистра него за инкрементирање акумулатора A. Пошто је X регистар 16-битни, а MPU ради са 8-битном аритметиком, потребна су му два корака за инкрементирање.

22) На који флег од флегова регистра услова утиче извршавање инструкције LDAA #$A5?

Решење: Погледајмо мнемоник LDA и пронађимо у табели LDAA (непосредно). У табели видимо да на N и Z флегове утичу подаци који се смештају у акумулатор A; ово је означено са ″ x ″ испод колона у табели означених са N и Z. У овом случају, подаци су $A5_{16}=10100101_2$. Пошто податак нема вредност нула, Z флег ће бити постављен на 0. N флег ће бити постављен на 1 пошто је бит знака податка (бит највеће тежине MSB) једнак 1.

Табела такође показује да ће V бит бити обрисан (постављен на 0) кад год се инструкција изводи , без обзира на податке

23) Проучите следећи кратак програм и одредите садржај ACCA, ACCB, и N и Z флега по завршетку програма.

Адреса	Лабела	Инстр. код	Мнемоник	Коментари
0300	START	86	LDAA #$D7	; Напуни ACCA са подацима
0301		D7		
0302		97	STAA $27	; Смести ACCA у меморију
0303		27		
0304		D6	LDAB $27	; Напуни ACCB из меморије
0305		27		
0306	KRAJ	3E	WAI	; Заустави извршавање

Решење: Прва инструкција, LDAA #$D7, користи непосредан начин адресирања да би напунила податком D7 ACCA. Друга инструкција, STAA $27, користи директан начин адресирања да би сместила [ACCA] у меморијску локацију $0027. Трећа инструкција пуни ACCB са подацима са адресе 0027. Последња инструкција, WAI, суштински зауставља MPU. Табела у даљем тексту сумира статус MPU регистара, флегова и меморијских локација после извршења сваке инструкције.

Инструкција	Статус после извршења
LDAA #$D7	[ACCA] = D7= 11010111 ; N=1, Z=0

STAA $27	[$0027] = D7= 11010111	; N=1, Z=0
LDAB $27	[ACCB] = D7= 11010111	; N=1, Z=0
WAI	Нема промена	

24) Програм у даљем тексту смешта [ACCA] и [ACCB] у меморију. Одредити адресе у које се смештају садржаји акумулатора.

Адреса	Лабела	Инстр. код	Мнемоник		Коментар
0300	START	8E	LDS #$01FF		; Напуни SP
0301			01		
0302			FF		
0303			36	PSHA	; Смести ACCA на стек
0304			37	PSHB	; Смести ACCB на стек
0305	KRAJ		3E	WAI	; Заустави извршавање

Решење: LDS #$01FF инструкција користи непосредно адресирање да би напунила стек поинтер [SP] са 01FF. Подсетимо се да је SP 16-битни регистар, тако да треба да буду специфицирана два узастопна бајта иза оп. кода. PSHA инструкција "гура" [ACCA] у меморијску локацију одређену са [SP]=01FF и затим декрементира [SP] на 01FE. PSHB инструкција смешта [ACCB] на меморијску локацију одређену са [SP]=01FE и затим декрементира [SP] на 01FD.
На крају: ACCA је смештен на локацију 01FF, а ACCB на локацију 01FE.

25) Употребити одговарајуће инструкције да би се направила ефикаснија верзија претходног програма.

Решење: Програм у претходном примеру је употребљен за преношење бајта у ACCA и затим пребацивање у ACCB. Ово се може постићи са само три инструкције као што ће бити приказано у овом решењу:

Адреса	Лабела	Инстр. код	Мнемоник	Коментар
0300	START	86	LDAA #$D7	; Напуни ACCA са подацима

```
0301            D7
0302            16      TAB     ; Пренеси податке у ACCB
0303    KRAJ    3E      WAI     ; Заустави извршавање
```

Аритметичке операције
(двобајтно сабирање, одузимање, множење и дељење)

26) Описати операције које извршава следећа инструкција

Инструкцијски код	Асемблерски језик
FB 02 53	ADDB $0253

Решење: Ова инструкција ће проузроковати да MPU узме податке са адресе 0243 и дода их садржају акумулатора B. Операциони код FB указује MPU-у које операције треба да извршава (ADDB) и коју врсту адресирања да користи.

27) Описати операције које извршава следећа инструкција.

Инструкцијски код	Асемблерски језик
DB 9C	ADDB $9C

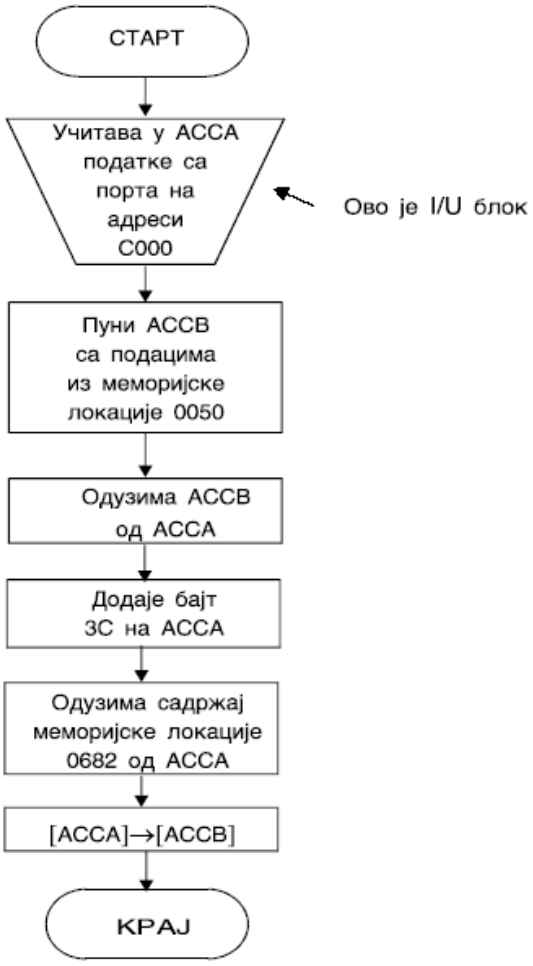
Решење: Ова инструкција ће изазвати MPU да узме податке са адресе 009C и дода их акумулатору B. Оп. код DB казује MPU-у коју операцију да извршава [ADDB] и који начин адресирања да користи.

28) Описати операције коју извршавају следеће инструкцијске секвенце:

Адреса	Инструкцијски код	Асемблерски језик
0200	D6	LDAB $65
0201	65	
0202	CB	ADDB #$02
0202	02	
0204	F7	STAB $8F00
0205	8F	
0206	00	

Решење: Прва инструкција, да би акумулатор B напунила подацима са адресе $0065, користи директно адресирање. Подсетимо се да се адреса интерпретира као да је на страници 00 (нпр. АДВ =00)
Друга инструкција користи непосредно адресирање (уочимо симбол #) да би податак 02 сабрала са садржајем акумулатора B.
Трећа инструкција користи проширено адресирање да би се сместила садржај акумулатора B на меморијску адресу $8F00.

29) Дијаграм тока програма који извршава неколико операција сабирања и одузимања. Написати програм за овај дијаграм тока.



Дијаграм тока програма

Решење:

Адреса	Лабела	Инстр. код	Мнемоник
2000	START	B6	LDAA $C000
2001		C0	
2002		00	
2003		D6	LDAB $0050
2004		50	
2005		10	SBA
2006		8B	ADDA #$3C
2007		3C	
2008		B0	SUBA $0682
2009		06	
200A		82	
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 оп. кодове за сваку инструкцију.

30) Претпоставити да акумулатори садрже бројеве без знака. Написати инструкцијску секвенцу која почиње од адресе 5C00 и грана се на 5C6F уколико је [ACCA] већи него [ACCB]; у супротном програм се зауставља.

Решење:

Адреса	Лабела	Инстр. код	Мнемоник	Коментар
5C00	START	10	SBA	; Одузми акумулаторе
5C01		22	BHI $56CF	; Гранај се на 5C6F
5C02		6C		; ако је резултат > 0
5C03	KRAJ	3E	WAI	; У супротном, крај

Инструкција SBA одузима [ACCB] од [ACCA] и оставља резултат у ACCA.

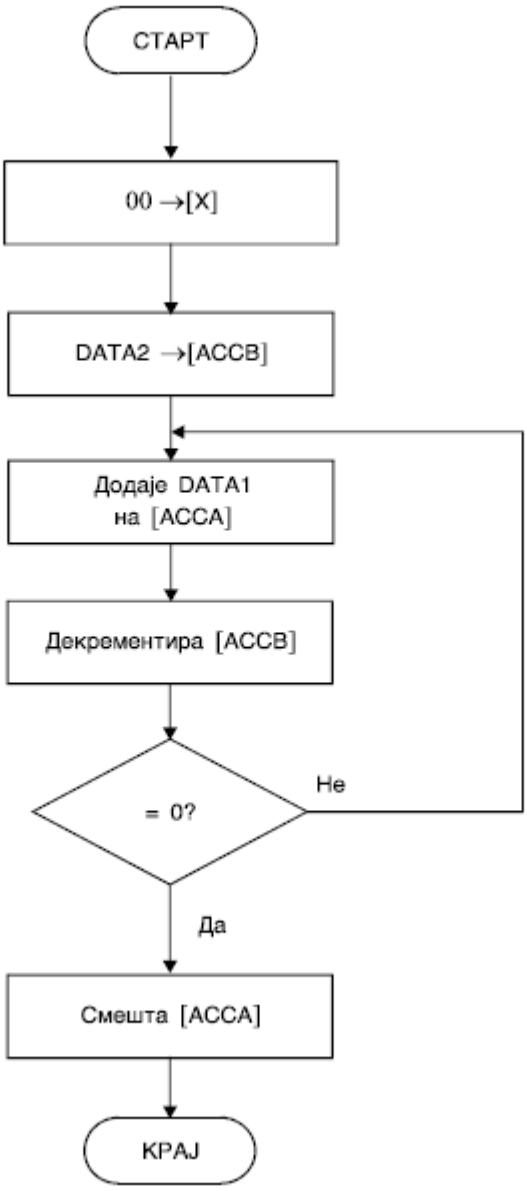
31) Како треба променити програм у претходном примеру под условом да акумулатори садрже бројеве са знаком?

Решење: Променити BHI у BGT. BGT инструкција ће тестирати резултат одузимања бројева са знаком.

32) Дијаграм тока програма који множи два 8-битна броја без знака, DATA1 и DATA2 и смешта их у меморију. Множење се постиже процесом узастопног сабирања. На пример, претпоставимо да је DATA1=2 и DATA2=4. Додавањем броја 2 четири пута може се добити производ 2X4. Другим речима, вредност DATA2 је број који указује колико пута додајемо вредност DATA1.

У дијаграму тока, АССА се брише и АССБ се пуни са DATA2. Петља у дијаграму тока представља поновљено извршавање сабирања DATA1 са АССА. После сваког сабирања, [АССВ] се декрементира и проверава да ли је постао нула. Када достигне нулу, постигнут је потребан број операција сабирања, и коначан резултат је у АССА.

Написати програм за овај дијаграм тока који почиње на адреси 0260. Претпоставити да су DATA1 и DATA2 у меморијској локацији 0080 и 0081, респективно, и да резултат треба сместити на 0082.



Дијаграм тока програма за множење преко сабирања

Решење:

Адреса	Лабела	Инстр. код	Мнемоник	Коментар
0260	START	4F	CLRA	; Поставити ACCA на 00
0261		D6	LDAB $81	; Напуни ACCB са DATA2
0262		81		
0263		9B	ADDA $80	; Додај DATA1 на ACCA
0264		80		
0265		5A	DECB	; Декрементирај [ACCB]
0266		26	BNE $0263	; Ако није нула, гранај се
0267		FB		; и поново додај DATA1
0268		97	STAA $82	; Завршено множење
0269		82		; сачувај резултат
026A	KRAJ	3E	WAI	; Крај

33) Претпоставимо да се у оба акумулатора налазе бројеви без знака. Написати инструкцијску секвенцу која почиње на адреси 5C00 и грана се до 5C6F уколико је [ACCA] веће од [ACCB]; у супротном програм се зауставља.

Решење:

Адреса	Лабела	Инстр. код	Мнемоник	Коментар
5C00	START	11	CBA	; Упоредити акумулаторе
5C01		22	BHI $5C6F	; Гранај се на 5C6F ако
5C02		6C		; је [ACCA]>[ACCB]
5C03		3E	WAI	; У супротном заустави се

CBA инструкција одузима [ACCB] од [ACCA], али резултат не оставља у ACCA; садржај оба акумулатора се не мења. Инструкција BHI проверава да ли је резултат већи од нуле пошто ово указује да је [ACCA]>[ACCB]. Уколико је овај услов испуњен, програм се грана на 5C6F.

34) Број $DCBA (представљен у неозначеној аритметици) сместити на меморијске локације $1000 (виши бајт) и $1001 (нижи бајт) и поделити га са 4, од добијене вредности одузети број $DBCA а резултат сместити на меморијске локације $0 (виши бајт) и $1 (нижи бајт).

Објашњење:

Да би се решио овај задатак потребно је реализовати алгоритме за:

- дељење двобајтног броја и
- одузимање два двобајтна броја.

Алгоритам за дељење двобајтног броја

У овом задатку имамо специјалан случај дељења. Двобајтни број се дели са 2 на n (у нашем случају n је 2, 2 на 2 је 4). За овај специјални случај није потребно реализовати генерални алгоритам за дељење већ се може применити особина да један шифт у десно одговара дељењу са 2. Аналогно овоме два шифта у десно би одговарала дељењу са 4, три шифта дељењу са 8 итд. Дељење увек почиње од бајта највеће тежине. У случају да се дели број представљен у неозначеној аритметици на место MSB бита (бит највеће тежине) треба убацити 0. У случају да се дели број представљен у означеној аритметици, MSB бит треба сачувати. Према томе у зависности од представе броја бира се и тип операције за шифтовање. Ако се дели број предсатвљен у неозначеној аритметици користићемо операцију LSR која на место MSB бита убацује нулу. У случају означене аритметике MSB битом је дефинисан предзнак који је треба сачувати, па ће мо користити ASR операцију која не мења MSB бит. Након извршавања било које од ове две операције LSB бит (бит најмање тежине) наћи ће се у CARRY флегу. Ако је број представљен са више бајтова онда се горе наведена анализа односи на бајт највеће тежине. У случају бајтова мање тежине на место MSB бита треба убацити LSB бит који је избачен из бајта веће тежине, а остале бите шифтовати за једно место у десно. За то ће мо искористити операцију ROR. Овај алгоритам се може применити на било који број, независно од броја бајтова са којим је представљен.

Алгоритам за одузимање два двобајтна броја

Процесор MC6800 има инструкције за одузимање (SUBA, SUBB, SBCA, SBCB) али се њима може извршити само одузимање једнобајтних бројева. Ако су

бројеви представљени са више бајтова тада је потребно реализовати програмски код за њихово одузимање коришћењем горе наведених инструкција. Процес одузимања започињемо од бајтова најмање тежине коришћењем инструкције SUB. Уколико је вредност броја (бајта) који одузимамао већа од вредности броја (бајта) од кога се одузима биће сетован CARRY флег који указује на појаву преноса. Да би се избегла потреба програмског праћења о томе да ли је било преноса и да ли је потребно бајт веће тежине броја од кога се одузима умањити за један, за одузимање свих осталих бајтова користићемо инструкцију SBC која то аутоматски ради. Овај алгоритам се може применити на било које бројеве, независно од броја бајтова којима су бројеви представљени.

Решење:

```
$INCLUDE "REGS.ASM"
        ORG MAIN
START LDX #$DBCA    ;пуњење индекс регистра са вредношћу $DBCA
        STX $1000   ;виши бајт индекс регистра се смешта на адресу $1000
                    ;а нижи на адресу $1001 аутоматски
        LSR $1000   ;логички шифт десно убацује 0 на место MSB бита
        ROR $1001   ;на место MSB бита се из CARRY-а убацује вредност LSB
                    ;бита вишег бајта
        LSR $1000
        ROR $1001
        LDAB $1001
        LDAA $1000
        SUBB #$CA
        SBCA #$DB   ;од вишег бајта двобајтног броја се одузима број $DB
                    ;и вредност CARRY флага
        STAB $01    ;смештање нижег бајта резултата на меморијску локацију
                    ;$1
        STAA $00    ;смештање вишег бајта резултата на меморијску локацију
                    ;$0
        WAI         ;даље извршавање програма се прекида до појаве
                    ;интерапта
        BRA START
```

35) Написати програм за шестнаестобитно одузимање. Од броја $2A3B одузети број који се налази на меморијским локацијама $10 (виши бајт) и $11 (нижи бајт) и резултат сместити на меморијске локације $6002 (виши бајт) и $6003 (нижи бајт).

Објашњење:

Да би се решио овај задатак потребно је реализовати алгоритам за:

- одузимање два двобајтна броја.

Детаљно објашњење је дато у задатку 1.

Решење:

```
$INCLUDE "REGS.ASM"
        ORG MAIN
START LDAB #$3B   ;пуњење акомулатора B са нижим бајтом броја од кога се
                  ;врши одузимање
      LDAA #$2A   ;пуњење акомулатора A са вишим бајтом броја од кога се
                  ;врши одузимање
      SUBB $11    ;од акомулатора Б се одузима садржај меморијске
                  ;локације $11
      SBCA $10    ;од акомулатора A се одузима садржај меморијске
                  ; локације $10 и садржај CARRY флага

      STAB $6003  ;смештање нижег бајта резултата на
                  ;меморијску локацију $6003
      STAA $6002  ;смештање вишег бајта резултата на
                  ;меморијску локацију $6002

      WAI         ;даље извршавање програма се прекида до појаве
                  ;интерапта
      BRA START
```

36) Написати програм за шеснестобитно одузимање. Од броја који се налази на меморијским локацијама $F000 (виши бајт) и $F001 (нижи бајт) одузети број $137F и резултат сместити на меморијске локације $F002 (виши бајт) и $F003 (нижи бајт).

Објашњење:

Да би се решио овај задатак потребно је реализовати алгоритам за

- одузимање два двобајтна броја.

Детаљно објашњење је дато у задатку 1.

Решење:

```
$INCLUDE "REGS.ASM"
        ORG MAIN
START LDAB $F001  ;пуњење акомулатора B са нижим бајтом броја од кога се
                  ;врши одузимање. Нижи бајт је на адреси $F001
      LDAA $F000  ;пуњење акомулатора A са вишим бајтом броја од кога се
                  ;врши одузимање. Виши бајт је на адреси $F000
      SUBB #$7F   ;од акумулатора B се одузима нижи бајт броја $137F

      SBCA #$13   ;од акумулатора A се одузима виши бајт броја $137F
                  ;и садржај CARRY флага

      STAB $F003  ;смештање нижег бајта резултата на
                  ;меморијску локацију $F003
      STAA $F002  ;смештање вишег бајта резултата на
                  ;меморијску локацију $F002

      WAI         ;даље извршавање програма се прекида до појаве
                  ;интерапта
      BRA START
```

37) Написати програм за шеснестобитно сабирање. Броју који се налази на меморијским локацијама $0 (виши бајт) и $1 (нижи бајт) додати број $ABCF и резултат сместити на меморијске локације $2 (виши бајт) и $3 (нижи бајт).

Објашњење:

Да би се решио овај задатак потребно је реализовати алгоритам за:

- сабирање два двобајтна броја.

Алгоритам за сабирање два двобајтна броја

Процесор MC6800 има инструкције за сабирање (ADDA, ADDB, ADCA, ADCB) али се њима може извршити само сабирање једнобајтних бројева. Ако су бројеви представљени са више бајтова тада је потребно реализовати програмски код за њихово сабирање коришћењем горе наведених инструкција. Процес сабирања започињемо од бајтова најмање тежине коришћењем инструкције ADD. Уколико је вредност збира бајтова већа од $FF биће сетован CARRY флаг који указује на појаву преноса. Да би се избегла потреба програмског праћења да ли је било преноса и да ли је потребно бајт веће тежине броја на који се додаје повећати за један, за сабирање свих осталих бајтова користити се инструкција ADCA која то аутоматски ради. Овај алгоритам се може применити на било које бројеве, независно од броја бајтова којима су бројеви представљени.

Решење:

```
$INCLUDE "REGS.ASM"
        ORG MAIN
START LDAB $01     ;пуњење акомулатора B са нижим бајтом броја са којим се
                   ;врши сабирање. Нижи бајт је на адреси $01
        LDAA $00   ;пуњење акомулатора A са вишим бајтом броја са којим се
                   ;врши сабирање. Виши бајт је на адреси $00
        ADDB #$CF  ;акомулатору B се додаје нижи бајт броја $ABCF
        ADCA #$AB  ;акомулатору A се додаје виши бајт броја $ ABCF
                   ;и садржај CARRY флага

        STAB $03   ;смештање нижег бајта резултата на
                   ;меморијску локацију $03
        STAA $02   ;смештање вишег бајта резултата на
```

```
                ;меморијску локацију $02

        WAI     ;даље извршавање програма се прекида до појаве
                ;интерапта
        BRA START
```

38) Написати програм за шеснестобитно сабирање. Броју $F10 додати број који се налази на меморијским локацијама $FF (виши бајт) и $100 (нижи бајт) и резултат сместити на меморијске локације $FFF (виши бајт) и $1000 (нижи бајт).

Објашњење:

Да би се решио овај задатак потребно је реализовати алгоритам за:

- сабирање два двобајтна броја.

Детаљно објашњење о реализацији овог алгоритма дато је у задатку 4.

Решење:

```
$INCLUDE "REGS.ASM"
        ORG MAIN
START LDAB #$10 ;пуњење акомулатора B са нижим бајтом броја $F10
                ;са којим се врши сабирање.
        LDAA $0F    ;пуњење акомулатора A са вишим бајтом броја $F10
                    ;са којим се врши сабирање.
        ADDB $100   ;акомулатору B се додаје нижи бајт броја који се налази
                    ;на меморијској локацији $100
        ADCA $FF    ;акомулатору A се додаје виши бајт броја који се налази
                    ;на меморијској локацији $FF  и садржај CARRY флага
        STAB $1000  ;смештање нижег бајта резултата на
                    ;меморијску локацију $1000
        STAA $FFF   ;смештање вишег бајта резултата на
                    ;меморијску локацију $FFF

        WAI         ;даље извршавање програма се прекида до појаве
                    ;интерапта
        BRA START
```

39) Садржају индекс регистра додати број $123. Као помоћне локације при израчунавању користити меморијске локације на адреси $F0 и $F1.

Објашњење:

Да би се решио овај задатак потребно је реализовати алгоритам за сабирање два двобајтна броја који је објашњен у задатку 1d. Микропроцесор MC6800 у свом инструкционом сету нема аритметичке операције за рад са шеснестобитним регистрима. Због тога се садржај индекс регистра смешта на помоћне меморијске локације у којима се врши двобајтно сабирање. Резултат сабирања се поново учитава у индекс регистар.

Решење:

```
$INCLUDE "REGS.ASM"
        ORG MAIN
START STX $F0       ;смештање садржаја индекс регистра на помоћне локације
                    ;виши бајт индекс регистра се смешта на адресу $F0
                    ;нижи бајт индекс регистра се смешта на адресу $F1
        LDAB $F1    ;пуњење акомулатора B са нижим бајтом броја са којим се
                    ;врши сабирање. Нижи бајт је на адреси $01
        LDAA $F0    ;пуњење акомулатора A са вишим бајтом броја са којим се
                    ;врши сабирање. Виши бајт је на адреси $00
        ADDB #$23   ;акомулатору B се додаје нижи бајт броја $123
        ADCA #$AB   ;акомулатору A се додаје виши бајт броја $123
                    ;и садржај CARRY флага

        STAB $F1    ;смештање нижег бајта резултата на
                    ;меморијску локацију $F1
        STAA $F0    ;смештање вишег бајта резултата на
                    ;меморијску локацију $F0
        LDX $F0     ;резултат се смешта у индекс регистар
        WAI         ;даље извршавање програма се прекида до појаве
                    ;интерапта
        BRA START
```

40) Написати програм за дељење шеснаестобитног броја са 8. Број $ABCD (представљен у неозначеној аритметици) сместити на меморијске локације $F000 (виши бајт) и $F001 (нижи бајт) и поделити га са 8, а резултат сместити на меморијске локације $F002 (виши бајт) и $F003 (нижи бајт).

Објашњење:

Да би се решио овај задатак потребно је реализовати алгоритам за:

- дељење двобајтног броја

Детаљно објашњење за реализацију овог алгоритма је дато у задатку 1. Да би смо број поделили са 8 потребно је померити све битове шеснаестобитног броја за три места у десно. То значи да би требало три пута поновити секвенцу:

```
LSR
ROR
```

Ако је потребно неки блок инструкција понављати већи број пута, тада се због уштеде меморијског простора и боље прегледности програма користе програмске петље. За разлику од решења датог у задатку 1а где се горе наведени блок инструкција понавља у коду два пута овде ће мо дати решење са применом програмске петље.

Решење:

```
$INCLUDE "REGS.ASM"
        ORG MAIN
START LDX #$ABCD ;пуњење индекс регистра са вредношћу $ABCD
        STX $F000   ;виши бајт индекс регистра се смешта на адресу $F000 а
                    ;нижи на адресу $F001 аутоматски
        LDAB #$3    ;акомулатор B се користи као програмски бројач којим
                    ;се дефинише број понављања

PON     LSR $F000   ;логички шифт десно убацује 0 на место MSB бита
        ROR $F001   ;на место MSB бита се из CARRY-а убацује убацује
                    ;вредност LSB бита вишег бајта
        DECB        ;декрементирање програмског бројача
```

```
BNE PON     ;ако програмски бројач није стигао до нуле понови блок
            ;инструкција почевши од лабеле PON
LDX $F000   ;узимање резултата у индекс регистар
STX $F002   ;смештање резултата на меморијске локације $F002 и
            ;$F003

WAI         ;даље извршавање програма се прекида до појаве
            ;интерапта
BRA START
```

41) Написати програм за множење шеснаестобитног броја са 4. Број $1234 сместити на меморијске локације $100 (виши бајт) и $101 (нижи бајт) и помножити га са 4 а резултат сместити на меморијске локације $0 (виши бајт) и $1 (нижи бајт).

Објашњење:

Да би се решио овај задатак потребно је реализовати алгоритам за:

- множење двобајтног броја

Алгоритам за множење двобајтног броја

У овом задатку имамо специјалан случај множења. Двобајтни број се множи са 2 на n (у нашем случају n је 2, 2 на 2 је 4). За овај специјални случај није потребно реализовати генерални алгоритам за множење већ се може применити особина да један шифт у лево одговара множењу са 2. Аналогно овоме два шифта у лево би одговарала множењу са 4, три шифта у лево множењу са 8 итд. Померање битова шеснестобитног броја започињемо од бајта најмање тежине. На место LSB бита треба убацити 0 а све остале бите померити за једно место (бит б0 треба да дође на место бита б1,бит б1 на место бита б2 итд. док ће бит б7 прећи у CARRY флаг). Цео овај процес обавља инструкција ASL. Сада на сличан начин треба извршити померање битова унутар бајта веће тежине, једино што на место LSB бита треба да дође MSB бит бајта мање тежине (овај бит се налази у CARRY флагу). Цео овај процес обавља инструкција ROL. Овај алгоритам се може применити на било који број, независно од броја бајтова са којим је представљен.

Решење:

```
$INCLUDE "REGS.ASM"
        ORG MAIN
START LDX #$1234   ;пуњење индекс регистра са вредношћу $1234
        STX $100   ;виши бајт индекс регистра се смешта на адресу $100 а
                   ;нижи на адресу $101
        ASL $101   ;аритметички шифт лево убацује 0 на место LSB бита
        ROL $100   ;на место LSB бита се из CARRY-а убацује вредност MSB
                   ;бита бајта мање тежине
        ASL $101
        ROL $100
        LDX $100   ;узимање резултата у индекс регистар
        STX $0     ;смештање резултата на меморијске локације $00 и $01

        WAI        ;даље извршавање програма се прекида до појаве
                   ;интерапта
        BRA START
```

Задаци за вежбу

42) Написати програм за дељење шеснестобитног броја са 8. Број $C000 сместити на меморијске локације $4000 (виши бајт) и $4001 (нижи бајт) и поделити га са 8 а резултат сместити на меморијске локације $4002 (виши бајт) и $4003 (нижи бајт).

43) Написати програм за множење шеснестобитног броја са 16. Број $C000 сместити на меморијске локације $1000 (виши бајт) и $1001 (нижи бајт) и помножити га са 16, а резултат сместити на меморијске локације $1002 (виши бајт) и $1003 (нижи бајт).

44) Број $A5A5 сместити на меморијске локације $10 (виши бајт) и $11 (нижи бајт). Од њега одузети број $5A5A а резултат сместити у акомулатор A (виши бајт) и акомулатор B (нижи бајт).

45) Написати програм за шеснестобитно одузимање. Од броја који се налази на меморијским локацијама $0 (виши бајт) и $1 (нижи бајт) одузети број који се налази на меморијским локацијама $2 (виши бајт) и $3 (нижи бајт) и резултат сместити на меморијске локације $FE (виши бајт) и $FF (нижи бајт). Добијени резултат помножити са 2.

46) Написати програм за шеснестобитно одузимање. Од броја који се налази на меморијским локацијама $6800 (виши бајт) и $6801 (нижи бајт) одузети број $6800 и резултат сместити на меморијске локације $F0 (виши бајт) и $F1 (нижи бајт). Добијени резултат који третирамо као означени број поделити са четири.

47) Од садржаја показивача стека одузети број $ABC. Као помоћне локације при израчунавању користити меморијске локације на адреси $1000 и $1001.

48) Број $AFFF сместити на меморијске локације $30 (виши бајт) и $31 (нижи бајт). Броју додати број $C00 а резултат сместити у X (индекс регистар).

49) Написати програм за шеснестобитно сабирање. Броју који се налази на меморијским локацијама $1000 (виши бајт) и $1001 (нижи бајт) додати број који се налази на меморијским локацијама $2 (виши бајт) и $3 (нижи бајт) и резултат сместити на меморијске локације $FE (виши бајт) и $FF (нижи бајт). Добијени резултат који третирамо као означени шеснестобитни број подели са 4.

50) Број $4321 сместити на меморијске локације $F0 (виши бајт) и $F1 (нижи бајт) и помножити га са 2. Од добијеног броја одузети број $ABCD а резултат сместити на меморијске локације $0 (виши бајт) и $1 (нижи бајт).

Рад са I/O портовима и реализација маски

51) Претпоставимо да је [ACCA]=37_{16} и [ACCB]=$A2_{16}$. За сваку од следећих инструкција, одредити резултат и вредности N и Z флегова:
а) ANDA #$E3
б) ORAA #$C8
ц) EORB #$A2

Решење: а) Инструкција извршава операцију AND између бајта података E3 и [ACCA]=37. Као резултат се добија [ACCA]=23_{16}=001000011_2. Z флег ће бити 0 пошто резултат није 0; N флег ће бити 0 пошто је бит 7 резултата 0.

б) Инструкција извршава операцију OR између бајта података C8 и [ACCA]=37. Као резултат се добија [ACCA]=FF_{16}=11111111_2. Z флег је 0 и N флег је 1.

ц) Инструкција извршава операцију EX-OR између бајта података A2 и [ACCB]=A2. Као резултат се добија [ACCB]=00_{16}. Z флег је постављен на 1 пошто је резултат 0; N флег је 0.

52) Напиши инструкцијску секвенцу која узима бајт из меморијске локације 06A5, брише MSB, поставља LSB, и враћа у меморију без утицаја на друге битове.

Adresa	Labela	Instr. kod	Mnemonik	Komentar
0200	START	F6	LDAB $06A5	;Напуни податке са адресе
0201			06	;
	;$06A5 у ACCB			
0202			A5	
0203			C4	
	ANDB #$FE		;И (AND) маска за LSB	
0204		FE		
0205			CA	
	ORAB #$80		;ИЛИ (OR) маска за	
0206			80	
	;постављање MSB			
0207			F7	
	STAB $06A5		;Смести податке у	

0208				06
	;меморију			
0209				A5
020A	KRAJ	3E	WAI	;Заустављање

Решење: Инструкцијска секвенца која реализује овај задатака треба: (1) да напуни податак из меморије у један од акумулатора: (2) да изврши И - маскирање акумулатора са 11111110 да би се обрисао LSB, и вратио у меморију без утицаја на друге битове (3) да изврши ИЛИ маскирање акумулатора са 10000000 да би се поставио MSB; и (4) да смести резултат у меморију.

53) Програм надгледа улазни порт да би проверио статус одређених битова. Потребна је инструкција INX да би пребројала колико пута је специфицирани бит има вредност 1. Програм очитава 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 $01A3_{16}$, колико пута је улазни бит имао вредност 1?

Решење: $01A3_{16} = 1 \times 16^2 + 10 \times 16^1 + 3 \times 16^0$

$= 256 + 160 + 3 = 419$

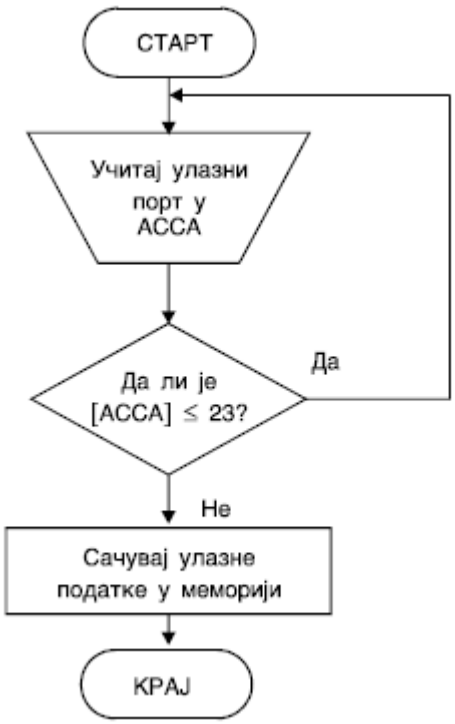
за време извршавања програма, X ће се инкрементирати 419 пута.

54) Написати инструкцијску секвенцу која почиње на адреси 0225, испитује податке са улазног порта са адреси F700, и условљава гранање на адресу 024C само уколико је бит најмање тежине података 1.

Адреса	Лабела	Инстр.код	Мнемоник	Коментар
0225	START	B6	LDAA $F700	; Напуни ACCA са
0226		F7		; улазног порта
0227		00		
0228		44	LSRA	; Помери LSB у C флег
0229		25	BCS $024C	; Гранај се на 024C ако
022A		21		; је C =1
022B		.		; У супротном, настави

Решење: Прва инструкција пуни ACCA подацима са улазног порта. LSRA инструкција помера бит најмање тежине ACCA у C флег. Инструкција BCS проверава да ли је C=1 и одређује да ли ће доћи до гранања.

55) Напиши инструкцијску секвенцу која континуално тестира податке на улазном порту док не постигну вредност већу од $+35_{10}= 23_{16}$, после чега их смешта на локацију 0922 и зауставља се. Улазни порт има адресу FC00 и подаци су бинарни са знаком. Дијаграм тока је:



Дијаграм тока програма

Решење:

Адреса	Лабела	Инстр. код	Мнемоник	Коментар
0200	START	B6	LDAA $FC00	; Напуни податке са порта
0201		FC		; у ACCA
0202		00		

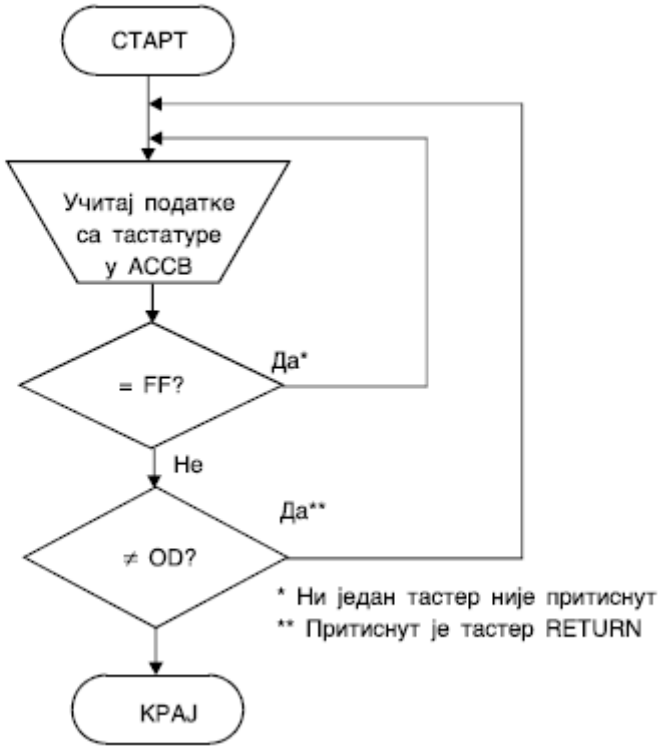
0203		81	CMPA #$23	; Упореди [ACCA] са
0204		23		; $35_{10}=23_{16}$
0205		2F	BLE $0200	; уколико [ACCA]$\leq 23_{16}$ иди
0206		F9		; натраг и понови
0207		B7	STAA $0922	; Ако је [ACCA]>23
0208		09		; смести вредност у меморију
0209		22		
020A	KRAJ	3E	WAI	; Заустављање

Уочавамо како CMPA и BLE инструкције извршавају операцију представљену блоком за одлучивање. BLE се користи зато што су подаци бинарни са знаком. CMPA инструкција не мења [ACCA], тако да ACCA, после извршавања STAA инструкције још увек садржи податке са улазног порта.

56) 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) очитава податке са тастатуре
б) уколико ниједан тастер није притиснут, поново а)
в) уколико је притиснут RETURN тастер зауставља програм
г) Уколико је било који други тастер притиснут, иде натраг на а) и поново очитава тастатуру

Слика 6.8 је дијаграм тока за ову операцију. Подаци са тастатуре су учитани у ACCB. 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 учитао податке. У супротном програм наставља са извршавањем следеће инструкције и зауставља се.



.

Дијаграм тока програма

Решење:

Адреса	Лабела	Инстр. код	Мнемоник	Коментар
0400	START	F6	LDAB $C090	; Учитај податке у ACCB
0401			C0	
0402			90	
0403			C1	CMP #$FF ; Упореди са FF
0404			FF	
0405			27	BEQ $0400 ; Уколико је = гранај
0406			F9	; се да поново учитава
0407			C1	CMPB #$0D ; Упореди са 0D, код за
0408			0D	; RETURN тастер

```
0409                    26          BNE $0400          ; Уколико је ≠
гранај се натраг
040A                    F5                             ; и учитај податке са тастатуре
040Б  KRAJ    3E        WAI         ; У супротном заустави се
```

Анализирати пажљиво програм и уочити како је повезан са дијаграмом тока. Уочити како се инструкције поређења и гранања користе за реализацију блокова за одлучивање.

57) Претпоставимо да је [ACCA]=27_{16}. Одредити операције које инструкцијска секвенца извршава.

```
TSTB
BMI  $0880
BNE $0890
WAI
```

Решење: 27_{16}= 00100111_2, TSTB ће поставити N=0 и Z=0. Због тога BMI инструкција неће изазвати гранање на 0890, а BNE ће изазвати гранање.

58) Написати инструкцијску секвенцу која узима реч податка са улазног порта са адресе $1005 у акумулатор B, инвертује битове 0 и 1 без утицаја на друге битове у бајту податка, а потом акомулатор B смешта на D/A конвертор који се налази на адреси $8031.

Објашњење:

Да би се решио овај задатак потребно је познавати:

- Инструкције за рад са I/O портовима
- Реализацију маски

Инструкције за рад са I/O портовима и реализација маски

Процесор MC6800 у свом инструкционом сету нема посебне инструкције за рад са I/O портовима. Приступ је идентичан као и приступ стандардним меморијским локацијама (RAM, EPROM) коришћењем инструкција (LDA, STA). Примера ради D/A конвертор наведен у задатку се у адресном простору види као меморијска локација на адреси $8031 у коју се само може уписивати, пошто је D/A конвертор на дата бус процесора повезан преко леча. Ова локација се може програмски и читати, али се неће добити вредност уписана у D/A конвертор већ ће бити очитан случајни садржај са дата бус-а који се налази у стању високе импедансе.

- **Реализација маски**

У задатку се тражи да се инвертују битови б1 и б0 без утицаја на друге битове. Пошто је потребно инвертовати битове то одмах упућује да ће мо користити инструкцију EOR. На местима битова које желимо да инвертујемо у маски ће мо поставити јединице а на местима која не желимо да мењамо ће мо поставити нуле.

Маска: $03

б7	б6	б5	б4	б3	б2	б1	б0
0	0	0	0	0	0	1	1
0				**3**			

Решење:

```
$INCLUDE "REGS.ASM"
ORG MAIN
START LDAB $1005  ;узимање садржаја са порта у акомулатор B
EORB #$03         ;инвертовање битова б0 и б1 без утицаја на друге битове
      STAB $8031  ;смештање садржаја акомулатора B на D/A конвертор
      BRA         START
```

59) Написати инструкцију за инвертовање битова б5 и б1 у акумулатору A.

Објашњење:

Да би се решио овај задатак потребно је познавати:

- Реализацију маски

Детаљно објашњење је дато у задатку 2а.

Маска: $22

б7	б6	б5	б4	б3	б2	б1	б0
0	0	1	0	0	0	1	0
2				**2**			

Решење: EORA #$22

60) Написати инструкцију за ресетовање битова б7 и б3 у акумулатору В.

Објашњење:

Да би се решио овај задатак потребно је познавати:

- Реализацију маски

Реализација маски

У задатку се тражи да се ресетују битови б7 и б3 без утицаја на друге битове. Пошто је потребно инвертовати битове то одмах упућује да ће мо користити инструкцију AND. На местима битова које желимо да ресетујемо у маски ће мо поставити нуле а на местима која не желимо да мењамо ће мо поставити јединице.

Маска: $77

b7	b6	b5	b4	b3	b2	b1	b0
0	1	1	1	0	1	1	1
7				**7**			

Решење: ANDB #$77

61) Написати инструкцију за ресетовање битова б4 и б0 у акумулатору А.

Објашњење:

Задатак је идентичан са задатком 20 па ће мо дати само решење.

Решење: ANDB #$EE

62) Написати инструкцију за инвертовање доњег нибла акумулатора А.

Објашњење:

Да би се решио овај задатак потребно је познавати:

- Реализацију маски

Детаљно објашњење је дато у задатку 18. У односу на задатак 18 потребно је знати и шта значи термин нибл. Горњи нибл у бајту представљају битови б7,б6,б5 и б4 док доњи нибл представљају битови б3,б2,б1 и б0. Ако ово знамо задатак би могао да гласи:

Написати инструкцију за инвертовање битова б3,б2,б1 и б0 акумулатора А.

Решење: EORA #$0F

Задаци за вежбу

63) Написати инструкцијску секвенцу која учитава реч податка са улазног порта са адресе $1000 у акомулатор А, брише битове 7 и 0 и инвертује битове 4 и 3 без утицаја на друге битове у акомулатору А.

64) Написати инструкцијску секвенцу која, инвертује битове 1 и 5 и ресетује битове 7 и 4 акомулатора А, без утицаја на друге битове у бајту, а потом садржај акомулатора А смешта на излазни порт на адреси $1200.

65) Написати низ инструкција којима се очитава резултат A/D конверзије у акомулатор B (регистар у коме се налази резултат конверзије је на адреси $B100), ресетује најтежи бит акомулатора B, а потом смешта на D/A конвертор који се налази на адреси $B205.

66) Написати низ инструкција којима се очитава стање трестате бафера у акомулатор B(трестате бафер је на адреси $1001), горњи нибл акомулатора B се ресетује, а потом смешта на леч који се налази на адреси $1002.

67) Написати инструкцијску секвенцу која узима у акомулатор А податак са улазног порта на адреси $8000, брише битове 6 и 7 без утицаја на друге битове у бајту податка.

68) Написати инструкцијску секвенцу која, инвертује битове 1 и 5 акомулатора А без утицаја на друге битове у бајту, а потом садржај акомулатора А смешта на излазни порт на адреси $6200.

69) Написати низ инструкција којима се очитава резултат A/D конверзије у акомулатор B (регистар у коме се налази резултат конверзије је на адреси $B100), ресетује најтежи бит акомулатора B, а потом смешта на D/A конвертор који се налази на адреси $B205

70) Написати низ инструкција којима се очитава стање трестате бафера у акомулатор B (трестате бафер је на адреси $1001), горњи нибл акомулатора B се ресетује, а потом смешта на леч који се налази на адреси $1002.

71) Написати низ инструкција којима се ресетују битови 15,7 и 3 индекс регистра, без утицаја на друге битове.

72) Написати низ инструкција којима се инвертују битови 14,8 и 4 индекс регистра, без утицаја на друге битове.

73) Написати низ инструкција којима се сетују битови 12,8,5 и 1 индекс регистра, без утицаја на друге битове.

Програмске петље за временско кашњење

74) Одређени 6800 MPU ради са 1Mhz тактом. Одредити укупно време потребно за извршавање следећег инструкцијског скупа.

PSHA
NOP
NOP
NOP
NOP
PULA

Решење: Време потребно за извршавање инструкција може да се нађе у Прилогу I. За извршавање PSHA и PULA инструкције потребна су по четири такт циклуса, а за сваки NOP два такт циклуса. Тако је укупно време за извршавање 4+2+2+2+4=14 такт циклуса = 14μsec.

75) Програм у даљем тексту у петљи за кашњење користи COUNT=40_{16} и једну NOP инструкцију. Израчунати укупно време извршавања. Претпоставити да је такт од 1Mhz.

Адреса	Лабела	Инстр. код	Мнемоник	Коментар
0350	START	86	(2) LDAA #$40	; Пуни [ACCA] са
0351		40		; COUNT=$40_{16}=64_{10}$
0352	LOOP	01	(2) NOP	; Касни два циклуса
0353		4A	(2) DECA	; Декрементира ACCA
0354		26	(4) BNE $0352	; Ако није нула, иди опет
0355		FC		; на LOOP
0356	KRAJ	3E	(9) WAI	; Заустављање

Решење: Број у загради који претходи сваком мнемонику инструкције указује на број такт циклуса који је потребан за извршавање тих инструкција. LDAA инструкција се изводи само једном. NOP, DECA и BNE инструкције се изводе 64 пута (COUNT= 40_{16}= 64_{10}). WAI инструкција се изводи само једном. Укупно време извршавања је:

2 + 64 X (2+2+4) + 9 = 523 такт циклуса = 523μs

76) Променити вредност COUNT у програму из претходног примера да би добили кашњење од прибижно 1ms.

Решење: Пошто је 1ms=1000μs, потребно је

1000 = Ts + (COUNT X Tp)=
11 + (COUNT X 8)
односно

COUNT = 123,625

Најближа цела вредност је $COUNT=124_{10}=7C_{16}$. Користећи $COUNT=7C_{16}$, стварно укупно кашњење ће бити:

11 + (124 X 8) = 10038 μs = 1,0003 ms

77) Променити програм у претходном примеру да би се добило кашњење приближно од 4.5ms.

Решење: Ово кашњење је превише велико да би се једноставно постигло увећавањем COUNT. Са $COUNT=FF_{16}=255_{10}$, кашњење ће бити само

11 + 255 X 8 = 2051μs = 2,051 ms

Можемо увећати укупно кашњење повећањем кашњења унутар петље (Tp) додавањем одговарајућег броја NOP инструкција. Да бисмо добили кашњење од 4.5 ms, потребно је

4500 = 11 + (COUNT X Tp)
или
COUNT X Tp =4489

Уколико одаберемо $COUNT = 250_{10}$, онда је

Tp = 4589/250 ≈ 18 μs

Можемо употребити шест NOP-ова да би смо добили Tp=18μc. Тачно кашњење ће бити

11+ (250 x 18) = 4511μs = 4.511 ms
Промењени програм је:

Адреса	Лабела	Инстр. код	Мнемоник	Коментар
0350	START	86	LDAA #$FA	; Пуни [ACCA] са
0351		FA		; COUNT=$FA_{16}=250_{10}$
0352	LOOP	01	NOP	; шест NOP-а за кашњење
0353		01	NOP	
0354		01	NOP	
0355		01	NOP	
0356		01	NOP	
0357		01	NOP	
0358		4A	DECA	; Декрементирај ACCA
0359		26	BNE $0352	; Ако је нула, гранај се
035A		F7		; назад на LOOP
035B	KRAJ	3E	WAI	; Заустављање

78) а) Одредити [X] за кашње од 2s. б) Одредити максимално кашњење за овај потпрограм.

Решење: а) 2s=2000 ms. Због тога је потребно $[X]=2000_{10}=07D0_{16}$.
б) Максимално кашњење се јавља за $[X]=FFFF_{16}=65536_{10}$ и то је 65536ms =65,536s.

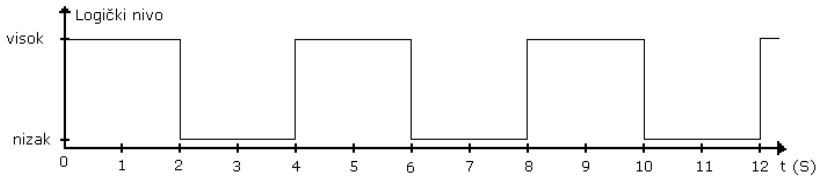
79) Написати програм који генерише правоугли облик сигнала од 50Hz на Q_0 излазу регистра на адреси $8000.

Решење: Q_0 излаз треба да мења стање сваких 10ms, пошто је периода сигнала 20ms када му је учестаност 50Hz.

Програм који следи ће произвести жељени таласни облик. Користићемо излазни регистар као бројач, и инструкцију INC која мења Q_0 биту стање сваких 10ms.

Адреса	Лабела	Инстр. код	Мнемоник	Коментар
0300	START	7F	CLR $8000	; Поставља све излазне
0301		80		; битове на 0
0302		00		
0303	LOOP	7C	INC $8000	; Инкрементирање излазног
0304		80		; регистра
0305		00		
0306		CE	LDX #$000A	; Пуњење X за 10ms
0307		00		; кашњења
0308		0A		
0309		BD	JSR $C800	; Скок на потпрограм
030A		C8		; за кашњење
030B		00		
030C		20	BRA $0303	; Гранање унатраг на петљу
030D		F5		

80) Претпоставити да се подпрограм који генерише кашњење од 200 ms налази на адреси $C600 (потпрограм не мења садржај акумулатора). Написати програм који генерише таласни облик сигнала приказан на слици 1. Таласни облик се генерише на излазним линијама Q0 и Q7 осмобитног леча који се налази на адреси $2100. На осталим излазним линијама треба генерисати инвертовани сигнал.



Сл. 1

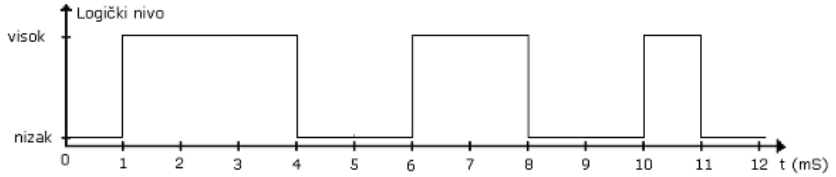
Објашњење:
Задаци овог типа представљају једноставнију варијанту задатака у којима се користе програмске петље за реализацију временског кашњења. Наиме предпоставља се да је петља за кашњење реализована у облику подпрограма, па је потребно само дефинисати колико пута треба позвати подпрограм да би се добило жељено кашњење. Са слике 1 се види да је потребно реализовати кашњење од 2 секунде при чему једно извршавање подпрограма на адреси $C600 траје 200 ms. То значи да је подпрограм потребно позвати 10 пута како би се добило кашњење од 2s. У овој анализи су занемарена времена позива подпрограма и изласка из подпрограма. Због уштеде меморијског простора (EPROM) и прегледности програма, подпрограм позивамо у оквиру програмске петље. Детаљнија објашњења везана за избацивање података на I/O портове могу се наћи у поглављу посвећеном I/O портовима.

Решење:

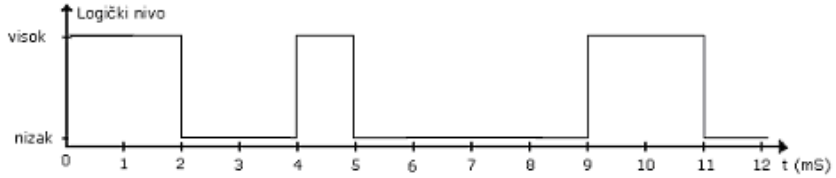
```
$INCLUDE "REGS.ASM"
        ORG MAIN
START LDAA #$81    ;пуњење акомулатора A са податком који у почетном
                   ;тренутку треба избацити на леч
NOVI    STAA $2100 ;избацивање садржаја акомулатора A на леч
        LDAB #!10  ;акомулатор B се користи као програмски бројач којим
                   ;се дефинише број понављања позива подпрограма
                   ;!-означава да се у акомулатор B уписује децимални broj

PON     JSR $C600  ;позив подпрограма за кашњење
        DECB       ;декрементирање програмског бројача
        BNE PON    ;ако програмски бројач није стигао до нуле понови
                   ; позив подпрограма
        COMA       ;инвертујемо садржај акомулатора A
        BRA NOVI   ;идемо на генерисање сигнала за следеће 2s
                   ;Програм се непрекидно врти у овој петљи
```

81) Написати програм који на линији најмање тежине осмобитног леча генерише таласни облик сигнала приказан на слици 2, а на линији највеће тежине таласни облик сигнала приказан на слици 3. Остале линије леча треба држати на високом нивоу (логичко 1). Леч се налази на адреси $6000. Претпоставити да се подпрограм који генерише кашњење од 1 мS налази на адреси $F000.



Сл. 2



Сл. 3.

Објашњење:

Задатак се може решити на више начина. Најједноставније решење је позивање подпрограма онолико пута колико је потребно да би се стигло до временског тренутка у коме треба извршити промену стања на линьама леча. У конкретном задатку промене на линијама леча треба извршити у тачкама: 0, 1, 2, 4, 5, 6, 8, 9, 11 мS. Значи у тренутку покретања програма на леч ће мо избацити податак $FE (линија најмање тежине је на 0 док су све остале линије на 1). Позваћемо подпрограм за кашњење и након тога стижемо у прву тачку у којој треба променити стање на лечу. На леч избацујемо податак $FF. Позивамо подпрограм за кашњење и стижемо до тачке 2мS у којој се опет врши промена стања на излазу леча. У овом тренутку на леч избацујемо $7F. Сада два пута позивамо подпрограм за кашњење да би смо стигли у тачку 4мS у којој се врши промена стања на излазним линијама леча. Поступак настављамо до тачке 11мS у којој на леч избацујемо податак $7E и прекидамо даље извршавање програма. Генералније решење би било ако би се направила програмска петља која се понавља 11 пута. У оквиру петље се позива подпрограм за кашњење и проверава да ли тренутни пролаз одговара некој од тачака у којима се врши промена и ако јесте на леч се избацује одговарајући податак.

У наставку је дато једноставније решење. Покушајте самостално да реализујете сложенији алгоритам.

Решење:

```
$INCLUDE "REGS.ASM"
        ORG MAIN
                        ;тачка 0mS
START LDAA #$FE   ;пуњење акомулатора А са податком који у почетном
                  ;тренутку треба избацити на леч
        STAA $6000  ;избацивање садржаја акомулатора А на леч
        JSR $F000   ;позив подпрограма за кашњење
                        ;тачка 1mS
        LDAA #$FF   ;најнижа линија леча на 1, највиша линија леча на1,
                  ;остале на 1
        STAA $6000  ;избацивање садржаја акомулатора А на леч
        JSR $F000   ;позив подпрограма за кашњење
                        ;тачка 2mS
        LDAA #$7F   ;најнижа линија леча на 1, највиша линија леча на0,
                  ;остале на1
        STAA $6000  ;избацивање садржаја акомулатора А на леч
        JSR $F000   ;позив подпрограма за кашњење
                        ;тачка 3mS
        JSR $F000   ;позив подпрограма за кашњење
                        ;тачка 4mS
        LDAA #$FE   ;најнижа линија леча на 0, највиша линија леча на1,
                  ;остале на1
        STAA $6000  ;избацивање садржаја акомулатора А на леч
        JSR $F000   ;позив подпрограма за кашњење
                        ;тачка 5mS
        LDAA #$7E   ;најнижа линија леча на 0, највиша линија леча на0,
                  ;остале на1
        STAA $6000  ;избацивање садржаја акомулатора А на леч
        JSR $F000   ;позив подпрограма за кашњење
                        ;тачка 6mS
        LDAA #$7F   ;најнижа линија леча на 1, највиша линија леча на0,
                  ;остале на1
        STAA $6000  ;избацивање садржаја акомулатора А на леч
        JSR $F000   ;позив подпрограма за кашњење
                        ;тачка 7mS
```

```
JSR $F000    ;poziv podprograma za kašnjenje
             ;тачка 8mS
LDAA #$7E    ;најнижа линија леча на 0, највиша линија леча на0,
             ;остале на1
STAA$6000    ;избацивање садржаја акомулатора А на леч
JSR $Ф000    ;позив подпрограма за кашњење
             ;тачка 9mS
LDAA #$FE    ;најнижа линија леча на 0, највиша линија леча на1,
             ;остале на1
STAA $6000   ;избацивање садржаја акомулатора А на леч
JSR $F000    ;позив подпрограма за кашњење
             ;тачка 10mS
LDAA #$FF    ;најнижа линија леча на 1, највиша линија леча на1,
             ;остале на1
STAA $6000   ;избацивање садржаја акомулатора А на леч
JSR $F000    ;позив подпрограма за кашњење
             ;тачка 11mS
LDAA #$7E    ;најнижа линија леча на 0, највиша линија леча на0,
             ;остале на1
STAA $6000   ;избацивање садржаја акомулатора А на леч
WAI          ;даље извршавање програма се прекида до појаве
             ;интерапта
BRA          START
```

82) Направити подпрограм за временско кашњење од 78s (инструкциони циклус је 1μS).

Објашњење:

Процесор MC6800 се израђује у две варијанте. Бржа варијанта процесора ради са инструкционим циклусом од 0.5 μS. За извршавање сваке инструкције потребно је неколико инструкционих циклуса (тактова) па је време потребно за извршавање инструкције функција такта са којом процесор ради. Због тога је за израчунавање времена потребног за извршавање неког блока инструкција неопходно знати такт са којим процесором ради. Типично се за генерисање петљи за кашњење користи индекс регистар или један од акомулатора као бројач. Програм остаје у петљи док вредност бројача не дође до нуле. Да би смо израчунали колико пута треба поновити петљу да би се остварило тражено кашњење неопходно је израчунати време

потребно за један пролазак кроз петљу. Предпоставимо да користимо индекс регистар као бројач. Типичан код за временску петљу би био:

```
PET     DEX
        BNE PET
```

Да би смо израчунали време једног проласка кроз петљу потребно је знати колико је потребно циклуса за извршавање сваке од инструкција које се налазе у оквиру петље:

DEX……4 циклуса
BNE……4 циклуса

Значи за један пролазак кроз петљу је потребно 8 циклуса, односно 8μS. Максималан број понављања петље је дефинисан максималним бројем који се може уписати у индекс регистар. Како је индекс регистар шеснестобитни то се у њега може уписати број 65535. На основу овога се може израчунати максимално време кашњења као:

МаксВреме= 8μS * 65535 = 524280 μS

Време је нешто веће од 0,5S што је значајно мање од тражених 78S. Постоји неколико начина како се овај проблем може разрешити.

- Убацивање NOP инструкција у петљу за кашњење како би се време трајања петље продужило
- Убацивање нове петље за кашњење у постојећу
- Додавање нове петље при чему би се посојећа петља за кашњење користила као унутрашња.

Узећемо трећу могућност. Израчунајмо колико пута треба (N) поновити кашњење од ~ 0,5 да би смо добили укупно кашњење од 78S:

N=78000000 μS / 524280 μS= 148.775.

Нисмо добили цео број. То значи да ће постојати грешка у односу на тражено време кашњења. Наиме ми можемо изабрати вредност 148 или 149. Пошто је број ближи 149 усвојимо ову вредност и израчунајмо одступање од жељене вредности.

КАШЊЕЊЕ=149 * 524280 μS= 78117720 μS

Грешка је око 117мS што би се у највећем броју апликација могло узети као прихватљиво.
Покушајмо ипак да добијемо тачније време. Један могући приступ би био да конструишемо унутрашњу петљу за кашњење која ће трајати 500000 μS. У овом случају треба израчунати број којим треба напунити индекс регистар да би се добило тражено кашњење:

Број=500000 μS / 8 μS= 62500

Понављањем ове петље за кашњење 156 (78*2) пута добијамо време које је доста блиско траженом времену од 78S. Наиме нисмо урачунали време потребно за позив подпрограма, време потребно за излазак из подпрограма и пуњење индекс регистра иницијалном вредношћу (62500). Али је грешка мања од 500 μS што је занемарљиво у односу на време од 78S. У случају конструисања временских петљи на нивоу неколико мS о овим временима треба водити рачуна.

Решење:

```
PET78S      LDAA    #!156    ;пуњење акомулатора A децималном вредношћу 156
;спољња петља

SPET        LDX     #!65000  ;пуњење индекс регистра са децималном вредношћу 65000
;унутрашња петља

UPET  DEX
            BNE     UPET
            DECA
            BNE     SPET
            RTS
```

83) Направити подпрограм за временско кашњење од приближно 1S (инструкциони циклус је 1μS). Користећи овај подпрограм у главној програмској петљи реализовати временско кашњење од 17S.

Објашњење:

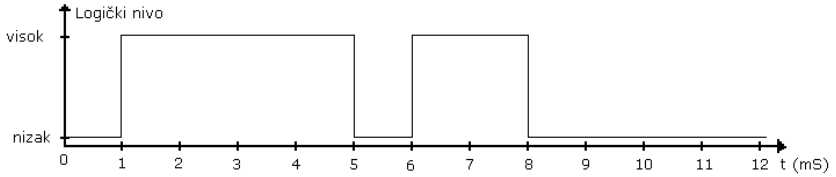
Задатак представља комбинацију претходно анализираних задатака за генерисање програмских петљи за кашњење. Једино ће мо скренути пажњу на начин реализације. Направљен је подпрограм за временско кашњење тако да је време кашњења дефинисано садржајем акомулатора А који се пуни пре позива подпрограма. Искоришћен је подпрограм објашњен у задатку 36. Једина разлика је што се садржај акомулатора А дефинише пре позива подпрограма, а да би се постигло време од 1S два пута се у оквиру подпрограма позива петља за кашњење базирана на индекс регистру као бројачу.

Решење:

```
$INCLUDE "REGS.ASM"
        ORG MAIN
                        ;главни програм
START   LDAA #!17
        JSR PETXS
        WAI             ;даље извршавање програма се прекида до појаве
                        ;интерапта
        BRA START
                        ;подпрограм PETXS
PETX    LDX #!65000     ;пуњење индекс регистра са
                        ;децималном вредношћу 65000
UPET    DEX
        BNE UPET
        LDX #!65000     ;пуњење индекс регистра са
                        ;децималном вредношћу 65000
UPET1   DEX
        BNE UPET1
        DECA
        BNE PETXS
        RTS
```

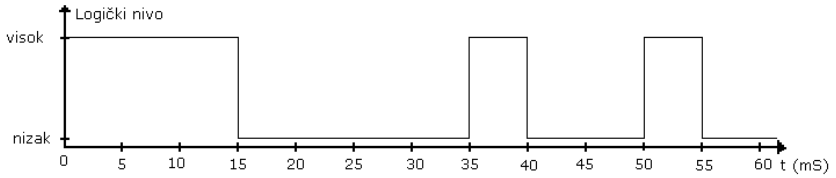
Задаци за вежбу

84) Претпоставити да се подпрограм који генерише кашњење од 0.25 мS налази на адреси $F000. Написати програм који генерише таласни облик сигнала приказан на слици 4. Таласни облик се генерише на излазној линији најмање тежине на осмобитном лечу који се налази на адреси $8000. Остале линије леча треба држати на ниском нивоу (ниво логичке 0).



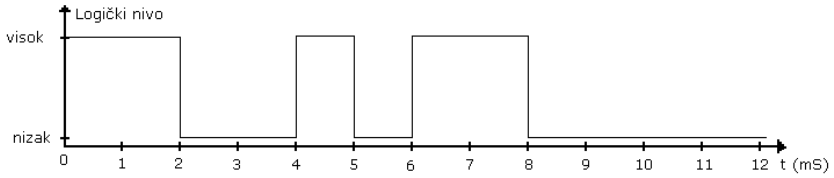
Сл. 4.

85) Претпоставити да се подпрограм који генерише кашњење од 1 мS налази на адреси $A800. Написати програм који генерише таласни облик сигнала приказан на слици 5. Таласни облик се генерише на свим излазним линијама доњег нибла осмобитног леча који се налази на адреси $2100. На излазним линијама горњег нибла треба генерисати инвертовани сигнал.



Сл. 5

86) Претпоставити да се подпрограм који генерише кашњење од 0.2мS налази на адреси $F61A. Написати програм који генерише таласни облик сигнала приказан на слици 6. Таласни облик се генерише на излазним линијама Q7 и Q6 осмобитног леча који се налази на адреси $1800. На осталим линијама леча треба генерисати инвертовани сигнал.



Сл. 6

87) Написати подпрограм за временско кашњење од приближно 0.5S (инструкциони циклус је 1μS). Користећи овај подпрограм у главној програмској петљи реализовати временско кашњење од 7S.

88) Написати подпрограм за временско кашњење од приближно 5мS (инструкциони циклус је 1μS). Користећи овај подпрограм у главној програмској петљи реализовати временско кашњење од 1S.

89) Направити подпрограм за временско кашњење од приближно 5мS (инструкциони циклус је 1μS). Користећи овај подпрограм у главној програмској петљи реализовати временско кашњење од 10S.

Примена индексног адресирања

90) Описати операције које извршава следећа инструкцијска секвенца:

Адреса	Инструкцијски код	Асемблерски језик
0200	CE	LDX #$E000
0201	E0	
0202	00	
0202	AB	ADDA $32,X
0204	32	
0205	A7	STAA X
0206	00	

Решење: Прва инструкција користи непосредно адресирање да би напунила X са вредношћу $E000_{16}$. Друга инструкција користи индексно адресирање да би додала податке акумулатору A. Подаци се узимају са адресе $E000_{16}$+ 32_{16}=$E032_{16}$. Трећа инструкција користи индексно адресирање да би сместила садржај ACCA на адресу $E000_{16}$+00_{16}=$E000_{16}$.

91) Написати програм који узима реч податка са врха стека и смешта је у ACCA без промене сaцаја показивача стека SP

Решење: Нормалан начин да се узму подаци са врха стека и ставе у акумулатор је да се употреби PULA инструкција. PULA инструкција при свом извршавању инкрементира SP пре него што скине податке са стека. Ово се дешава пошто се SP декрементирао када су подаци ставqани на стек.
Други начин да се превази|е претходно описан поступак приказан је следећим програмом.

Адреса	Лабела	Инстр. код	Мнемоник	Коментар
0200	START	30	TSX	; Пренос [SP] у [X]
0201		A6	LDAA $00,X	; Напуни ACCA са адресе
0202		00		; која се налази у X
0203	KRAJ	3E	WAI	; Заустави извршавање

Да би смо анализирали програм, претпоставимо да је [SP] =036B. SP указује на адресу 036B, која је следећа слободна локација на стеку; адреса 036C је врх стека, где је смештен последњи податак који је на стек послат. То је илустровано у даљем тексту.

TSX инструкција преноси [SP] у [X] и затим инкрементира [X]. На крају ове инструкције [X]=036B+1=036B. Инструкција LDAA $00,X користи индексно адресирање да би напунила ACCA са адресе дате са [X] (одступање 00). Пошто је [X] = 036C, подаци узети са ове адресе се смештају у ACCA. Тако су узети подаци са врха стека ($036C) без икаквог утицаја на SP.

		Стек	
[SP] →	036B	????	← врх стека
	036C	подаци	
	036D	подаци	
	036E	подаци	
	036F	подаци	

92) Написати програм за тражење минималног неозначеног броја у меморији. Програмамом се претражује меморијско подручије од $F100 да $F500, укључујући и ове локације. Нађену вредност треба сместити на меморијску локацију $100.

Објашњење:

Бројеви уписани у меморију су осмобитни бројеви. Како је у задатку наведено да су у питању неозначени бројеви то значи да је 0 најмања вредност којом се може представити осмобитни број а $FF максимална вредност. У задатку се тражи да претражимо подручије од $400 (1024) локације. За овако велики број локација је неприхватљиво писање линијског кода, већ претраживање треба радити у петљи коришћењем индексног адресирања. На почетку програма ће мо меморијску локацију $100 напунити $FF (највећи осмобитни број). У петљи ће мо применом индексног адресирања приступати меморијским локацијама у опсегу од $F100 до $F500. Уколико је вредност меморијске локације мања од вредности уписане на локацији $100 на локацију $100 ће мо уписати ову вредност. По изласку из програмске петље на локацији $100 ће бити вреност најмањег броја. При реализацији алгоритма посебну пажњу треба обратити на избор инструкција за условно гранање. Код

процесора MC6800 се разликују инструкције условног гранања у зависности од тога да ли се ради са бројевима представљеним у означеној или неозначеној аритметици.

Решење:

```
$INCLUDE "REGS.ASM"
        ORG MAIN
START LDAA #$FF
                STAA $100   ;пуњење меморијске локације $100 са највећим
                                                        ;неозначеним бројем
                LDX #$F100  ;пуњење индекс регистра почетном адресом подручија које
                                                        ;се претражује
PET             CMPA 0,X            ;упоредјујемо садржај акомулатора А са локацијом на коју
                                                        ;указује индекс регистар
                BLE         VECI                ;ако је садржај акомулатора А мањи од меморијске локације
                                                        ;идемо на следећу меморијску локацију
                LDAA 0,X
                STAA $100
VECI            INX
                CPX         #$F501      ;провера да ли смо стигли до краја подручија које испитујемо
                                                        ;поредимо са адресом + 1
                BNE         PET
        WAI                             ;даље извршавање програма се прекида до појаве интерапта
                BRA         START
```

Задаци за вежбу

93) Написати програм за тражење максималног неозначеног броја у меморији. Програмом се претражује меморијско подручије од $A000 да $A200, укључујући и ове локације. Нађену вредност треба сместити на меморијску локацију $0.

94) Напистати програм за копирање садржаја EPROM-а од локације $F000 до локације $F100 у подручије RAM-а од локације $60 до локације $160.

95) Написати програм који налази суму неозначених бројева смештених у подручије RAM-а од адресе $0 до $A0. Резултат сместити на меморијске локације $FE (виши бајт) и $FF (нижи бајт).

96) Написати програм који тражи податак $A5 у меморији од адресе $100 до адресе $1000. Адресу првог нађеног податка смешта на локације $0 (виши бајт) и $01 (нижи бајт).

97) Написати програм који броји колико у меморији од адресе $100 до адресе $1FF има података чија је вредност већа од $80. Нађени број сместити на локацију $0. Бројеви у меморији су представљени у неозначеној аритметици.

98) Написати програмску петљу која 256 пута очитава резултат A/D конверзије са паралелног осмобитног конвертора који се налази на адреси $8200. Резултате конверзије треба смештатити на узастопне меморијске локације почевши од локације $A0.

Решења задатака

Аритметичке операције
(двобајтно сабирање, одузимање, множење и дељење)

42)

```
$INCLUDE "REGS.ASM"
            ORG MAIN
START       LDX         #$C000      ;пуњење индекс регистра са вредношћу $
C000
            STX  $4000  ;виши бајт индекс регистра се смешта на адресу $4000 а
                        ;на адресу $4001 нижи
                        LDAB #$3                ;колико пута шифтујемо 16-
битни број(дељење са 8-3 шифта)
PET                     LSR         $1000       ;логички шифт десно убацује 0
на место MSB бита
             ROR        $1001       ;на место MSB бита се из CARRY-а убацује
        убацује
                        ;вреднос LSB бита вишег бајта
                        DECB
            BNE         PET
            LDX         $4000       ;узимање шеснаестобитног резултата у X
    регистар
            STX         $4002       ;смештање шеснаестобитног резултата на
    меморијску
                                    ; локацију $4002
                        WAI                     ;даље
извршавање програма се прекида до појаве интерапта
                        BRA         START
```

43)

```
$INCLUDE "REGS.ASM"
            ORG MAIN
```

```
START     LDX     #$C000    ;пуњење индекс регистра са вредношћу $
C000
          STX     $1000     ;виши бајт индекс регистра се смешта на
             адресу $1000 а нижи
                            ;на адресу $1001
          LDAB #$4          ;колико пута шифтујемо 16-
битни број(множење са 16-4 шифта)
PET       ASL     $1000     ;логички шифт лево убацује 0
на место LSB бита
          ROL     $1001     ;на место LSB бита се из CARRY-а убацује
          вреднос MSB
                            ; бита нижег бајта
          DECB
          BNE     PET
          LDX     $1000     ;узимање шеснаестобитног реуултата у X
     регистар
          STX     $1002     ;смештање шеснаестобитног резултата на
     меморијску
                            ; локацију $1002
          WAI                       ;даље
извршавање програма се прекида до појаве интерапта
          BRA     START
```

44)

```
$INCLUDE "REGS.ASM"
          ORG MAIN
START     LDX     #$A5A5    ;пуњење X регистра вредношћу $A5A5
          STX     $10               ;виши бајт индекс регистра се
             смешта на адресу $10 а нижи
                            ;на адресу $11
          LDAB #$5A ;пуњење акомулатора B са нижим бајтом броја који се
                                    ;одузима
          LDAA #$5A ;пуњење акомулатора A са вишим бајтом броја који се
                                    ;одузима
                    SUBB $11        ;од акомулатора B се одузима
садржај меморијске локације
                                    ;$11
```

```
        SBCA $10            ;од акомулатора A се одузима садржај меморијске локације
                                                ;$10 и садржај CARRY флага
                WAI                             ;даље извршавање програма се прекида до појаве интерапта
                BRA     START
```

45)

```
$INCLUDE "REGS.ASM"
        ORG MAIN
START   LDAB $01            ;пуњење акомулатора B са нижим бајтом броја од кога се
                                                ;врши одузимање
        LDAA $0             ;пуњење акомулатора A са вишим бајтом броја од кога се
                                                ;врши одузимање
                SUBB $3             ;од акомулатора B се одузима садржај меморијске локације
                                                ;$3
        SBCA $2             ;од акомулатора A се одузима садржај меморијске локације
                                                ;$2 и садржај CARRY флага
;смештање вишег и нижег бајта резултата на меморијске локације FF и FE
        STAB $FF
                STAA $FE
;множење резултата са 2
                ASL     $FF                     ;логички шифт лево убацује 0 на место LSB бита
                ROL     $FE                     ;на место LSB бита се из CARRY-a убацује убацује
                                                ; вреднос MSB бита нижег бајта
                WAI                             ;даље извршавање програма се прекида до појаве интерапта
                BRA     START
```

46)

```
$INCLUDE "REGS.ASM"
            ORG MAIN
START       LDAB $6801  ;пуњење акомулатора B са нижим бајтом броја од
                                                        ; кога се врши
одузимање
            LDAA $6800  ;пуњење акомулатора A са вишим бајтом броја од кога се
                                                        ;врши одузимање
                        SUBA $68                ;од акомулатора A се одузима
виши бајт шеснаестобитног
                                                        ;броја ($68). Од
акомулатора B треба одузети 0. Пошто се
                                                        ;тиме ништа не
мења ова операција се не ради
;смештање вишег и нижег бајта резултата на меморијске локације
            STAB        $F1
                        STAA        $F0
;дељење резултата са 4
                        ASR         $F0                 ;логички шифт
лево убацује 0 на место LSB бита
             ROR        $F1                 ;на место LSB бита се из
      CARRY-a убацује вреднос MSB
                                                        ; бита нижег бајта

                        ASR         $F0                 ;логички шифт
лево убацује 0 на место LSB бита
             ROR        $F1                 ;на место LSB бита се из
      CARRY-a убацује убацује
                                                        ;вреднос MSB бита нижег бајта

            WAI                                         ;даље извршавање програма
    се прекида до појаве интерапта
                        BRA         START
```

47)

```
$INCLUDE "REGS.ASM"
            ORG MAIN
START       STS         $1000       ;виши бајт стек поинтера се смешта на адресу $1000 а нижи
                                                    ;на адресу $1001
            LDAC #$BC   ;пуњење акомулатора B са нижим бајтом броја који се
                                                    ;одузима
            LDAA #$A    ;пуњење акомулатора A са вишим бајтом броја који се
                                                    ;одузима
                        SUBB $1001  ;од акомулатора B се одузима садржај меморијске локације
                                                    ;$1001
            SBCA $1000  ;од акомулатора A се одузима саџај меморијске локације
                                                    ;$1000  и садржај CARRY флага
;смештање вишег и нижег бајта резултата на меморијске локације
            STAB $1001
                        STAA $1000

                        LDS         $1000       ;узимање резултата у стек поинтер

            WAI                                 ;даље извршавање програма се прекида до појаве интерапта
                        BRA         START


48)

$INCLUDE "REGS.ASM"
            ORG MAIN
START       LDX         #$A5A5      ;пуњење X регистра вредношћу $AFFF
            STX         $30                     ;виши бајт индекс регистра се смешта на адресу $30 а нижи
                                                    ;на адресу $31
            LDAB #$0                ;пуњење акомулатора B са нижим бајтом броја са  којим
                                                    ;се сабира
```

```
        LDAA #$C0     ;пуњење акомулатора А са вишим бајтом броја са  којим
                                                              ;се сабира
                      ADDB $31          ;акомулатору  B  се  додаје
садржај меморијске локације
                                                              ;$31
        ADCA $30              ;акомулатору  А  се  додаје  сaцај  меморијске
    локације
                                                              ;$30  и  садржај
CARRY флага
;смештање вишег и нижег бајта резултата на меморијске локације
        STAB $31
                      STAA $30

        LDX           $30                 ;узимање  резултата  у  индекс
    регистар

        WAI                               ;даље  извршавање  програма
    се прекида до појаве интерапта
                      BRA           START
```

49)

```
$INCLUDE "REGS.ASM"
        ORG MAIN
START     LDAB $1001  ;пуњење акомулатора B са нижим бајтом броја од кога се
                                                              ;врши одузимање
        LDAA $1000    ;пуњење акомулатора А са вишим бајтом броја од кога се
                                                              ;врши одузимање
                      ADDB $3             ;акомулатору А се додаје нижи
бајт шеснаестобитног који
                                                              ;се  налази  на
адреси $3
        ADCA $2               ;акомулатору  А  се  додаје  сaцај  меморијске
    локације
                                                              ;$2  на  којој  се
налзи виши бајт шеснаестобитног броја
                                  ;и садржај CARRY флага
```

```
;смештање вишег и нижег бајта резултата на меморијске локације
            STAB        $FF
                        STAA        $FE

;дељење резултата са 4
                        ASR         $FE                 ;логички шифт
лево убацује 0 на место LSB бита
             ROR        $FF                 ;на место LSB бита се из
       CARRY-a убацује вреднос MSB
                                            ; бита нижег бајта

                        ASR         $FE                 ;логички шифт
лево убацује 0 на место LSB бита
             ROR        $FF                 ;на место LSB бита се из
       CARRY-a убацује вреднос MSB
                                            ; бита нижег бајта

            WAI                             ;даље извршавање програма
    се прекида до појаве интерапта
                        BRA         START
```

50)

```
$INCLUDE "REGS.ASM"
            ORG MAIN
START        LDX        #$4321      ;пуњење X регистра вредношћу $4321
            STX         $F0                 ;виши бајт индекс регистра се
               смешта на адресу $F0 а нижи
                                            ;на адресу $F1

;множење резултата са 2
                        ASL         $F1                 ;логички шифт
лево убацује 0 на место LSB бита
             ROL        $F0                 ;на место LSB бита се из
       CARRY-a убацује вреднос MSB
                                            ; бита нижег бајта
            LDAB #$CD   ;пуњење акомулатора B са нижим бајтом броја који се
                                            ;одузима
```

```
        LDAA #$AB     ;пуњење акомулатора А са вишим бајтом броја који се
                                                              ;одузима
                      SUBB $F1                ;од акомулатора В се одузима
садржај меморијске локације
                                                              ;$F1
        SBCA $F0                ;од акомулатора А се одузима садржај
    меморијске локације
                                                              ;$F0 и садржај
CARRY флага
;смештање вишег и нижег бајта резултата на меморијске локације

        STAB $F1
                      STAA $F0

                      WAI                                     ;даље
извршавање програма се прекида до појаве интерапта
                      BRA       START
```

Рад са I/O портовима и реализација маски

63)

```
        LDAA      $1000
                  ANDA      #$7E
                  EORA      #$14
```

64)

```
                  EORA #$22
                  ANDA      #$6F
                  STAA      $1200
```

65)

```
        LDAB    $B100
        ANDB    #$7F
        STAB    $B205
```

66)

```
        LDAB    $1001
        ANDB    #$0F
        STAB    $1002
```

67)

```
        LDAA    $8000
        ANDA #$3F
```

68)

```
        EORA     #$22
        STAA    $6200
```

69)

```
        LDAB    $B100
        ANDB    #$7F
        STAB    $B205
```

70)

```
        LDAB    $1001
        ANDB    #$0F
        STAB    $1002
```

71)

```
        STX             $100    ;као помоћну
```

локацију за смештање X-а смо произвољно

;изабрали

локације $100 и $101. На локацију $100 смештамо

адресу $101 нижи бајт X-а. У инструкционом MC6800 не постоје логичке инструкције за шеснаестобитним регистрима, због тога садржај X-а меморију и реализујемо маске за виши и нижи резултат враћамо у X

```
LDAA    $100    ;виши бајт ,а на
LDAB    $101    ;сету процесора
ANDA    #$7F    ;рад са
ANDB    #$77    ;смештамо у
STAA    $100    ;бајт и након тога
STAB    $101
LDX             $100
```

72)

```
STX             $100
LDAA    $100
LDAB    $101
EORA    #$41
EORB    #$10
STAA    $100
STAB    $101
LDX             $100
```

73)

```
STX             $100
LDAA    $100
LDAB    $101
ORAA    #$11
ORAB    #$22
STAA    $100
STAB    $101
LDX             $100
```

Програмске петље за временско кашњење

84)

```
$INCLUDE "REGS.ASM"
            ORG         MAIN
START       LDAB #$0
                        STAB $8000  ;избацујемо 0 на леч
                        LDAA #$4
P1MS        JSR $F000               ;четири пута позивамо подпрограм
                    ;за кашњење 4*0.25мS=1мS
                        DECA
                        BNE         P1MS
                        LDAB #$1
                        STAB $8000  ;избацујемо 1 на леч
                        LDAA #$16
P4MS        JSR $F000               ;16 пута позивамо подпрограм
                    ;за кашњење 16*0.25мS=4мS
                        DECA
                        BNE         P4MS
            LDAB #$0
                        STAB $8000  ;избацујемо 0 на леч
                        LDAA #$4
P1MS2       JSR         $F000       ;четири пута позивамо подпрограм
                    ;за кашњење 4*0.25мS=1мS
                        DECA
                        BNE         P1MS2
            LDAB #$1
                        STAB $8000  ;избацујемо 1 на леч
                        LDAA #$8
P2MS        JSR         $F000       ;осам пута позивамо подпрограм
                    ;за кашњење 8*0.25мS=2мS
                        DECA
                        BNE         P2MS
            LDAB #$0
                        STAB $8000  ;избацујемо 0 на леч
            WAI                     ;даље извршавање програма се прекида до
    појаве интерапта
```

```
                    BRA         START
```

85)

```
$INCLUDE "REGS.ASM"
          ORG MAIN
START     LDAB #$0F
                    STAB $2100  ;избацујемо садржај акомулатора B на леч
                    LDAA #!15
P15MS     JSR       $A800       ;15 пута позивамо подпрограм
                      ;за кашњење 15* 1мS=15мS
                    DECA
                    BNE         P15MS
                    COMB                    ;инвертујемо    садржај
акомулатора B
                    СТАБ $2100  ;избацујемо садржај акомулатора B на леч
                    LDAA #!20
P20MS     JSR       $ A800      ;20 пута позивамо подпрограм
                      ;за кашњење 20 * 1мS=20мS
                    DECA
                    BNE         P20MS
          COMB                  ; инвертујемо садржај акомулатора B
                    STAB $2100
                    LDAA #$4
P1MS2     JSR       $F000       ;четири пута позивамо подпрограм
                      ;за кашњење 4*0.25мS=1мS
                    DECA
                    BNE         P1MS2
          LDAB #$1
                    STAB $8000  ;избацујемо 1 на леч
                    LDAA #$8
P2MS      JSR       $F000       ;осам пута позивамо подпрограм
                      ;за кашњење 8*0.25мS=2мS
                    DECA
                    BNE         P2MS
          LDAB #$0
                    STAB $8000  ;избацујемо 0 на леч
```

```
            WAI                                 ;даље извршавање програма
    се прекида до појаве интерапта
                        BRA         START
```

86)

```
$INCLUDE "REGS.ASM"
            ORG MAIN
START       LDAA !100
                        STAA  $40     ;Помћна локација на коју се смешта бројач
                           ;пролазака кроз петљу
                        LDAA #$C0
                        STAA $1800
PET1          JSR $F61A
            DEC  $40
                        BNE PET1
            LDAA !100
                        STAA  $40     ;Помоћна локација на коју се смешта бројач
                           ;пролазака кроз петљу
                        CLRA
                        STAA $1800
PET2          JSR $F61A
            DEC  $40
                        BNE PET2
            LDAA !50
                        STAA  $40     ;Помоћна локација на коју се смешта бројач
;пролазака кроз петљу

                        LDAA #$C0
                        STAA $1800
PET3          JSR $F61A
            DEC  $40
                        BNE PET3
            LDAA !50
                        STAA  $40                 ;Помоћна локација на коју се
смешта бројач
                           ;пролазака кроз петљу
```

```
                        CLRA
                        STAA $1800
PET4        JSR $F61A
            DEC  $40
                        BNE PET4
            LDAA !100
                        STAA  $40                ;Помоћна локација на коју се
смешта бројач
                            ;пролазака кроз петљу

                        LDAA #$C0
                        STAA $1800
PET5        JSR $F61A
            DEC  $40
                        BNE PET5
            CLRA
                        STAA $1800
                        BRA START
```

87)

```
$INCLUDE "REGS.ASM"
            ORG MAIN
START       LDAA !14
PET05       JSR KAS05
            DECA
            BNE PET05
            BRA START
KAS05       LDX !62500
PET                     DEX
                        BNE PET
                        RTS
                        END
```

88)

```
$INCLUDE "REGS.ASM"
            ORG MAIN
START       LDAA !200
```

```
PET05       JSR KAS05
            DECA
            BNE PET05
            BRA START
KAS05       LDX !625
PET                     DEX
                        BNE PET
                        RTS
                        END
```

89)

```
$INCLUDE "REGS.ASM"
            ORG MAIN
START       LDAA !200
                        LDAB !10
PET05       JSR KAS05
            DECA
            BNE PET05
            DECB
            BNE PET05
            BRA START
KAS05       LDX !625
PET                     DEX
                        BNE PET
                        RTS
                        END
```

93)

```
$INCLUDE "REGS.ASM"
            ORG MAIN
START       LDAA $A000
                        LDX #$A000
PET                     CMPA 0,X
                        BLS NIJE
                        LDAA 0,X
NIJE                    INX
                        CPX #$A201
                        BNE PET
                        BRA START
                        END
```

94)

```
$INCLUDE "REGS.ASM"
            ORG MAIN
                        LDX #$F000
                        STX $40
                        LDX #$60
                        STX $42
PET         LDX $40
                        LDAA 0,X
                        INX
                        STX $40
                        LDX $42
                        STAA 0,X
                        INX
                        STX $42
                        CPX #$161
                        BNE PET
                        END
```

95)

```
$INCLUDE "REGS.ASM"
            ORG MAIN
            LDX #$0
            STX $FE
PET                     LDAA 0,X
                        ADDA $FF
                        STAA $FF
                        LDAA #0
                        ADCA $FE
                        STAA $FE
                        INX
                        CPX #$A1
                        BNE PET
                        END
```

96)

```
$INCLUDE "REGS.ASM"
            ORG MAIN
START        LDX #$FFFF
                        STXX $0
            LDX #$100
PET                     LDAA 0,X
                        CMPA #$A5
                        BNE NAS
                        STX $0
                        BRA KRAJ
NAS                     INX
                        CPX #$1001
                        BNE PET
KRAJ         BRA START
                        END
```

97)

```
$INCLUDE "REGS.ASM"
            ORG MAIN
```

```
START       CLR $0
                        LDX #$100
PET                     LDAA 0,X
                        CMPA #$80
                        BLS NIJE
                        INC $0
NIJE                    INX
                        CPX #$200
                        BNE PET
                        BRA START
                        END
```

98)

```
$INCLUDE "REGS.ASM"
            ORG MAIN
START       LDX #$A0
PET                     LDAA $8200
                        STAA 0,X
                        INX
                        CPX #$1A0
                        BNE PET
                        BRA START
                        END
```

ПРИЛОГ I - КОМПЛЕТАН ИНСТРУКЦИЈСКИ СКУП 6800

У овом поглављу ће бити приказан преглед свих инструкција 6800. За сваку инструкцију, биће дат: 1)текстуални опис, 2)симболичка ознака, 3)табела која показује мнемоничку ознаку инструкције, начин адресирања, оп. код, број такт циклуса потребних за извршавање, број бајтова оп. кода, и 4)ознаке како инструкција утиче на различите флегове.

У даљем тексту је листа номенклатура које се користе у овом опису:

A —	Садржај акумулатора A
B —	Садржај акумулатора Б
X —	Садржај индекс регистра X
XH/XL —	Виши/нижи бајт регистра X
PC —	Садржај програмског бројача
PCH/PCL—	Виши/нижи бајт регистра PC
SP —	Садржај стек показивача
SPH/SPL—	Виши/нижи бајт SP
CCR —	Регистар услова
(M) —	Садржај меморијске локације која је дата адресом операнда
(M+1) —	Садржај меморијске локације која је дата адресом операнда + 1
(M_{SP}) —	Садржај меморијске локације на коју указује SP
00 —	Бајт једнак нули
0 —	Бит једнак
$\rightarrow$	Пренос у
$\oplus$	Ексклузивно OR операција

Листа симбола који се користе са статусним флеговима је

0 —	Флег је обрисан на 0
1 —	Флег је постављен на 1
x —	Флег је постављен или обрисан у зависности од резултата операције
a), b)...	Погледај ознаке а, б, итд.

ABA: (*Add B to A*)

Додаје се садржај B на садржај A и резултат смешта у A.

$$\mathbf{B + A \rightarrow A}$$

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
ABA	Имплицитно	18	2	1	x		x	x	x	x

ADC: *(Add Memory and Carry to Accumulator)*

Додаје се податак из меморије и C флег на садржај акумулатора A или B. Резултат се смешта у акумулатор.

(M) + C + B→B ili (M) + C + B→B

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
ADCA	Непосредно	89	2	2	x		x	x	x	x
	Директно	99	3	2	x		x	x	x	x
	Индексирано	A9	5	2	x		x	x	x	x
	Проширено	B9	4	3	x		x	x	x	x
ADCB	Непосредно	C9	2	2	x		x	x	x	x
	Директно	D9	3	2	x		x	x	x	x
	Индексирано	E9	5	2	x		x	x	x	x
	Проширено	F9	4	3	x		x	x	x	x

ADD: *(Add Memory to Accumulator)*

Додаје се податак из меморије садржају акумулатора A или B. Резултат се смешта у акумулатор.

(M) + A → A ili (M) + B→B

	Начин	Оп.	Број	Број

Мнемоник	адресирања	код	циклуса	бајтова	H	I	N	Z	V	C
ADDA	Непосредно	8B	2	2	x		x	x	x	x
	Директно	9B	3	2	x		x	x	x	x
	Индексирано	AB	5	2	x		x	x	x	x
	Проширено	BB	4	3	x		x	x	x	x
ADDB	Непосредно	CB	2	2	x		x	x	x	x
	Директно	DB	3	2	x		x	x	x	x
	Индексирано	EB	5	2	x		x	x	x	x
	Проширено	FB	4	3	x		x	x	x	x

AND: *Logical AND Memory with Accumulator*

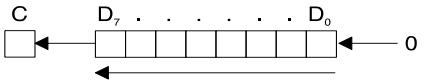
Логичка И операција се изводи бит по бит, између садржаја мемориске локације и акумулатора А или В, а резултат остаје у акумулатору.

(M)•A→A ili (M)•B→B

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
ANDA	Непосредно	84	2	2			x	x	0	
	Директно	94	3	2			x	x	0	
	Индексирано	A4	5	2			x	x	0	
	Проширено	B4	4	3			x	x	0	
ANDB	Непосредно	C4	2	2			x	x	0	
	Директно	D4	3	2			x	x	0	
	Индексирано	E4	5	2			x	x	0	
	Проширено	F4	4	3			x	x	0	

ASL: Arithmetic Shift Left

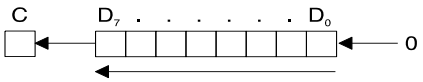
Извршава померање садржаја меморије улево за један бит. Бит D_7 помера се у C флег, а 0 се помера у бит D_0.



Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
ASL	Индексирано	68	7	2			x	x		x
	Проширено	78	6	3			x	x		

ASLA/ASLB: *Shift Accumulator Data Left One Bit*

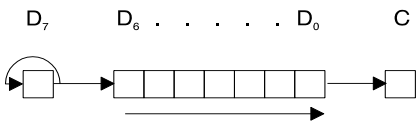
Извршва се померање садржаја акумулатора A или B за један бит у лево. Бит D_7 се помера у C флег, а 0 у D_0



Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
ASLA	Имплицитно	48	2	1			x	x		x
ASLB	Имплицитно	58	2	1			x	x		x

ASR: *Shift Memory Data Right One Bit (with Bit 7 Unchanged)*

Битови D_6-D_0, бајта податка смештеног у специфицираној меморијској локацији, померају се за један бит удесно. Бит D_7 (бит знака) остаје непромењен, а бит D_0 се помера у C флег.



За бајт података 10110101 биће:

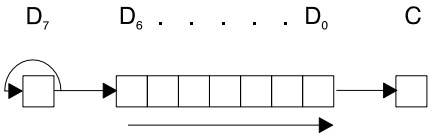
	Подаци из меморије	*С заставица*
Пре ASR	10110101	X
После ASR	11011010	1

Уочимо да се бит D_7 није променио, и да је бит D_0 пренет у C

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
ASR	Индексирано	67	7	2			x	x		x
	Проширено	77	6	3			x	x		x

ASRA/ASRB: *Shift Accumulator Data Right One Bit (except Sign Bit)*

Битови D_6-D_0, бајта податка смештеног у акумулатору, померају се за један бит удесно. Бит D_7 (бит знака) остаје непромењен, а бит D_0 се помера у C флег.



Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
ASRA	Имплицитно	47	2	1			x	x		x
ASRB	Имплицитно	57	2	1			x	x		x

BCC: *Branch If Carry Flag Cleared*

Грана се на адресу која се одређује додавањем бајта одступања на PC, само уколико је C=0. У супротном, наставља се са извршавањем следеће инструкције у секвенци инструкција.

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C

BCC	Релативно	24	4	2							

BCS: Branch If Carry Flag is Set

Грана се на адресу која се одређује додавањем бајта одступања на PC, само ако је C=1. У супротном, наставља се са извршавањем следеће инструкције у секвенци.

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
BCS	Релативно	25	4	2						

BEQ: *Branch if Result Equal to Zero*

Грана се на адресу која се одређује додавањем бајта одступања на PC, само ако је резултат једнак нули (Z=1). У супротном, наставља се са извршавањем следеће инструкције у секвенци.

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
BEQ	Релативно	27	4	2						

BGE: *Branch if Result is Greater Than or Equal to Zero- Signed numbers*

Грана се на адресу која се одређује додавањем бајта одступања на PC, само ако је резултат већи или једнак нули. У супротном, наставља се са извршавањем следеће инструкције у секвенци.

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
BGE	Релативно	2C	4	2						

BGT: *Branch if Result Greater Than Zero - Signed numbers*

Грана се на адресу која се одређује додавањем бајта одступања на PC, само ако је резултат већи од нуле. У супротном, наставља се са извршавањем следеће инструкције у секвенци. Резултат је бинарни број са знаком, а другим комплементом су представљени негативни бројеви. Ова инструкција се разликује од BPL инструкције по томе што дозвољава премашење; дозволиће да дође до гранања чак и ако је резултат >+127.

MPU тестира Z, N и V флегове. Гранање се јавља уколико је Z=0 и N=V

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
BGT	Релативно	2E	4	2						

BHI: *Branch if Result Higher Than Zero - Unsigned numbers*

Грана се на адресу која се одређује додавањем бајта одступања на PC, само ако је резултат већи од нуле. У супротном, наставља се са извршавањем следеће инструкције у секвенци. Резултат се третира као број без знака.

MPU тестира Z, и C флегове. Гранање се јавља уколико је Z=0 и C=0.

Ово се разликује од BGT инструкције по томе што BHI оперише са бинарним бројевима без знака, док BGT оперише са бинарним бројевима са знаком.

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
BHI	Релативно	22	4	2						

BIT: *Bit Test*

Логичка И операција се изводи бит по бит, између садржаја мемориске локације и акумулатора A или B, али резултат не остаје у
акумулатору. Резултат утиче само на Z и N флегове; садржај акумулатора остаје неизмењен.

(M) ● A ili **(M) ● B**

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
BITA	Непосредно	85	2	2			x	x	0	
	Директно	95	3	2			x	x	0	
	Индексирано	A5	5	2			x	x	0	
	Проширено	B5	4	3			x	x	0	
BITB	Непосредно	C5	2	2			x	x	0	
	Директно	D5	3	2			x	x	0	
	Индексирано	E5	5	2			x	x	0	
	Проширено	F5	4	3			x	x	0	

BLE: *Branch if Result Less Than or Equal Zero- Signed numbers*

Грана се на адресу која се одређује додавањем бајта одступања на PC, само ако је резултат мањи или једнак нули. У супротном, наставља се са извршавањем следеће инструкције у секвенци. Резултат је бинарни број са знаком, а другим комплементом су представљени негативни бројеви.

Ова инструкција се разликује од BPL инструкције по томе што дозвољава премашење; дозволиће да дође до гранања чак и ако је резултат >+127.

MPU тестира Z, N и V флегове. Гранање се јавља уколико је Z=1 или N≠V.

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
BLE	Релативно	2F	4	2						

BLS: *Branch if Result is Lower or the Same As Zero - Unsigned numbers*

Грана се на адресу која се одређује додавањем бајта одступања на PC, само ако је резултат мањи или једнак нули. У супротном, наставља се са извршавањем следеће инструкције у секвенци.

Инструкција се користи за тестирање резултата одузимања бинарних бројева без знака. Разликује се од BLE инструкције, која ради са бинарним бројевима са знаком.

MPU тестира Z, и C флегове. Гранање се јавља уколико је Z=1 или C=1.

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
BLS	Релативно	23	4	2						

BLT: *Branch if Result Less Than Zero - Signed numbers*

Грана се на адресу која се одређује додавањем бајта одступања на PC, само ако је резултат мањи или једнак нули. У супротном, наставља се са извршавањем следеће инструкције у секвенци. Резултат је бинарни број са знаком, а другим комплементом су представљени негативни бројеви.

Ова инструкција се разликује од BMI инструкције по томе што дозвољава премашење; дозволиће да дође до гранања чак и ако је резултат <-127.

MPU тестира N и V флегове. Гранање се јавља уколико је N≠V.

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
BLT	Релативно	2D	4	2						

BMI: *Branch if Result is Minus*

Грана се на адресу 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 инструкција се разликује од BPL инструкције по томе што BMI само тестира N флег да би се видело да ли је резултат негативан (нпр. N=1). Он не тестира да ли је негативни резултат премашио минус 128 (нпр. B=1)

	Начин	Оп.	Број	Број

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
BMI	Релативно	2B	4	2						

BNE: *Branch if Result Not Equal to Zero (Z=0)*

Грана се на адресу која се одређује додавањем бајта одступања на PC, само ако резултат није једнак нули (Z=0). У супротном, наставља се са извршавањем следеће инструкције у секвенци.

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
BNE	Релативно	26	4	2						

BPL: *Branch if Result is Plus*

Грана се на адресу која се одређује додавањем бајта одступања на PC, само ако је резултат позитиван. У супротном, наставља се са извршавањем следеће инструкције у секвенци. Резултат је бинарни број са знаком.

Ова инструкција се разликује од BGT инструкције по томе што BPL само тестира N флег да би се видело да ли је резултат позитиван (нпр. N=0). Он не тестира да ли је позитиван резултат премашио +127 (нпр. V=1)

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
BPL	Релативно	2A	4	2						

BRA: *Branch Always*

Увек се грана на адресу која се одређује додавањем бајта одступања на PC.

PC + одступање → PC

	Начин	Оп.	Број	Број						

Мнемоник	адресирања	код	циклуса	бајтова	H	I	N	Z	V	C
BRA	Релативно	20	4	2						

BSR: *Branch to Subroutine*

Позива се потпрограм са адресе која се одређује додавањем бајта одступања на PC. Пре гранања, адреса повратка се смешта на стек.

PCL $\rightarrow$ (M_{SP})	; Стави на стек PCL
SP-1$\rightarrow$ SP	; Декрементирај SP
PCH $\rightarrow$(M_{SP})	; Стави на стек PCH
одступање + PC $\rightarrow$ PC	; Направи адресу за гранање

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
BSR	Релативно	8D	8	2						

BVC: *Branch If Overflow Flag is Cleared*

Грана се на адресу која се одређује додавањем бајта одступања на PC, само ако је флег премашења постављен на 0 (V=0). У супротном, наставља се са извршавањем следеће инструкције у секвенци.

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
BVC	Relativno	28	4	2						

BVS: *Branch If Overflow Flag is Set*

Грана се на адресу која се одређује додавањем бајта одступања на PC, само ако је флег премашења постављен на 1 (V=1). У супротном, наставља се са извршавањем следеће инструкције у секвенци.

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
BVS	Релативно	29	4	2						

CBA: *Compare Accumulator B to Accumulator A*

Одузима се садржај акумулатора B од акумулатора A, али се резултат не ставља ни у један акумулатор (садржаји акумулатора остају неизмењени). Резултат се користи само да би се поставили N, Z, V флегови.

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
CBA	Имплицитно	11	2	1			x	x	x	x

CLC: *Clear C fleg*

Брише се C флег у CCR регистру (поставља се на 0). Други флегови остају непромењени.

0→C

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
CLC	Имплицитно	0C	2	1						0

CLI: *Clear I fleg*

Брише се I флег у CCR регистру (поставља се на 0) - тј. забрањује се прихватање маскираног прекида. Други флегови остају непромењени.

0→I

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C

CLI	Имплицитно	0E	2	1	0						

CLR: *Clear Contents of Memory*

Брише се садржај селектованог бајта у меморији. Z флег се поставља на 1; N, C и V флегови се постављају на 0.

00 → (M)

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
CLR	Индексирано	6F	7	2			0	1	0	0
	Проширено	7F	6	3			0	1	0	0

CLRA/CLRB: *Clear the Content of Accumulator A or B*

Садржај акумулатора поставља се на нулу. Z флег се поставља на 1; N, C и V флегови се постављају на 0.

00→A ili 00→B

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
CLRA	Имплицитно	4F	2	1			0	1	0	0
CLRB	Имплицитно	5F	2	1			0	1	0	0

CLV: *Clear Overflow Flag*

Флег премашења у CCR се поставља на 0 (V=0). Нема утицаја на друге флегове.

Мнемони	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C

к										
CLV	Имплицитно	0A	2	1					0	

CMP: *Compare Accumulator with Memory*

Податак из меморије се одузима од селектованог акумулатора, али се резултат не смешта у акумулатор (тј. садржај акумулатора остаје непромењен). Резултат има само утицај на N, Z, V и C флегове.

A → (M) ili B → (M)

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
CMPA	Непосредно	81	2	2			x	x	x	x
	Директно	91	3	2			x	x	x	x
	Индексирано	A1	5	2			x	x	x	x
	Проширено	B1	4	3			x	x	x	x
CMPB	Непосредно	C1	2	2			x	x	x	x
	Директно	D1	3	2			x	x	x	x
	Индексирано	E1	5	2			x	x	x	x
	Проширено	F1	4	3			x	x	x	x

COM: *Complement Memory Data*

Комплементира се (инвертује) сваки бит селектованог бајта у меморији. Резултат ће утицати на N и Z флегове; C флег се поставља на један, а V флег се поставља на 0.

$$(\overline{M}) \rightarrow (M)$$

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C

COM	Индексирано	63	7	2		x	x	0	1
	Проширено	73	6	3		x	x	0	1

COMA/COMB: *Complement Contents of Accumulator A or B*

Комплементира се (инвертује) садржај селектованог акумулатора.. Резултат ће утицати на N и Z флегове; C флег се поставља на један, а V флег се поставља на 0.

$$(\overline{A}) \rightarrow (A) \text{ ili } (\overline{B}) \rightarrow (B)$$

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
COMA	Имплицитно	43	2	1			x	x	0	1
COMB	Имплицитно	53	2	1			x	x	0	1

CPX: *Compare X Register with Memory*

Одузима се двобајтни податак из меморије од садржаја X регистра, али резултат се не смешта у X регистар (тј. нема никаквог утицаја на садржај X регистра). Резултат има утицаја на флегове N, Z и V.

X регистар је 16-битни (два бајта). 16-битни операнд који се одузима од X се добија са два суседна бајта која су смештена у меморију почевши од адресе операнда. На пример, инструкција CPX $B000 ће довести да MPU уради следеће:

- Повезују се бајтови смештени на адресама B000 и B001 у 16-битну реч, где је бајт B000 бајт веће тежине.
- Одузима се ова 16-битна реч од X. 15-ти бит одређује знак.
- Мења се статус N, Z, и V флегова у зависности од резултата одузимања.
- Ова инструкција не утиче на C флег.

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
CPX	Непосредно	8C	3	3			x	x	x	

Директно	9C	4	2		x	x	x		
Индексирано	AC	6	2		x	x	x		
Проширено	BC	5	3		x	x	x		

DAA: *Decimal Adjust Accumulator A*

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 бинарни сабирак користи за сабирање BCD вредности.

DAA ради само са акумулатором A.

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
DAA	Имплицитно	19	2	1			x	x	x	x

DEC: *Decrement Memory Byte*

Декрементира се бајт смештен у меморији на адреси операнда и смешта резултат на исту адресу.

(M)-1 →(M)

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
DEC	Индексирано	6A	7	2			x	x	x	
	Проширено	7A	6	3			x	x	x	

DECA/DECB: *Decrement Contents o Accumulator A or B*

Декрементира се садржај селектованог акумулатора и резултат остаје у акумулатору. Уочимо да нема утицаја на С флег.

A-1 →A ili **B-1→B**

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
DECA	Имплицитно	4A	2	1			x	x	x	
DECB	Имплицитно	5A	2	1			x	x	x	

DES: *Decrement Stack Pointer*

Декрементира се садржај показивача стека. Нема утицаја на флегове.

SP-1 →SP

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
DES	Имплицитно	34	4	1						

DEX: *Decrement X Register*

Декрементира се садржај индекс регистра X. Ова инструкција утиче само на З флег.

X-1 →X

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
DEX	Имплицитно	09	4	1				x		

EOR: *Exclusive - OR Memory with Accumulator*

Извршава се операција ексклузивног ИЛИ податка из меморије, бит по бит, са садржајем селектованог акумулатора. Резултат остаје у акумулатору.

(M)⊕A → A ili (M)⊕B → B

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
EORA	Непосредно	88	2	2			x	x	0	
	Директно	98	3	2			x	x	0	
	Индексирано	A8	5	2			x	x	0	
	Проширено	B8	4	3			x	x	0	
EORB	Непосредно	C8	2	2			x	x	0	
	Директно	D8	3	2			x	x	0	
	Индексирано	E8	5	2			x	x	0	
	Проширено	F8	4	3			x	x	0	

INC: *Increment Memory Byte*

Инкрементира се садржај меморијске локације дате адресом операнда и резултат смешта на исту адресу.

(M) + 1→(M)

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
INC	Индексирано	6C	7	2			x	x	x	
	Проширено	7C	6	3			x	x	x	

INCA/INCB: *Increment Accumulator A or B*

Инкрементира се садржај селектованог акумулатора и резултат смешта у акумулатор. Нема никаквог утицаја на C флег.

A + 1→A ili **B + 1→B**

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
INCA	Имплицитно	4C	2	1			x	x	x	
INCB	Имплицитно	5C	2	1			x	x	x	

INC: *Increment Stack Pointer*

Инкрементира се садржај показивача стека. Нема утицаја на флегове.

SP+1 →SP

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
INS	Имплицитно	31	4	1						

INX: *Increment X Register*

Инкрементира се садржај индекс регистра X. Ова инструкција утиче само на 3 флег.

X+1 →X

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
INX	Имплицитно	08	4	1				x		

JMP: *Unconditional Jump*

MPU узима своју следећу инструкцију са адресе која је специфицирана адресом операнда. Ова инструкција има два могућа начина адресирања и оба су илустрована у даљем тексту:

Проширено

7Е	ЈМП $3Ц77
3Ц	
77	

Ова инструкција доводи до тога да се адреса операнда, 3С77, смести у програмски бројач. MPU затим узима следећу инструкцију са адресе 3С77.

Индексирано

6Е	ЈМП $24,Х
24	

Инструкција проузрокује да MPU дода одступање 24, које је 8-битни број без знака, на садржај Х регистра да би се добила адреса операнда. На пример, уколико је [Х]=03А0, онда ефективна адреса операнда постаје 03А0+24=03С4. Затим се РС пуни са 03С4, и MPU скаче на ту адресу по следећу инструкцију.

JCP: *Jump to Subroutine*

После узимања инструкције, MPU смешта текући садржај програмског бројача на стек. Ово је адреса инструкције која је иза JSR инструкције (тз. повратна адреса). MPU ставља у програмски бројач адресу потпрограма и наставља извршавање програма са те адресе.

JSR инструкција користи индексно или проширено адресирање, као и JMP инструкција. Погледати опис JMP инструкције и како се различите врсте адресирања користе за смештање адресе у РС (нпр. адресе потпрограма).

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
JSR	Индексирано	AD	8	2						
	Проширено	BD	9	3						

LDA: *Load Accumulator from Memory*

Садржај меморијске локације који је специфициран адресом операнда смешта се у специфицирани акумулатор. Садржај меморијске локације остаје неизмењен.

(M) → A ili **(M) → B**

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
LDAA	Непосредно	86	2	2			x	x	0	
	Директно	96	3	2			x	x	0	
	Индексирано	A6	5	2			x	x	0	
	Проширено	B6	4	3			x	x	0	
LDBA	Непосредно	C6	2	2			x	x	0	
	Директно	D6	3	2			x	x	0	
	Индексирано	E6	5	2			x	x	0	
	Проширено	F6	4	3			x	x	0	

LDS: *Load Stack Pointer*

Смешта се 16 битова (два бајта) податка из меморије у регистар показивача стека (SP). 16 битова података се узима са две узастопне меморијске локације почевши од адресе операнда.

На пример, инструкција LDS $45EA ће сместити бајт са адресе 45EA у SPX (виши бајт SP), и бајт са адресе 45EB у SPL (нижи бајт SP).

(M)→ SPH i **(M+1)→ SPL**

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
LDS	Непосредно	8E	3	3			(a)	x	0	
	Директно	9E	4	2			(a)	x	0	
	Индексирано	AE	6	2			(a)	x	0	
	Проширено	BE	5	3			(a)	x	0	

(а) N флег узима вредност 15-тог бита SP

LDX: *Load X Register*

Смешта се 16 битова (два бајта) податка из меморије у индекс регистар. 16 битова податка се узима са две узастопне меморијске локације почевши од адресе операнда.

На пример, инструкција LDS $45EA ће сместити бајт са адресе 45EA у XH (виши бајт X), а бајт са адресе 45EB у XL (нижи бајт X).

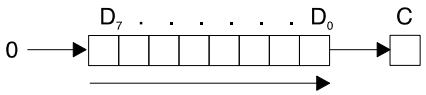
(M)→ XH i (M+1)→ XL

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
LDX	Непосредно	CE	3	3			(а)	x	0	
	Директно	DE	4	2			(а)	x	0	
	Индексирано	EE	6	2			(а)	x	0	
	Проширено	FE	5	3			(а)	x	0	

(а) N флег узима вредност 15-тог бита X

LSR: *Logical Shift Right*

Извршава се померање садржаја меморије за један бит удесно. Бит D_0 се помера у C флег, а нула се помера у бит D_7

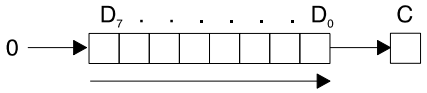


Мнемони	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C

к								
LSR	Индексирано	64	7	2		0	x	x
	Проширено	74	6	3		0	x	x

LSRA/LSRB: *Logical Shift Right of Accumulator or B*

Садржај специфицираног акумулатора се помера удесно један бит. Бит D_0 се помера у C флег, а 0 се помера у D_7



Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
LSRA	Имплицирано	44	2	1			0	x		x
LSRB	Имплицирано	54	2	1			0	x		x

NEG: *Negate Memory Data*

Од садржаја селектоване меморијске локације формира се други комплемент. Суштински конвертује позитивне у негативне бројеве и обрнуто.

$$(\overline{M})+1 \rightarrow (M)$$

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
NEG	Индексирано	60	7	2			x	x	(a)	(b)
	Проширено	70	6	3			x	x	(a)	(b)

NEGA/NEGB: *Negate Contents of Accumulator A ili B*

Од садржаја специфицираног акумулатора формира се други комплемент. Суштински конвертује позитивне у негативне бројеве и обрнуто.

$$\overline{\mathbf{A}} + \mathbf{1} \rightarrow \mathbf{A} \text{ ili } \overline{\mathbf{B}} + \mathbf{1} \rightarrow \mathbf{B}$$

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
NEGA	Имплицирано	40	2	1			x	x	(a)	(b)
NEGB	Имплицирано	50	2	1			x	x	(a)	(b)

NOP: *No Operation*

MPU не извршава никакве операције (активности) осим што инкрементира PC да би се припремио за узимање следеће инструкције.

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
NOP	Имплицирано	01	2	1						

ORA: *OR the Contents of Accumulator with Memory*

Подаци из меморије и садржај специфицираног акумулатора су у операцији ИЛИ бит по бит. Резултат остаје у акумулатору.

$$\mathbf{(M)+A \rightarrow A} \text{ ili } \mathbf{(M)+B \rightarrow B}$$

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
ORAA	Непосредно	8A	2	2			x	x	0	

	Директно	9A	3	2		x	x	0	
	Индексирано	AA	5	2		x	x	0	
	Проширено	BA	4	3		x	x	0	
ORAB	Непосредно	CA	2	2		x	x	0	
	Директно	DA	3	2		x	x	0	
	Индексирано	EA	5	2		x	x	0	
	Проширено	FA	4	3		x	x	0	

PSH: *Push accumulator Data onto Stack*

Садржај специфицираног акумулатора се ставља на стек на адресу специфицирану преко SP. Садржај SP се затим декрементира.

$$\mathbf{A \rightarrow (M_{SP})} \text{ ili } \mathbf{B \rightarrow (M_{SP})}$$
$$\mathbf{SP\text{-}1 \rightarrow SP}$$

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
PSHA	Имплицирано	36	4	1						
PSHB	Имплицирано	37	4	1						

PUL: *Pull Data from Stack*

SP се инкрементира. Садржај локације специфициране са SP се смешта у специфицирани акумулатор.

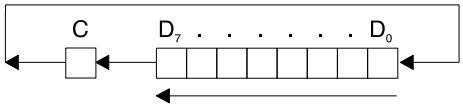
$$\mathbf{SP+1 \rightarrow SP}$$
$$\mathbf{(M_{SP}) \rightarrow A} \text{ ili } \mathbf{(M_{SP}) \rightarrow B}$$

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C

PULA	Имплицирано	32	4	1							
PULB	Имплицирано	33	4	1							

ROL: *Rotate Memory Data Left*

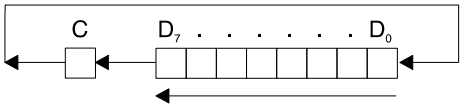
Садржај специфициране меморијске локације се помера улево за један бит. Претходна вредност C флега се помера у бит D_0. Бит D_7 се помера у C флег.



Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
ROL	Индексирано	69	7	2			x	x		x
	Проширено	79	6	3			x	x		x

ROLA/ROLB: *Rotate Accumulator Data Left*

Садржај специфицираног акумулатора се помера улево за један бит. Претходна вредност C флега се помера у бит D_0. Бит D_7 се помера у C флег.

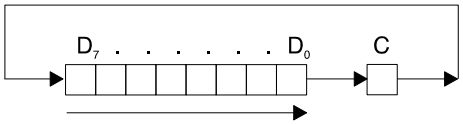


Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
ROLA	Имплицирано	49	2	1			x	x		x
ROLB	Имплициран	59	2	1			x	x		x

о

ROR: *Rotate Memory Data Right*

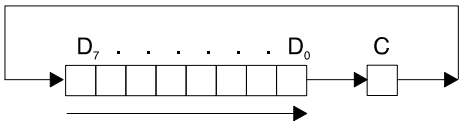
Садржај спец фициране меморијске локације се помера удесно за један бит. Претходна вредност C флега се помера у бит D_7. Бит D_0 се помера у C флег.



Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
ROR	Индексирано	66	7	2			x	x		x
	Проширено	76	6	3			x	x		x

RORA/RORB: *Rotate Accumulator Data Right*

Садржај специфицираног акумулатора се помера удесно за један бит. Претходна вредност C флега се помера у бит D_7. Бит D_0 се помера у C флег.



Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
RORA	Имплицирано	46	2	1			x	x		x
RORB	Имплицирано	56	2	1			x	x		x

RTI: *Return from Interrupt*

MPU скида седам узастопних бајтова са стека, инкрементирајући SP пре сваког узимања. Седам бајтова је смештено у CCR, ACCB, ACCA, XH, XL, PCX и PCL, респективно. Овим се враћа садржај ових MPU регистара на вредности које су имали када је MPU реаговао (одговарао) на прекид. Нова вредност PC ће вратити MPU на одговарају}у тачку у главном програму.

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
RTI	Имплициран о	3B	10	1	x	x	x	x	x	x

RTS: *Return from Subroutine*

Ово би требало да буде последња инструкција било ког потпрограма. MPU скида два бајта са стека и инкрементира SP пре сваког скидања. Два бајта се смештају у програмски бројач (PCX и PCL, респективно). Нова PC вредност ће вратити MPU на тачку у главном програму која је одмах иза JSR (скок у потпрограм).

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
RTS	Имплициран о	39	5	1						

SBA: *Subtract Accumulator B from Accumulator A*

Одузима се садржај ACCB од ACCA, а резултат ставља у ACCA. Нема утицаја на садржај ACCB.

$$\mathbf{A - B \rightarrow A}$$

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
SBA	Имплициран о	10	2	1			x	x	x	x

SBC: *Subtract memory Byte from Accumulator with Carry)*

Одузима се садржај меморијске локације и вредност C флега од специфицираног акумулатора, а резултат остаје у акумулатору.

A- (M) - C →A ili **B - (M) - C →A**

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
SBCA	Непосредно	82	2	2			x	x	x	x
	Директно	92	3	2			x	x	x	x
	Индексирано	A2	5	2			x	x	x	x
	Проширено	B2	4	3			x	x	x	x
SBCB	Непосредно	C2	2	2			x	x	x	x
	Директно	D2	3	2			x	x	x	x
	Индексирано	E2	5	2			x	x	x	x
	Проширено	F2	4	3			x	x	x	x

SEC: *Set C Flag*

Поставља се C флег на 1. Нема утицаја на друге флегове

1→ C

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
SEC	Имплицирано	0D	2	1						1

SEI: *Set I Flag*

Поставља I флег на 1 (дозвољава се прихватање маскираних прекида). Нема утицаја на друге флегове

1→ I

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
SEI	Имплицирано	0F	2	1		1				

SEV: *Set V Flag*

Поставља V флег на 1. Нема утицаја на друге флегове

1→ V

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
SEV	Имплицирано	0B	2	1					1	

STA: *Store Accumulator in Memory*

Смешта се садржај специфицираног акумулатора у одабрану меморијску локацију. Нема утицаја на садржај акумулатора

.

A →(M) ili **B →(M)**

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
STAA	Директно	97	4	2			x	x	0	
	Индексирано	A7	6	2			x	x	0	
	Проширено	B7	5	3			x	x	0	
STAB	Директно	D7	4	2			x	x	0	

	Индексирано	E7	6	2		x	x	0	
	Проширено	F7	5	3		x	x	0	

STS: *Store Stack Pointer*

Смешта се садржај SP у две узастопне меморијске локације почевши од адресе операнда. На пример, инструкција STS $453A ће сместити SPH у меморијску локацију 453A, а SPL у меморијску локацију 453Б.

SPH →(M) i SPL →(M)

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
STS	Директно	9F	5	2			(a)	x	0	
	Индексирано	AF	7	2			(a)	x	0	
	Проширено	BF	6	3			(a)	x	0	

(a) N флег узима MSB вредност SPH (нпр. бит 15 SP)

STX: *Store X Register*

Смешта се садржај X у две узастопне меморијске локације почевши од адресе операнда. На пример, инструкција STX $453A ће сместити XH у меморијску локацију 453A, а XL у меморијску локацију 453B.

XH →(M) i XL →(M)

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
STX	Директно	DF	5	2			(a)	x	0	

Индексирано	EF	7	2		(a)	x	0
Проширено	FF	6	3		(a)	x	0

(a) N флег узима MSB вредност XH (нпр. бит 15 XH)

SUB: *Subtract Memory Byte from Accumulator - No Carry*

Одузима се садржај одабране меморијске локације од специфицираног акумулатора, а резултат остаје акумулатору.

A- (M) →A ili **B - (M) →A**

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
SUBA	Непосредно	80	2	2			x	x	x	x
	Директно	90	3	2			x	x	x	x
	Индексирано	A0	5	2			x	x	x	x
	Проширено	B0	4	3			x	x	x	x
SUBB	Непосредно	C0	2	2			x	x	x	x
	Директно	D0	3	2			x	x	x	x
	Индексирано	E0	5	2			x	x	x	x
	Проширено	F0	4	3			x	x	x	x

SWI: *Software Interrupt*

MPU смешта PCL, PCX, XL, XH, ACCA, ACCB и CCR на стек и поставља флег за маскирање прекида I (*Interrupt Mask Flag)* на 1 тако да $\overline{IRQ}$ не може да прекида MPU. SWI вектор са меморијске локације FFFA и FFFB се учитава у PC. MPU наставља извршавање од ове адресе.

МПУ регистри	→	стек
1	→	И
(ФФФФА)	→	ПЦХ
(ФФФБ)	→	ПЦЛ

SWI инструкција проузрокује да MPU реагује на исти начин као што реагује на $\overline{IRQ}$ сигнал из I/O уређаја. Разлика је у томе што програмер управља где ће се у програму појавити SWI.

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
SWI	Имплицирано	3F	12	1						

TAB: *Transfer Accumulator A to Accumulator B*

Преноси се садржај ACCA у ACCB, остављајући ACCA непромењен.

A→B

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
TAB	Имплицирано	16	2	1			x	x	0	

TAP: *Transfer Accumulator A to CCR*

Преноси се садржај ACCA у регистар услова CCR, остављајући ACCA непромењен.

A→CCR

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
TAP	Имплициран	06	2	1	(a	(a	(a	(a	(a	(a

	о				)	)	)	)	)	)	

(а) Флегови у CCR-у добијају вредности према садржају ACCA

TBA: *Transfer Accumulator B to Accumulator A*

Преноси се садржај ACCB у ACCA, остављајући ACCB непромењен.

B→A

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
TBA	Имплицирано	17	2	1			x	x	0	

TPA: *Transfer CCR to Accumulator A*

Преноси се садржај регистра услова CCR у ACCA, остављајући регистар услова CCR непромењен.

CCR → A

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
TPA	Имплицирано	07	2	1						

TST: *Test Contents of Memory*

Постављају се N и Z флегови у CCR, према садржају специфициране меморијске локације и бришу се C и V флегови. MPU у ствари одузима 00_{16} од бајта у меморији и у зависности од тога поставља флегове.

(M)-00

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
TST	Индексно	6D	7	2			x	x	0	0
	Проширено	7D	6	3			x	x	0	0

TSTA/TSTB: *Test Contents of Accumulator*

Одузима се 00_{16} од специфицираног акумулатора. Овим ће бити постављени N и Z флегови према текућем садржају акумулатора. C и V флегови се бришу.

A-00 ili B-00

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
TSTA	Имплицирано	4D	2	1			x	x	0	0
TSTB	Имплицирано	5D	2	1			x	x	0	0

TSX: *Trnsfer SP to X*

Преноси се садржај SP у X регистар, остављајући SP не промењен, а затим се инкрементира X.

$$\mathbf{SP \rightarrow X}$$
$$X + 1 \rightarrow X$$

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
TSX	Имплицирано	30	4	1						

TSX: *Trnsfer X to SP*

Преноси се садржај X у SP регистар, остављајући X не промењен, а затим се декрементира SP.

X → SP
SP - 1 → SP

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
TXS	Имплицирано	35	4	1						

WAI: *Wait for Interrupt*

MPU ставља PCL, PCH, XL, XH, ACCA, ACCB и CCR на стек. MPU затим извршава инструкције и чека да се појаве сигнали на једном од улаза за прекид ($\overline{NMI}$ или $\overline{IRQ}$) или $\overline{RESET}$ улазу. Ове инструкције се обично користе као инструкције за заустављање на крају програма.

Мнемоник	Начин адресирања	Оп. код	Број циклуса	Број бајтова	H	I	N	Z	V	C
WAI	Имплицирано	3E	9	1						