Љубица Маркудова
Искра Јовановска
Јасна Домазетовска

Дигитална електроника и микропроцесори

за III година

2011

Рецензентска комисија

- Д-р Марија Кацарска, професор на Факултетот за електротехника и информатички технологии
- Дипл.инж. Софија Темкова, професор во СУГС ,,Михајло Пупин"-Скопје
- Дипл.инж. Невенка Смилевска, професор во СУГС ,,Михајло Пупин"-Скопје

*
* *

Лектор
Гордана Илиевска

*
* *

Компјутерска подготовка
Љубица Маркудова

*

Издавач: Министерство за образование и наука за Република Македонија

Печати: Графички центар дооел, Скопје

Тираж: 550

Со решение на Министерот за образование и наука на Република Македонија бр. 22-5315/1 од 30.11.2010 година се одобрува употребата на овој учебник.

CIP - Каталогизација во публикација
Национална и универзитетска библиотека "Св.Климент Охридски" , Скопје
621.38 . 049 .. 77 (075.3)
004 . 31 (075 . 3)

МАРКУДОВА, Љубица
Дигитална електроника и микропроцесори : [електротехничар за електроника и телекомуникации] / Љубица Маркудова, Искра Јовановска, Јасна Домазетовска. - Скопје : Министерство за образование и наука на Република Македонија, 2011. - 326, [1] стр. : илустр. ; 23 см

ISBN 978-608-226-175-1
1. Јовановска, Искра [автор] 2. Домазетовска, Јасна [автор]
COBISS.MK-ID 86462986

Предговор

Овој учебник е наменет за учениците од трета година во средните стручни училишта, електротехничка струка, за образовниот профил електротехничар за електроника и телекомуникации. Учебникот е работен според реформираната наставна програма и има за цел во целост да ги покрие наставните содржини по предметот Дигитална електроника и микропроцесори во трета година. Оваа наставна програма е реформирана во 2006 година, во согласност со реформите во средното стручно образование, со што се направи осовременување на наставата и наставните програми во стручното образование.

Наставниот предмет Дигитална електроника и микропроцесори во трета година е застапен со фонд од 5 часа неделно или 180 часа годишно. Посебен предизвик за авторскиот тим на овој учебник претставуваше атрактивноста и интересот што го предизвикува развојот на микропроцесорите и компјутерите. За да користењето на компјутерите биде поуспешно потребно е истите добро да ги познаваме. Учебникот е резултат на десетгодишна работа т.е. истражување, анализа и селекција на многу обемен наставен материјал од областа на дигиталната електроника и микропроцесорската техника. Во создавањето на овој учебник свој придонес имаат ѝ самите ученици преку давање на свои мислења и сугестии за наставата и презентацијата на наставниот материјал по овој предмет.

Учебникот опфаќа девет теми.

Првата тема, Основни комбинациски и секвенцијални компоненти претставува повторување на наставниот материјал по предметот Дигитална електроника и микропроцесори од втора година. Ваквото повторување е од посебна важност за понатамошното разбирање на наставниот материјал бидејќи микропроцесорите се дизајнирани токму од вакви компоненти.

Втората наставна тема, Основи на компјутерите е во корелација со предметот Информатика од прва година. Учениците треба да го утврдат своите знаења за карактеристиките и функцијата на основните хардверски и софтверски компоненти.

Со основните карактеристики, составните делови и начинот на работа на општиот модел на микропроцесор ќе се запознаеме во третата наставна тема, Општа архитектура на микропроцесор.

После запознавањето со генералната архитектура на микропроцесорот потребно е да се запознаеме со начинот за негово поврзување со различните видови мемории и периферни уреди. Ова е наставна цел на четвртата тема, Поврзување на микропроцесорот.

Петтата наставна тема се вика Програмирање на микропроцесор. Предметот Дигитална електроника и микропроцесори во трета година е единствен редовен предмет во кој се изучува примената на програмските

јазици. Како програмски јазик е избран асемблерот, кој спаѓа во групата на програмски јазици од понизок ред. Но, токму неговата блискост со машинскиот јазик ни овозможува подобро да ги проучиме хардверсите ресурси на микропроцесорите и другите компоненти во компјутерот.

Шестата тема, Микроконтролери се бави со проучување на хардверот и софтверот на еден реален микроконтролер, PIC16f84. Во последниве две-три години оваа тема предизвикува посебен интерес кај учениците бидејќи теоретските знаења од оваа област ги применуваат практично по предметот Практична настава во трета и четврта година.

Со шестата тема, 8-битни микропроцесори (8085 микропроцесор) започнува изучувањето на реалните микропроцесори на компанијата Интел. Со своите перформанси овој микропроцесор не може да се споредува со перформансите на Пентиум микропроцесорите. Но, микропроцесорот 8085 сè уште се произведува во огромен број на примероци бидејќи истиот наоѓа широка примена во едноставните електронски уреди и апарати. Овој микропроцесор е своевиден ,,евергрин" во областа на микропроцесорската техника.

Преку проучувањето на 16 и 32 битните микропроцесори во осмата тема полека, но сигурно се доближуваме до модерните Пентиум микропроцесори. Многу е нереално директно да се проучува архитектурата на Пентиум микропроцесорот, без да се познаваат составните делови и начинот на работа на претходните генерации микропроцесори. За оваа наставна тема од посебна важност е постапноста во излагањето на наставниот материјал.

Последната, девета тема ги обработува Пентиум микропроцесорите. За да учениците полесно го разберат овој комплексен наставен матаријал користен е блоковски пристап и селективност во анализата на пин дијаграмите.

Во учебникот после секоја тема има прашања, задачи и активности со што се настојува да се применат знаењата и направи проверка на постигнувањата на учениците.

Авторскиот тим на овој учебник се обиде наставниот материјал да го презентира на што е можно поедноставен и разбирлив начин, со прецизни објаснувања на постапките и процесите и изработка на јасни цртежи и слики.

Им се заблагодаруваме на нашите семејства за несебичната и сестрана подршка во изработката на овој учебник. Исто така, се заблагодаруваме на Рецензентската комисија која со своите корисни сугестии придонесе за подобра реализација на овој учебник.

Од авторскиот тим

Содржина

1. Основни комбинациски и секвенцијални компоненти

1.1. Бројни системи

За подобро разбирање на работата на микропроцесорите од голема важност е добро да ги познаваме бинарниот и хексадецималниот броен систем. Во секојдневниот живот човекот го користи декадниот броен систем. Во овој броен систем броевите се формираат од десет цифри, од 0 до 9. Почнувајќи од десно кон лево, првата цифра го дава бројот на единици, втората цифра го дава бројот на десетки, третата бројот на стотки, следните три цифри го даваат бројот на илјадарки итн. Можеме да заклучиме дека секоја цифра во бројот има своја тежина што зависи од нејзината положба во тој број. Така, цифрите што го даваат бројот на единици имаат тежина 10^0=1, оние што го даваат бројот на десетки имаат тежина 10^1=10, цифрите што го даваат бројот на стотки имаат тежина 10^2=100 итн.

1.1.1. Бинарен броен систем

Во бинарниот броен систем броевите се формираат од две цифри 0 и 1. Како и кај декадниот така и во бинарниот броен систем цифрите имаат различна тежина, но наместо десет, во основата на степенот е бројот 2. Така, почнувајќи од десно кон лево првата цифра има тежина 2^0=1, втората 2^1=2, третата 2^2=4, четвртата 2^3=8 итн

Доколку сакаме еден број запишан во бинарен броен систем да го претвориме во декаден запис, тогаш треба да ги собереме сите цифри од бинарниот број, но претходно помножени со нивните тежини, како што е прикажано во примерот 1.1.

Пример 1.1: Бројот 101011_2 сакаме да го претставиме во декаден броен систем

2^5 2^4 2^3 2^2 2^1 2^0 - тежини
1 0 1 0 1 $1_2 = 1\cdot2^0 + 1\cdot2^1 + 0\cdot2^2 + 1\cdot2^3 + 0\cdot2^4 + 1\cdot2^5 = 1+2+4+8 = 43_{10}$

Цифрите 0 и 1 во бинарнот броен систем се викаат битови. Битот што се множи со тежината $2^0=1$ се вика бит со најмало значење, а битот што се множи со најголемиот степен (во примеров 2^5) бит со најголемо значење.

Доколку сакаме некој број од декаден броен систем да го претвориме во бинарен, тогаш постапуваме на следниов начин. Бројот го делиме со 2 и настрана го запишуваме добиениот остаток . Добиениот целобројен резултат од делењето повторно го делиме со 2 и настрана го запишуваме остатокот. Ова го повторуваме сè додека не добиеме резултат нула. Потоа ги читаме добиените остатоци, и тоа почнувајќи од последно запишаниот кон првиот запишан остаток или од доле нагоре, како што е прикажано во примерот 1.2.

Пример 1.2: Бројот 38_{10} сакаме да го претставиме со низа од нули и единици.

	остаток
38 : 2 = 19	0
19 : 2 = 9	1
9 : 2 = 4	1
4 : 2 = 2	0
2 : 2 = 1	0
1 : 2 = 0	1

Добиениот резултат изнесува $38_{10} = 100110_2$.

Можеби побрз начин на претворање броеви од декаден во бинарен броен систем е декадниот број да го претставиме како сума од некои од броевите 1, 2, 4, 8, 16, 32, 64, 128 итн. Ова, всушност, се тежините на цифрите во бинарниот броен систем. Доколку некоја од овие тежини влегла во збирот, тогаш на местото од таа цифра ќе ставиме 1, а ако не влегла тежината во збирот, тогаш на тоа место ќе ставиме нула. Ова е прикажано во пример број 1.3.

Пример 1.3: Бројот 38 може да се добие како сума од броевите 32, 4 и 2. Ова значи дека единици ќе бидат втората, третата и шестата цифра од бинарниот број, почнувајќи од десно кон лево, бидејќи втората цифра има тежина $2^1=2$, третата $2^2=4$ и шестата $2^5=32$.

Можеби некој ќе се запраша зошто бинарниот броен систем е толку важен. Тој е од огромна важност, бидејќи соодветствува со машинскиот јазик. Машинскиот јазик е јазик на компјутерите и на сите дигитални уреди. Во машинскиот јазик единицата значи течење струја или постоење напон, а нулата значи нетечење струја или непостоење напон.

1.1.2. Хексадецимален броен систем

Броевите запишани во бинарен броен систем се многу долги. Со цел да се скрати запишувањето, се користи хексадецималниот броен систем. Во овој броен систем броевите се формираат од 16 цифри. Првите 10 цифри се исти како и на декадниот броен систем од 0 до 9, а потоа следуваат цифрите претставени со букви, и тоа $A_{16}=10_{10}$, $B_{16}=11_{10}$, $C_{16}=12_{10}$, $D_{16}=13_{10}$, $E_{16}=14_{10}$, $F_{16}=15_{10}$. Наместо индексот 16 хексадецималните броеви се бележат со додавање на буквата H на крајот од бројот. Претворањето од хексадецимален во декаден броен систем се врши на ист начин како и претворањето од бинарен во декаден, со таа разлика што наместо со основа два, работиме со основа 16, а вредноста на степенот зависи од положбата на цифрата во бројот. Ова е прикажано во пример 1.4.

Пример 1.4: 2AH= $2\cdot16^1+A\cdot16^0=2\cdot16+A\cdot1$=32+10 = 42_{10}

Доколку сакаме декадниот број да го претвориме во хексадецимален, тогаш бројот го делиме со 16 и остатокот го запишуваме настрана.

За нас од поголем интерес е претворањето на броевите од хексадецимален во бинарен броен систем и обратно. За да го претвориме хексадецимелниот број во бинарен не е потребна никаква пресметка, туку едноставна замена на хексадецимелните цифри со бинарни. Една цифра од хексадецималниот броен систем се претставува со четири цифри од бинарниот броен систем . Која било цифра од хексадецималниот броен систем може да се претстави како збир од броевите 1, 2, 4 и 8. На позициите од цифрите што влегуваат во збирот се пишува 1, а на позициите од цифрите што не влегуваат во збирот се пишува 0. Ова е илустрирано во примерите 1.5, 1.6 и 1.7.

Пример 1.5: Бројот $C_{(16)}=12_{(10)}$ можеме да го претставиме како збир од броевите 8 и 4 и поради тоа во бинарен броен овој број би бил 1100.
На местото од 2 и 1 напишавме нула, бидејќи тие не влегоа во збирот.

Пример 1.6: Бројот $7_{(16)}=7_{(10)}$ претставува збир од броевите 4, 2 и 1, па во бинарен броен систем овој број би бил еднаков на 0111.

Пример 1.7:

E4H = $\underline{1110}\ \underline{0100}_2$

$E_{16}= 14_{10}=8_{10}+4_{10}+2_{10}$ $4_{16}=4_{10}$

Доколку сакаме еден бинарен број да го претвориме во хексадецимелен, тогаш бинарниот број го делиме во групи по 4 цифри, почнувајќи од десно кон лево и на секоја таква група одговара по една хексадецимална цифра. Доколку

последната група одлево нема 4 цифри, тогаш додаваме нули одлево. Ова е прикажано во пример 1.8.

Пример 1.8: Бинарниот број 000110101001 треба да се претвори во хексадецимален броен систем

0001 1010 1001 = 1A9H

$1_{16}=1_{10}$, $A_{16}=10_{10}= 8_{10}+2_{10}$, $9_{16}=9_{10}=8_{10}+1_{10}$

Собирањето во хексадецимален броен систем се врши така што се собираат цифри на исти позиции слично како и кај декадниот броен систем. Ако збирот на двете цифри е поголем од 16, тогаш збирот се дели со 16 и како резултат на таа позиција се запишува остатокот, а на следната позиција го пренесуваме добиениот количник. Собирањето е прикажано во пример 1.9.

Пример 1.9:

```
    1          - пренос
  3 A 2 B H    - прв број
+ A 9 C 1 H    - втор број
  E 3 E C H    - збир
```

При собирањето на овие два броја ќе се појави пренос на третата позиција од десно.

$A_{16}+9_{16}=10_{10}+9_{10}=19_{10}$, $19_{10}:16_{10}=1_{10}$ остаток 3_{10}

Остатокот 3 го запишуваме како резултат на третата позиција од десно, а количникот 1 го пренесуваме на четвртата позиција. За четвртата позиција ќе добиеме резултат $A_{16}+3_{16}+1_{16}= 10_{10}+3_{10}+1_{10}=14_{10}=E_{16}$

Кај операцијатата собирање имаме пренос од пониска кон повисока позиција, а кај операцијата одземање имаме позајмување од повисока кон пониска позиција. Во декаден броен систем се позајмува 10, а во хексадецимален броен систем 16. Ова е прикажано во примерот 1.10.

Пример 1.10:

```
   16   16
  7 5  D  3
- 1 C  A  8
  5 9  2  B
```

Бројот 8 е поголем од бројот 3. Позајмуваме од повисоката позиција. На првата позиција ги собираме 16 и 3 и од сумата 19 го одземаме бројот 8. На првата позиција добиваме резултат $19-8=11_{(10)}=B_{(16)}$. Бидејќи за првата позиција позајмивме од втората, бројот D_{16} ќе го намалиме за еден и тој ќе стане C_{16}. $C_{16}-A_{16} = 2_{16}$.

Бинарниот броен систем е најзастапен при работа со сметачи и со други дигитални системи, декадниот е најразбирлив за човекот, а хексадецималниот е често употребуван поради кратките записи што тој ги нуди.

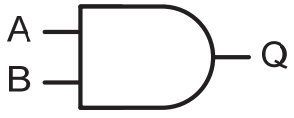
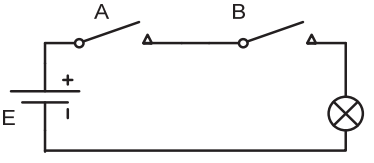
1.2. Основни логички кола

Основни логички кола што се користат за обработка на дигиталните сигнали се инвертор(НЕ), ИЛИ, И, НИ, НИЛИ, ЕКСИЛИ и ЕКСНИЛИ колата. Потребно е да се познаваат симболите за овие логички кола и нивните таблици на вистинитост. За подобро разбирање на нивната работа секоја логичка функција е презентирана преку електрично коло составено од два прекинувачи, извор на еднонасочен напон и сијалица како потрошувач.

И коло

За да на излез од И колото добиеме единица сите влезови треба да бидат на високо ниво односно ниво на логичка единица. За да на излез од И колото добиеме логичка нула доволно е барем еден од влезовите да биде на ниво на логичка нула.

За да светне сијалицата потребно е двата прекинувачи да бидат затворени.

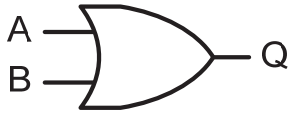
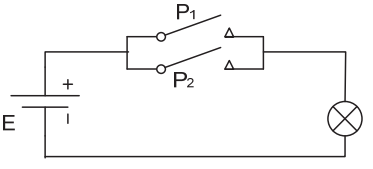
Логички симбол	Табела на вистинитост			Електрично коло
 Q=A∧B	A	B	Q	
	0	0	0	
	0	1	0	
	1	0	0	
	1	1	1	

Слика 1.1 Претставување на И коло

ИЛИ коло

За да на излез од ИЛИ колото добиеме нула сите влезови треба да бидат на ниско ниво односно ниво на логичка нула. За да на излез од ИЛИ колото добиеме логичка единица доволно е барем еден од влезовите да биде на ниво на логичка единица.

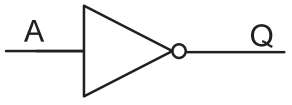
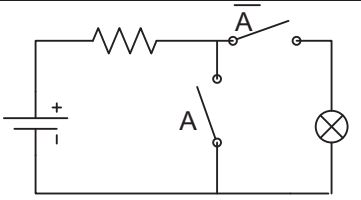
За да светне сијаличката доволно е барем еден прекинувач да биде затворен.

Логички симбол	Табела на вистинитост			Електрично коло
 Q=A∨B	A	B	Q	
	0	0	0	
	0	1	1	
	1	0	1	
	1	1	1	

Слика 1.2 Претставување на ИЛИ коло

Инвертор (НЕ)

Инверторот служи за пресметка на прв комплемент на влезниот бит. Ако на влез од инверторот имаме логичка единица тогаш на излез ќе добиеме логичка нула и обратно.

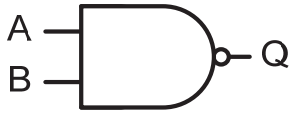
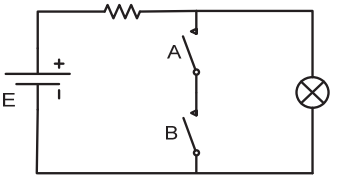
Логички симбол	Табела на вистинитост		Електрично коло
 $Q=\overline{A}$	A	Q	
	0	1	
	1	0	

Слика 1.3. Претставување на инвертор

НИ коло

НИ колото претставува комбинација од две логички кола, И коло и инвертор. На својот излез дава нула само кога двата влеза се единици односно се на високо ниво. За да на излез добиеме еден доволно е само еден влез да биде на ниско ниво односно логичка нула.

Сијалицата нема да свети само кога и двата прекинувачи се затворени. Двата затворени прекинувачи формираат куса врска за сијалицата.

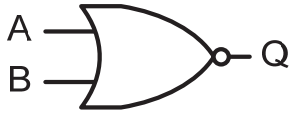
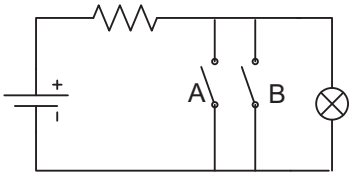
Електричен симбол	Табела на вистинитост			Електрично коло
 $Q=\overline{A \wedge B}$	A	B	Q	
	0	0	1	
	0	1	1	
	1	0	1	
	1	1	0	

Слика 1.4. Претставување на НИ коло

НИЛИ коло

НИЛИ колото претставува комбинација од две логички кола, ИЛИ коло и инвертор. На својот излез дава единица само кога двата влеза се нули односно се на ниско ниво. За да на излез добиеме нула доволно е само еден влез да биде на високо ниво односно логичка единица.

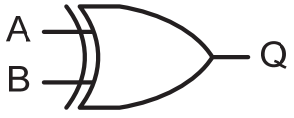
Сијалицата нема да свети кога барем еден од двата прекинувачи е затворен бидејќи тогаш се формираа куса врска за сијалицата.

Логички симбол	Табела на вистинитост			Електрично коло
 $Q=\overline{A \vee B}$	A	B	Q	
	0	0	1	
	0	1	0	
	1	0	0	
	1	1	0	

Слика1.5. Претставување на НИЛИ коло

ЕКСИЛИ и ЕКСНИЛИ коло

ЕКСИЛИ-колото врши споредба на влезните битови. На неговиот влез добиваме нула ако влезните битови се еднакви, а еден ако тие се различни.

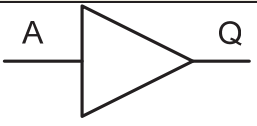
Логички симбол	Табела на вистинитост		
 Q=A⊕B	A	B	Q
	0	0	0
	0	1	1
	1	0	1
	1	1	0

Слика 1.6. Претставување на ЕКСИЛИ коло

ЕКСНИЛИ претставува комбинација од две логички кола, ЕКСИЛИ коло и инвертор. На неговиот влез добиваме единица ако влезните битови се еднакви, а нула ако тие се различни.

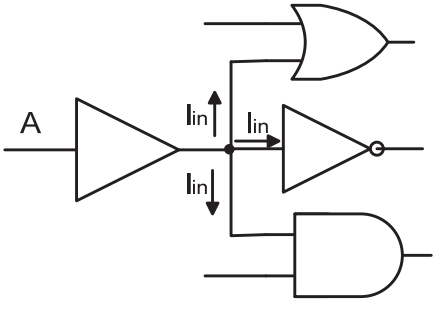
Бафер

Излезот на баферот е еднаков на неговиот влез.

Логички симбол	Табела на вистинитост	
 Q=A	A	Q
	0	0
	1	1

Слика 1.7. Претставување на бафер

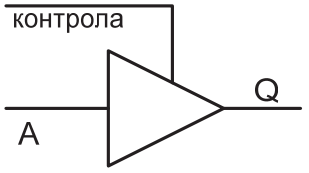
Баферот не врши обработка на сигналите , но за разлика од останатите логички кола тој може да поднесе поголеми струјни оптеретувања на неговиот излез. Поради тоа тој се користи кога на излезот од едно логичко коло треба да се поврзат повеќе други логички кола. Ова е прикажано на слика 1.8.



Слика 1.8. Поврзување на излез А со повеќе логички кола преку бафер

Покрај стандардниот бафер во дигиталните кола се користи таканаречениот **бафер со три состојби**. Овој вид на бафер има три пина: влез, излез и контролен пин. Ако контролниот бит е нула, тогаш излезот е во

состојба на висока импенданса. Ако контролниот бит е еден, тогаш излезот ќе биде еднаков на влезот. Преку контролниот бит вршиме вклучување и исклучување на излезниот пин. Баферот многу често се користи кога повеќе уреди треба да се поврзат на иста линија за пренос.

Логички симбол	Табела на вистинитост		
	контрола	A	Q
	1	0	0
	1	1	1
	0	0	Состојба на висока импенданса
	0	1	Состојба на висока импенданса

Слика 1.9. Претставување на бафер со три сосотојби

1.3. Интегрирани логички кола

Со групирање на основните логички кола се формираат посложени кола како што се комбинациските и секвенцијалните компоненти. Но, за да логичките кола се поврзат во функционална целина истите мора да исполнат одредени услови. Тие треба да користат ист напон за напојување и логичките нивоа за влез и излез да бидат усогласени. Интегрираните кола кои го исполнуваат овој услов формираат таканаречени фамилии од интегрирани кола. Најпознати фамилии интегрирани кола се TTL (Transistor-Transistor –Logic) и CMOS (Complementary Metal-Oxide-Silicon) фамилијата. Во рамките на овие фамилии се развиле повеќе серии интегрирани кола. На пример во рамките на TTL фамилијата постојат сериите: TTL(74xx), TTL(74LSxx), TTL(74Sxx), TTL(74ALSxx), TTL(74ALSxx), додека во рамките на CMOS фамилијата сериите: CMOS(40xx), CMOS(74HCxx), CMOS(CD4xx).

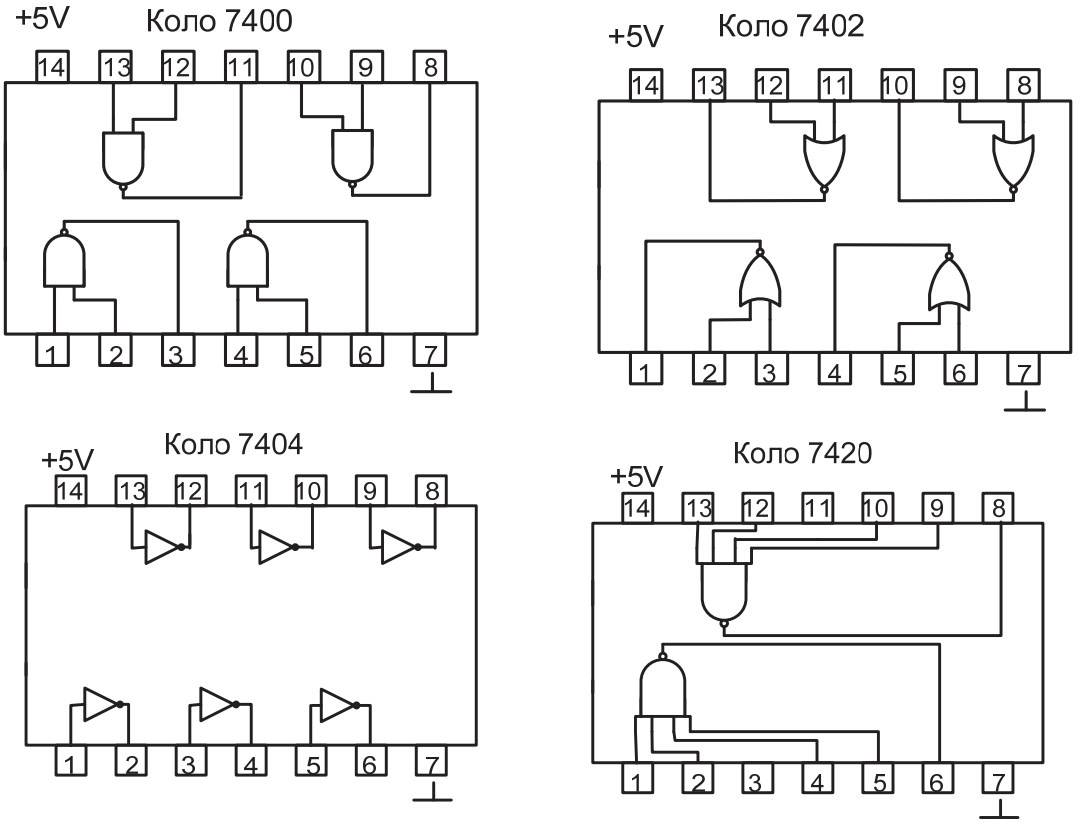
Сите интегрирани кола од TTL фамилијата користат напон за напојување од +5V±0,25V. Во табелата 1.1. се дадени влезно – излезните нивоа на напонот за состојба на логичка нула и состојба на логичка единица.

Логичко ниво	Влезен напон	Излезен напон
0	0,8V max	0,45V max
1	2V min	2,4 V min

Табела 1.1. Вредности на напон за TTL фамилијата интегрирани кола

Ако на влезниот пин пристигне сигнал, при што напонот има вредност меѓу 0.8 до 2.0 V, тогаш состојбата ќе биде недефинирана.

На слика 1.10. дадени се шемите на неколку интегрирани кола од TTL фамилијата. Сите овие кола припаѓаат на стандардната TTL серија 74xx и се во пакување со двојна линија. Колото 7400 содржи четири НИ порти , секоја со по два влеза. Колото 7402 содржи четири НИЛИ порти , секоја по два влеза. 7404 содржи шест идентични инвертори и колото 7420 содржи две НИ кола , секое со по четири влеза. Неискористените влезови на И и НИ портите треба да се поврзат со напонот на напојување. Доколку овој напон е поголем од 5,5V тогаш треба де се употребат pull up отпорници. Неискористените влезови на ИЛИ и НИЛИ портите треба да се поврзат со маса.



Слика 1.10. Структура на интегрирани кола од TTL фамилијата

Во табелата 1.2. дадени се максималните вредности на влезните и излезните струи за ниво на логичка нула и логичка единица.

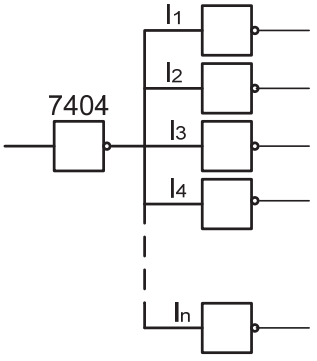
Логичко ниво	Влезна струја	Излезна струја
0	-1,6mA max	16mA max
1	0,04mA max	0,4mA max

Табела 1.2. Вредности на струја за TTL фамилијата интегрирани кола

Знакот минус на влезната струја за ниво на логичка единица значи дека насоката на струјата не е кон пинот туку од пинот. Максималните вредности на струите се користат за пресметка на оптоварувањето на излезните пинови на

логичките кола. Имено на излезниот пин од едно логичко коло можат да се приклучат други комбинациски кола. За да се постигне добра компатибилност, збирот од влезните струи на приклучените логички кола не смее да биде поголем од струјата на излезниот пин на првото логичко коло.

Поврзувањето на едно интегрирано коло со повеќе други интегрирани кола е прикажано на слика 1.11. Се работи за интегрирано коло 7404 кое содржи шест идентични инвертори во пакување со двојна линија (слика 1.10.). Кога на излез од колото имаме логичка нула тогаш излезната струја ќе изнесува $I_{из} \leq 16mA$. Влезните струи на приклучените интегрирани кола изнесуваат - $I_{вл} \leq 1,6mA$. Споредувајќи ги вредностите на максималните вредности на струите можеме да заклучиме дека на излез од едно интегрирано коло 7404 можеме да приклучиме десет други такви кола без да направиме преоптеретување.



Слика 1.11. Поврзување на едно TTL коло со девет други

Кога на излез од колото 7404 имаме логичка единица тогаш максималната вредност на струјата изнесува $I_{из} \leq 0,4mA$, а на влез од приклучените кола $I_{вл} \leq 0,04mA$. Заклучокот ќе биде ист како и во предходниот случај односно максималниот број на приклучени интегрирани кола изнесува 10.

Ако излезните пинови на интегрираното коло имаат поголемо оптоварување од пропишаното, тогаш е потребно да се изврши поврзувањето преку бафер. Баферот дозволува трипати поголемо оптоварување во однос на другите интегрирани кола.

Предности на интегрираните кола од CMOS фамилијата е малата потрошувачка и поголемата брзина на работа. Овие интегрирани кола користат напон на напојување од +5V до +15V.

Логичко ниво	Влезен напон	Излезен напон
0	1,5V	0,5V
1	3,5V	4,5V

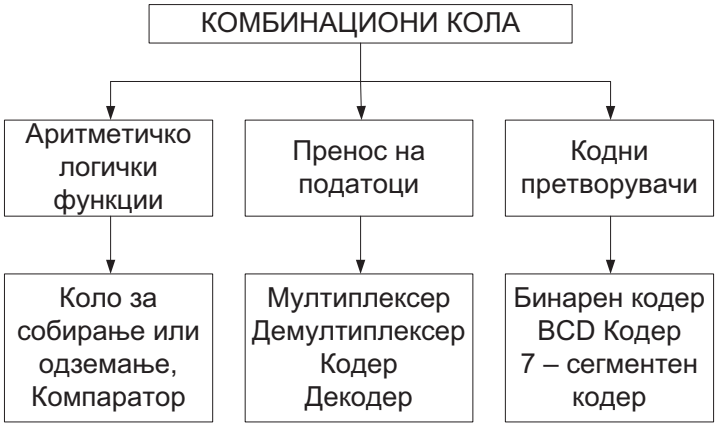
Табела 1.3. Вредности на напон за CMOS фамилијата интегрирани кола

Во табелата 1.3. се прикажани влезно – излезните нивоа на напонот за состојба на логичка нула и состојба на логичка единица за интегрираните кола од CMOS фамилијата.

Влезните и излезните струи на колата од CMOS фамилијата за ниво на логичка нула и логичка единица се многу помали од оние на TTL фамилијата и тие се од редна величина на μA. Ова е поради големата влезна импенданса на фетовите од кои се направени CMOS интегрираните кола.

1.4. Комбинациски кола

Од основните логички порти можат да се изведат посложени, таканаречени комбинациски кола. Кај комбинациските кола состојбата на излезите зависи само од моменталната состојба на влезовите. За разлика од секвенцијалните кола претходните состојби на влезовите и излезите немаат никакво влијание врз моменталната состојба на излезите.



Слика 1.12. Поделба на комбинациските кола

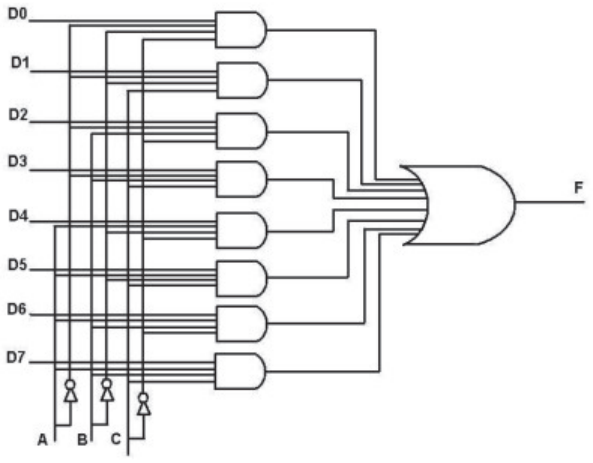
На слика 1.12. прикажана е поделба на комбинациските кола во зависност од нивната намена. Секое од овие кола ќе биде објаснето во понатамошниот текст.

1.4.1. Мултиплексер-демултиплексер

Мултиплексерот е коло со 2^n податочни влеза, еден податочен излез и n контролни (адресни) влеза со кои се селектира еден од влезовите. Селектираниот влез се поврзува со единствениот излез.

На слика 1.13. е прикажана шема на мултиплексер со 8 податочни влеза, 3 контролни (адресни) влеза и еден податочен излез. Табелата 1.4. претставува негова табела на вистинитост.

Секое од осумте И кола всушност претставува порта за пропуштање на корисничкиот податок D. За да И портата биде отворена битовите кои пристигнуваат на нејзиниот втор, трет и четврти влез треба да бидат единици. Ако било кој од овие влезови биде нула И колото ќе го помножи податокот D со нула и на излез од И колото ќе добиеме нула.



Слика 1.13. Структура на мултиплексер

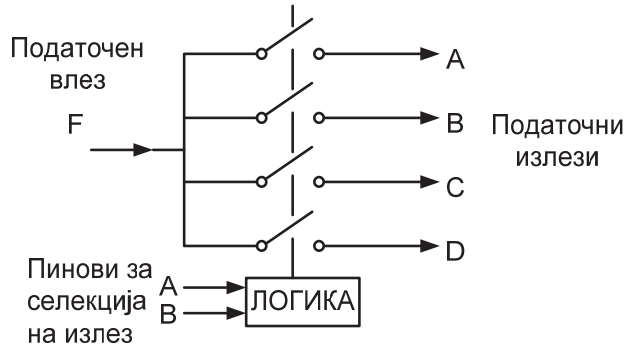
Адресирање			Избор
A	B	C	на влез
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

Табела 1.4. Табела на вистинитост за мултиплексер

Секоја И порта си има своја единствена адресна комбинација ABC. Не може да се случи истовремено да бидат отворени две или повеќе И порти. ИЛИ колото на излез ги собира сите излези од осумте И порти и збирот го пренесува на единствениот податочен излез. ИЛИ колото на својот влез ќе има седум нули и еден податочен бит D. Кој податочен бит ќе дојде до портата ИЛИ и ќе продолжи до единствениот излез зависи од адресната комбинација составена од битовите A, B и C.

Демултиплексерот е коло со спротивна функција од мултиплексерот, односно тој е коло со еден влез и повеќе излези, а кој излез ќе биде поврзан со единствениот влез зависи од комбинацијата адресни битови. Ова е прикажано на слика 1.14 .

Демултиплексерот земен како пример има еден податочен влез, два контролни (адресни) влеза и четири податочни излези. Влезниот адресен код AB одлучува кој од прекинувачите ќе биде затворен, односно решава кој податочен излез ќе има иста вредност како и единствениот податочен влез. Во табелата на вистинитост 1.5. прикажани се адресните кодови за сите четири податочни излези.



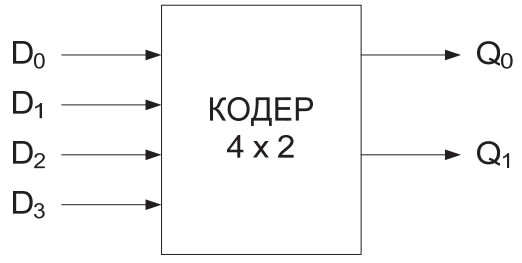
Слика 1.14. Функционална шема на демултиплексер

Адресирање		Избор на излез
A	B	
0	0	A
0	1	B
1	0	C
1	1	D

Табела 1.5. Табела на вистинитост за демултиплексер

1.4.2. Кодер и декодер

За разлика од мултиплесерот кој одбира еден податочен влез од повеќето и неговиот податок го испраќа до единствениот излез , **кодерот** ги зема во предвид сите податочни влеза истовремено и ги претвара во единствен бинарен излезен код. На слика 1.15. прикажана е блок шемата на декодер со четири влеза и два излези и неговата табела на вистинитост 1.6.



Слика 1.15. Блок шема на кодер

влезови				излези	
D_3	D_2	D_1	D_0	Q_1	Q_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

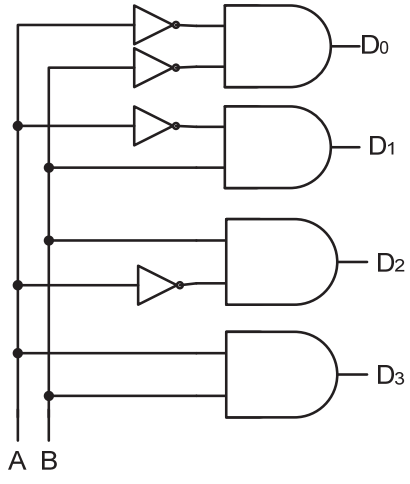
Табела 1.6.Табела на вистинитост за кодер

Од табелата на вистинитост можеме да заклучиме дека за секоја бинарна излезна комбинација треба да биде активен само еден влез. Проблем е што кодерот генерира погрешен излезен код доколку два или повеќе влеза се на високо ниво. На пример доколку влезовите D_1 и D_2 се на високо ниво тогаш кодерот нема да генерира ни 01 , ни 10 туку погрешен излезен код 11. Решение е влезовите да бидат со различен приоритет и ако има повеќе единици на влез тогаш излезите ќе одговараат на активниот влез со најголем приоритет. Табелата 1.7. претставува табела на вистинитост за истиот кодер прикажан на слика 1.15, но со најголем приоритет на влезот D_3 и најмал приоритет на влезот D_0. Од табелата на вистинитост 1.7. гледаме дека доколку е активен влезот со најголем приоритет тогаш кодерот нема да ги земе во предвид влезовите со помал приоритет односно тие се незначајни и се обележани со X.

влезови				излези	
D_3	D_2	D_1	D_0	Q_1	Q_0
0	0	0	1	0	0
0	0	1	x	0	1
0	1	x	x	1	0
1	x	x	x	1	1

Табела1.7.Табела на вистинитост за кодер со приоритет на најзначајниот влез

Декодерот има спротивна функција од кодерот односно бира еден излез во зависност од бинарниот влезен код. Бројот на излези зависи од бројот на влезови. Ако на влез од декодерот доаѓа n-битен број тогаш со него се селектира само еден излез од вкупно 2^n излеза. На сликата 1.16. е прикажан декодер со два влеза и четири излеза и неговата табела на вистинитост 1.8. Овој декодер може да се користи за селектирање на четири мемориски чипови



Слика 1.16. Логички дијаграм за декодер

влезови		излези			
A	B	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Табела 1.8. Табела на вистинитост за декодер со два влеза и четири излези

Ако на влез од декодерот дојде комбинација 00, тоа значи дека е селектиран првиот мемориски чип, а ако влезната комбинација е 01, тогаш ќе биде селектиран вториот мемориски чип итн.

1.4.3. Коло за аритметичко собирање

Не може да се замисли компјутер кој ќе нема коло за аритметичко собирање. Кај логичкото собирање, кое го врши ИЛИ колото, 1+1 е еднакво на 1, а кај аритметичкото собирање 1+1 е еднакво на нула и имаме пренос 1 од помалку кон повеќе значаен бит, односно од понизок кон повисок бит. Ова ќе го илустрираме преку пример 1.11.

Пример 1.11: Имаме два четирибитни броја A=0101, B=1100. Прво ќе ги собереме логички, а потоа и аритметички.

Кај логичкото собирање прикажано на сликата 1.17. битовите од соседните позиции не си влијаат еден на друг. Посебно се собираат битовите од секоја позиција при што се користи табелата на вистинитост за ИЛИ коло прикажана на сликата 1.4 .

A	0101	Логичко собирање
B	1100	
Збир	1101	

Слика 1.17. Пример за логичко собирање

Кај аритметичкото собирање имаме пренос. На слика 1.18. прикажана е постапката за аритметичко собирање и табелата на вистинитост 1.9. според која се врши овој вид на собирање.

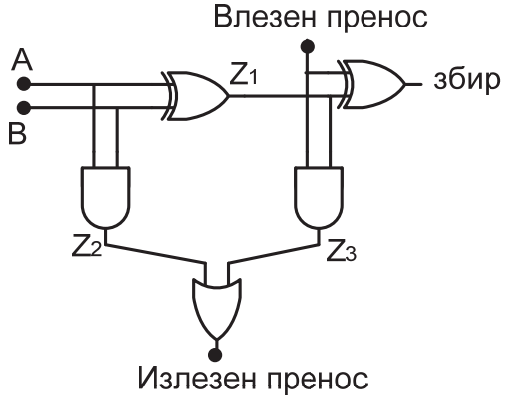
Пренос	11	
A	0101	Аритметичко собирање
B	1100	
Збир	10001	

Слика 1.18 Пример за аритметичко собирање

Бит 1	Бит2	збир	пренос
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Табела 1.9 Табела на вистинитост за аритметичко собирање

При собирањето на битовите од третата позиција, 1+1 ќе добиеме резултат 0, но и пренос на бит 1 од третата на четвртата позиција. При собирањето на битовите од четвртата позиција ќе имаме 0(од A)+1(од B)+1(пренос)=0 ќе добиеме збир нула и пренос 1 од четвртата на повисоката петта позиција.



Слика 1.19. Коло за аритметичко собирање со пренос

На сликата 1.19. е прикажано коло за аритметичко собирање. Гледаме дека истото е составено од основните логички кола: ЕКСИЛИ, ИЛИ и И коло.

Табелата 1.10. е деталната табела на вистинитост на колото за аритметичко собирање прикажано на слика 1.19. Заради поголема прегледност дадени се равенките за пресметка на внатрешните битови односно меѓурезултатите.

влезови			внатрешни битови			излези	
A	B	$П_{вл}$	$Z_1=A \oplus B$	$Z_2=A \wedge B$	$Z_3=Z_1 \wedge П_{вл}$	Збир=$П_{вл} \oplus Z_1$	$П_{из}=Z_1 \vee Z_2$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	1	0	0	1	0
0	1	1	1	0	0	0	1
1	0	0	1	0	0	1	0
1	0	1	1	0	0	0	1
1	1	0	0	1	0	0	1
1	1	1	0	0	1	1	1

Табела 1.10. Табела на вистинитост за коло за аритметичко собирање на два битови со пренос

Колото што е прикажано на сликата 1.19. собира само еднобитни броеви. Доколку сакаме да собираме четирибитни броеви, каков што е примерот, тогаш редно треба да се поврзат четири вакви еднобитни собирачи, со тоа што излезниот бит за пренос (carry out) се носи на следниот собирач одлево, а влезниот бит за пренос (carry in) доаѓа од претходниот собирач оддесно.

1.4.4. Еднобитна аритметичко-логичка единица

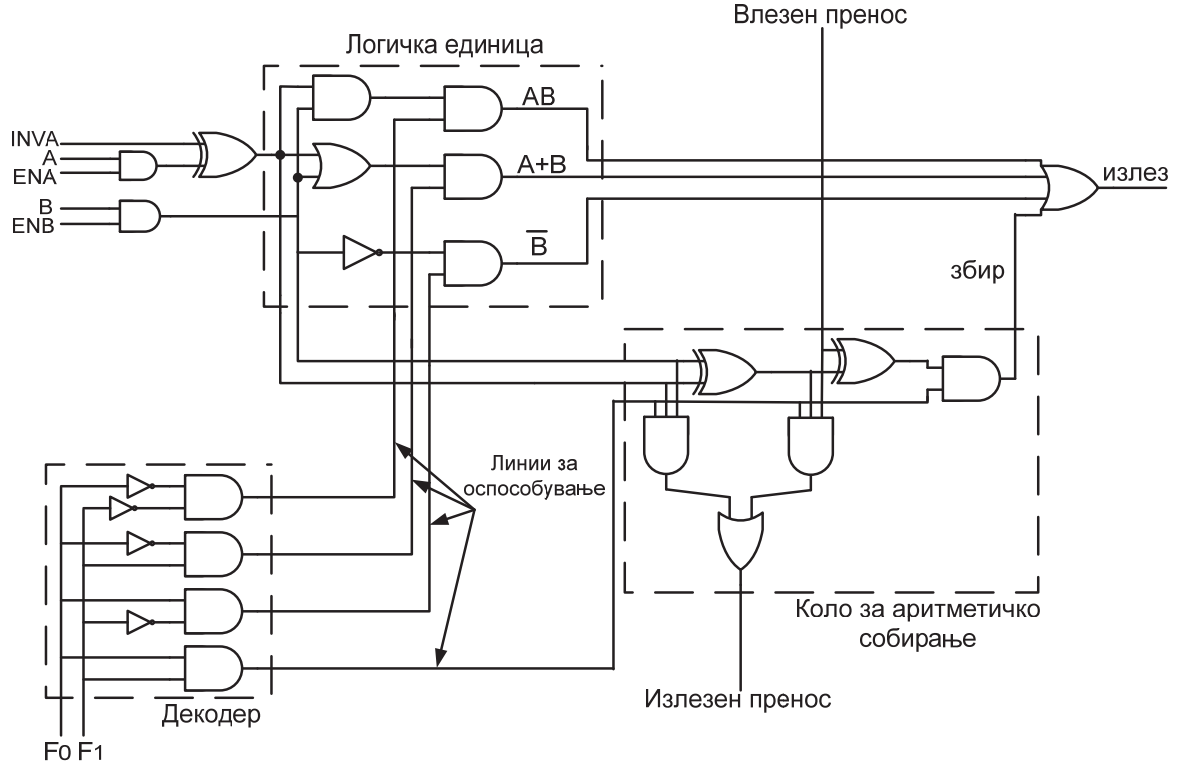
Аритметичко-логичката единица (Aritmetic Logical Unit) е составен дел на кој било микропроцесор. Како што ни кажува и самото име, оваа единица ги извршува ариметичко-логичките операции.

Ќе се запознаеме со логичкиот дијаграм на една едноставна аритметичко-логичка единица, која хардверски може да извршува само четири операции: логичко множење, логичко собирање, инвертор и аритметичко собирање. Сите други прости и сложени операции треба да се изведат софтверски преку овие четири операции.

Во горниот лев дел од сликата 1.20. се наоѓаат И-колото, ИЛИ-колото и инверторот. Во долниот десен дел се наоѓа аритметичката единица, односно колото за аритметичко собирање со кое се запознавме претходно. Во долниот лев дел од сликата 1.20. се наоѓа декодерот со два влеза F_0 и F_1. Која од

операциите AB, A+B (логично), негација од B или A+B (аритметичко) ќе биде извршена, зависи од битовите F_0 и F_1

F_0F_1	операција
0 0	A·B
0 1	A+B (логичко собирање)
1 0	Негација од B
1 1	A+ B (аритметичко собирање)



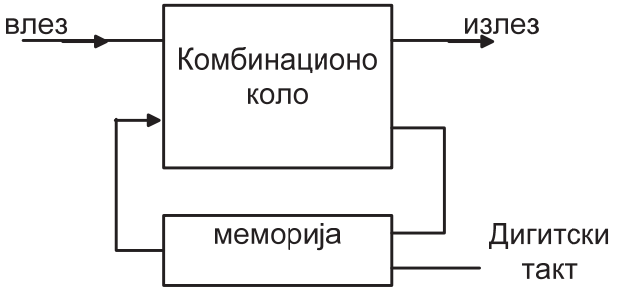
Слика 1.20. Шема на еднобитна аритметичко логичка единица

На излез од сите четири кола за пресметка (И колото, ИЛИ колото, инверторот и колото за собирање) се наоѓа по една И порта . Секоја од овие И порти има по два влеза.

1. Првиот влез е самиот резултат што доаѓа од колото за пресметка.
2. Вториот влез е сигнал што доаѓа од декодерот. Ако овој сигнал е единица И портата ќе биде отворена за да го пропушти добиениот резултат кон излезната ИЛИ-порта. Ако сигналот од декодерот е нула, тогаш И-портата е затворена. На трите влеза од излезното ИЛИ коло ќе има три нули и еден бит што одговара на пресметаниот резултат.

1.5. Секвенцијални кола

За разлика од комбинациските кола кај секвенцијалните кола состојбата на излезите зависи не само од моменталната состојба на влезовите туку и од нивната претходна состојба. Бидејќи секвенцијалните кола ги ,,памтат" претходните состојби на влезовите тие претставуваат еден вид мемориски модули. Структурата на секвенцијалните кола и нивните влезно излезни сигнали е прикажана на слика 1.21 .



Слика 1.21.Принцип на работа на секвенцијалните кола

Може да се заклучи дека кај секвенцијалните кола, врз состојбата на излезите големо влијание има времето како посебна физичка величина. Од таму доаѓа и терминот секвенцијални. Сите промени во колата се случуваат последователно односно зависат една од друга. За да се одреди вредноста на излезниот сигнал во следната временска секвенца Q_{t+1} освен вредноста на влезовите треба да се познава и состојбата на излезот во претходната секвенца Q_t.

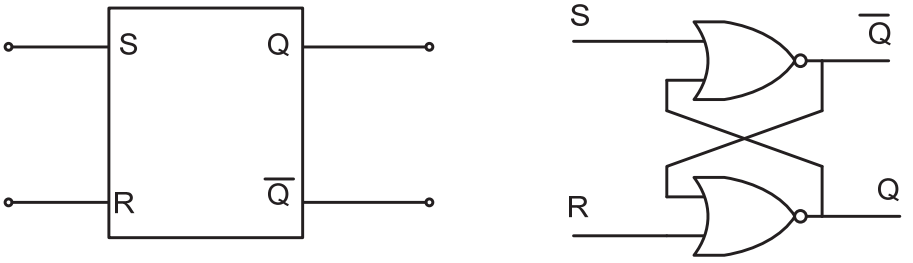
Секвенцијалните кола можат да бидат тактирачки (синхрони) или нетактирачки (асинхрони). Кај нетактирачките кола состојбата на излезите зависи само од промените на податочните влезови и овие промени се сосема случајни. Кај тактирачките кола вредноста на такт сигналот е услов промената на податочниот влез да предизвика промена и на податочниот излез. Сигналите кои дозволуваат промена на излезните сигнали под дејство на влезните се викаат активатори (англиски-trigger). Кај некои секвенцијални кола за да дојде до побуда потребно е дигитскиот такт односно активаторот да биде на високо ниво, а кај некои потребен е растечки или опаѓачки раб на дигитскиот такт.

Секвенцијални кола се флип флоповите, мултивибраторите, регистрите и бројачите. Со комбинирање на овие секвенцијални кола понатака се добиваат посложени мемориски чипови.

1.5.1. Флип-флоп

Флип флопот е најмал мемориски модул во кој може да се зачува информација од еден бит. Постојат повеќе видови флип-флопови: SR, JK, D и T. Работата на флип флоповите може да се анализира табеларно, аналитички и графички.

SR флип флопот е наједноставен по својата структура. Тој има два влеза: влез за сетирање S и влез за ресетирање R. Обично има два излеза Q и $\bar{Q}$, при што вториот излез претставува прв комплемент од првиот. Флип флиповите можат да бидат направено од НИ или НИЛИ порти. На слика 1.22. прикажани се симболот на SR флип флопот и неговата логичка шема кога истиот е изработен од НИЛИ порти. Од шемата се гледа дека постои позитивна повратна врска односно излезите се враќаат на влез.



Слика 1.22. Симбол и логичка шема на SR флип флоп

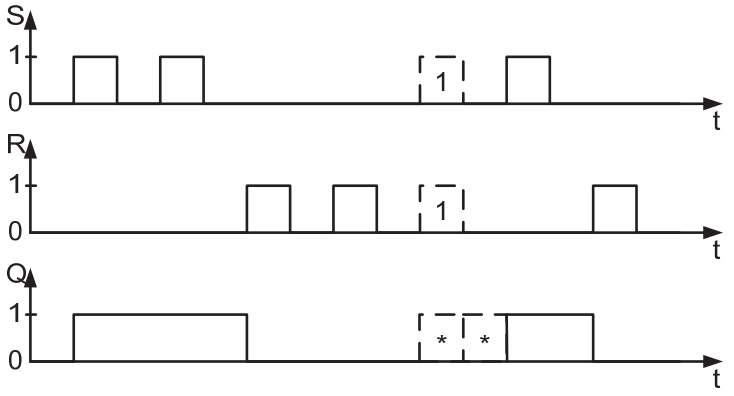
Табелата 1.11. претставува табела на вистинитост за SR флип флоп. Кога на влезот S ќе се појави логичка единица тогаш излезот Q ќе се сетира односно ќе биде на високо ниво. Кога на влезот R ќе се појави логичка единица излезот Q се ресетира односно ќе биде на ниско ниво. Излезот $\bar{Q}$ е секогаш на спротивно ниво од излезот Q. Состојбата S=1 и R=1 е недозволена бидејќи во тој случај двата излези се на ниско ниво. Состојбата на излезите на се менува кога и двата влеза се нули.

S	R	Q_{t+1}
0	0	Q_t
0	1	0
1	0	1
1	1	*

*= недозволена состојба

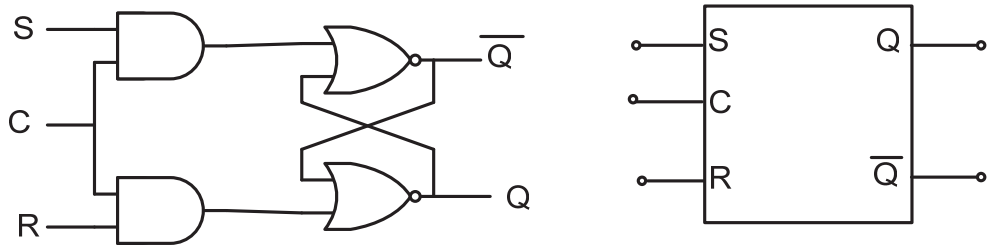
Табела 1.11. Табела на вистинитост за SR флип флоп

Временскиот дијаграм на флип флопот SR е прикажан на сликата 1.23.



Слика 1.23. Временски дијаграм за SR флип флоп

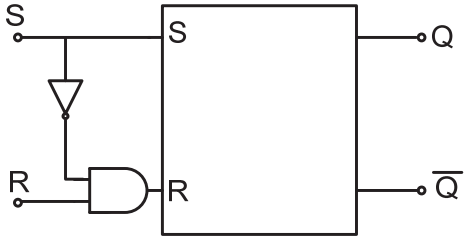
На слика 1.24. е прикажан тактирачки SR флип флоп. Овој флип флоп има три влеза: S, R и дигитски такт C (clock).



Слика 1.24. Логичка шема и симбол за тактирачки SR флип флоп

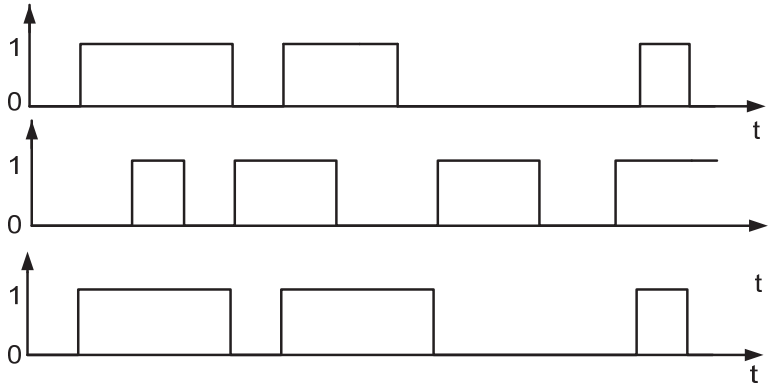
Дигитскиот такт се носи на влез од две И порти и тој решава кога И портите ќе бидат отворени за да низ нив поминат сигналите S и R. Кога дигитскиот такт е на ниско ниво тогаш двата излези не се менуваат, а кога дигитскиот такт е на високо ниво тогаш тактирачкиот флип флоп се однесува исто како и нетактирачки флип флоп.

Со цел да се одбегне несаканата состојба на влез S=1 и R=1 на еден од овие влезови може да му се даде поголем приоритет во однос на другиот. Тоа се постигнува со додавање на дополнителни логички кола на влез од SR флип флопот. Прашање е на кој од влезовите да му се даде поголем приоритет. На слика 1.25. е прикажан симболот на SR флип флоп со поголем приоритет на влезот за сетирање S.



Слика 1.25. Симбол на SR флип флоп со приоритет на S влезот

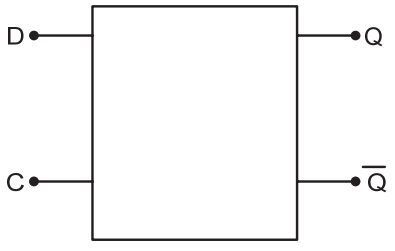
Доколку на влезот S пристигне логичка единица тогаш преку инверторот единицата ќе се претвори во нула. Таа нула ќе го направи влезот за ресетирање небитен бидејќи сигналот R помножен со нула ќе даде нула. На слика 1.26. прикажан е временскиот дијаграм за SR флип флоп со поголем приоритет на S влезот.



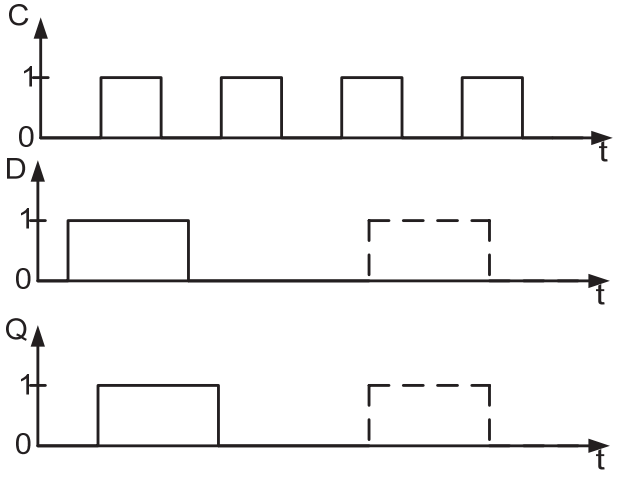
Слика 1.26. Временски дијаграм на SR флип флоп со поголем приоритет на S влезот

D флип флопот е еден од наједноставните во својата работа. Кај него промената на влез директно се пресликува на излез. D флип-флопот може да биде тактирачки и нетактирачки.

На сликата 1.27. е прикажан симболот на тактирачкиот D флип флопот. Кога дигитскиот такт е еднаков на еден тогаш излезот е еднаков на влезот Q=D, а кога дигитскиот такт е еднаков на нула тогаш излезот е еднаков на последно меморираната состојба пред тактот да падне од еден на нула $Q_{t+1}=Q_t$.



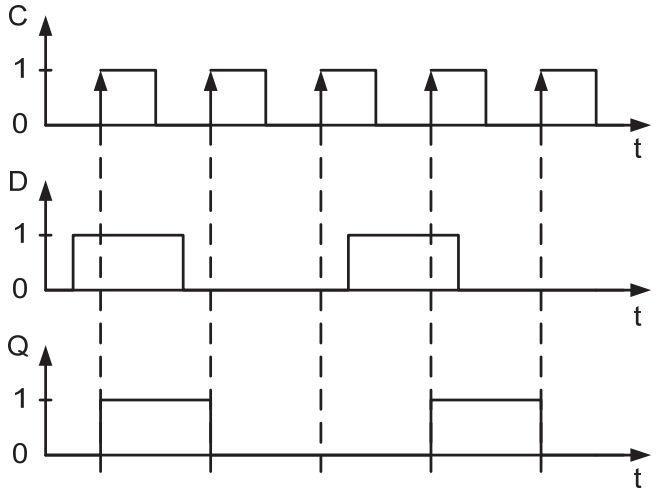
Слика 1.27. Симбол на тактирачки D флип флоп



Слика 1.28. Временски дијаграм на тактирачки D флип флоп

На слика 1.28. е прикажан временскиот дијаграм за тактирачкиот D флип флоп. Се додека тактот е еден, излезот ја следи состојбата на влез .

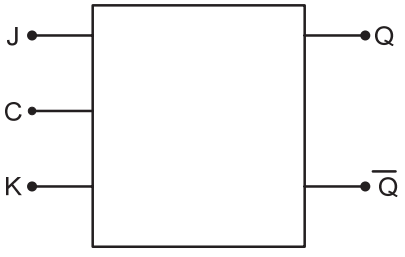
Но постојат флип флопови кај кои за да дојде до промена на излезот не е потребно високо ниво на дигитскиот такт, туку растечки или опаѓачки раб на тактот. На слика 1.29. е прикажан временскиот дијаграм на D флип флоп со растечки раб на дигитскиот такт како активатор. Од него можеме да заклучиме дека излезот станува еднаков на влезот во моментот кога имаме растечки раб на дигитскиот такт. Во оваа состојба излезот ќе остане до наредниот растечки раб. Имено влезниот сигнал може да се промени додека трае единицата на дигитскиот такт, но состојбата на излезот нема да се промени



Слика 1.29. Временски дијаграм за тактирачки D флип флоп со растечки раб како активатор

ЈК флип флоповите се едни од најупотебуваните флип флопови. Преку ЈК флип флопот е поправен недостатокот на SR флип флоповите, односно одбегната е недозволената состојба S=1 и R=1. Речиси секогаш активатор за ЈК флип флоповите е растечкиот или опаѓачкиот раб на дигитскиот такт , а понекогаш и двата рабови заедно.

Влезот J е влез за сетирање на флип флопот, а влезот К е за ресетирање. Доколку се активни и двата влеза тогаш се менува логичкото ниво на излезот од флип флопот односно ако е тој ресетиран тогаш истиот се сетира или ако е сетиран тогаш тој се ресетира. Подолу се прикажани симболот ЈК флип флоп и неговата табела на вистинитост.

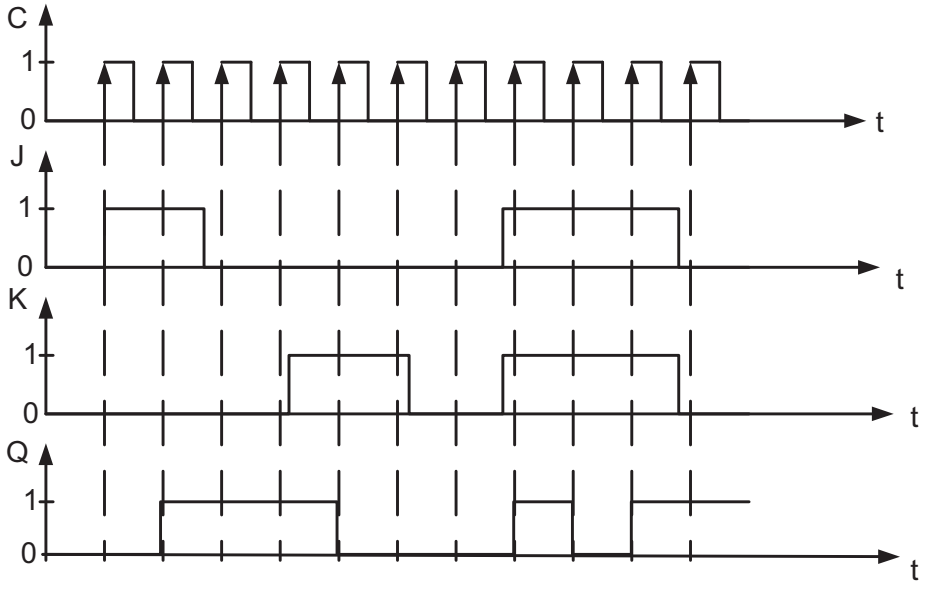


Слока 1.30. Симбол за ЈК флип флоп

J	К	Q_{t+1}
0	0	Q_t
0	1	1
1	0	0
1	1	$\overline{Q}_t$

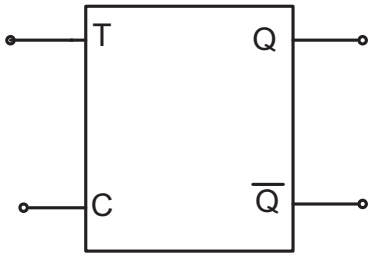
Табела 1.12. Табела на вистинитост за ЈК флип флоп

На слика 1.31. прикажан е временскиот дијаграм на ЈК флип флоп со растечки раб како активатор.



Слика 1.31. Временскиот дијаграм на ЈК флип флоп со растечки раб како активатор.

Ако влезовите J и K кусо се поврзат се добива **Т флип флоп**. На слика 1.32. е прикажан симболот на тактирачки Т флип флоп, а неговата табела на вистинитост е табелата 1.13.



Слика 1.32. Симбол на Т флип флоп

C	T	Q_{t+1}
0	0	Q_t
0	1	Q_t
1	0	Q_t
1	1	$\overline{Q}_t$

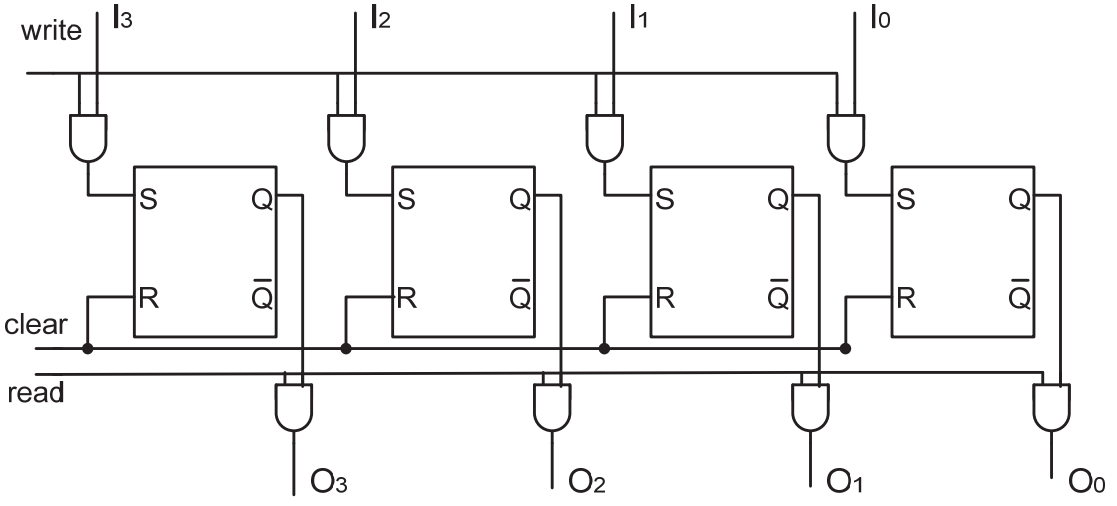
Табела 1.13. Табела на вистинитост за Т флип флоп

Кога дигитскиот такт е на ниско ниво состојбата на флип флопот не се менува. Состојбата на флип флопот не се менува и кога дигитскиот такт е на високо ниво, но влезот Т е еднаков на логичка нула. За да се промени состојбата на Т флип флопот потребно е дигитскиот такт да биде на високо ниво и на Т влезот да се појави логичка единица. Сосотојбата се менува така што се комплементира претходната вредност на излезот Q.

1.5.2. Регистри

Регистрите се поголеми мемориски модули од флип флоповите. Колку бита може да собере еден регистар зависи од тоа од колку флип-флопови е составен (за секој бит по еден флип-флоп). Постојат различни видови на регистри , но најчесто користени се стационарните и поместувачките регистри.

Стационарните регистри служат за привремено чување на информација од неколку битови. На слика 1.33. е прикажан четири битен стационарен регистар со паралелен влез и паралелен излез. Постојат три контролни сигнали. Clear служи за ресетирање на SR флип флоповите, Write за впишување на битови и Read за читање на претходно внесените битови. Пред да започне впишувањето на податоци флип флоповите се ресетираат преку поставување на сигналот Clear на високо ниво. После ресетирањето сигналот Clear повторно се поставува на ниско ниво.



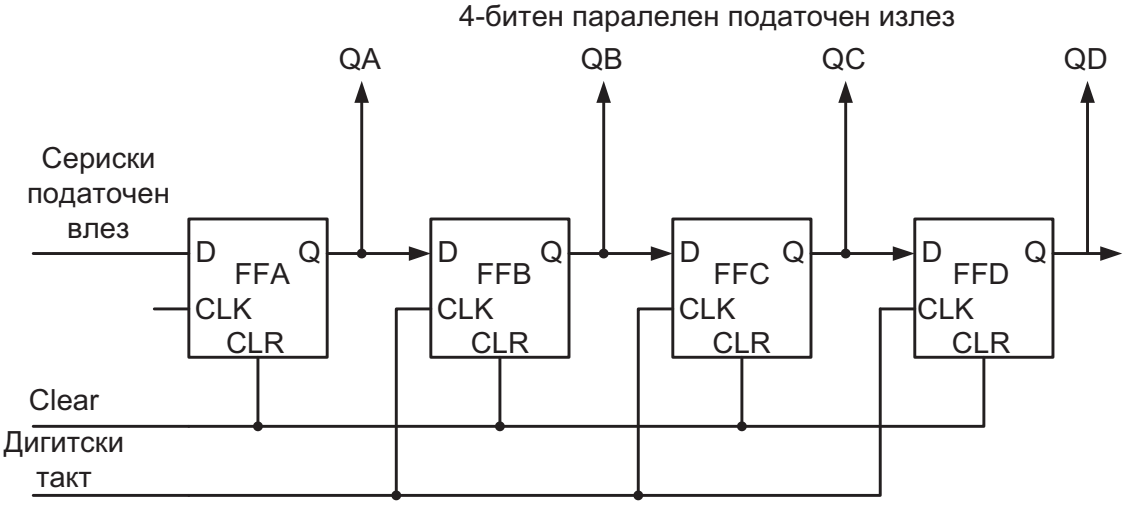
Слика 1.33. Логичка шема на стационарен регистар

Кога впишуваме податоци сигналот Write треба да биде на високо ниво како би се отвориле И портите поврзани на влезовите за сетирање на секој од флип флоповите. Влезните сигнали (Input) од I_0 до I_3 ќе се пресликаат на излезите од флип флоповите. Ако влезниот сигнал I е логичка нула флип флопот нема да се сетира и состојбата на неговиот излез ќе остане логичка нула, исто како после ресетирањето. Ако сигналот I е еднаков на логичка единица тогаш флип флопот ќе се сетира односно Q=1.

Кога читаме потребно е да се активира сигналот Read. Ако Read =1 се отвараат излезните И порти кои се приклучени на излезот од секој флип флоп. Со отварањето на овие порти сигналите од излезите на флип флоповите се пренесуваат до излезите (Output) O_0, O_1, O_2 и O_3.

Поместувачките регистри се составени од сериски врзани флип флопови при што на влез од секој следен флип флоп доаѓа излезниот сигнал од претходниот флип флоп. На таков начин битот од низата кој доаѓа на влез од првиот флип флоп ќе се движи од еден во друг флип флоп , од лево кон десно. Постојат четири видови поместувачки регистри: со сериски влез и паралелен излез (Serial In to Parallel Out -SIPO), со сериски влез и сериски излез (Serial in to Serial Out –SISO), со паралелен влез и паралелен излез (Parallel In to Parallel out –PIPO) и со паралелен влез и сериски излез (Parallel In to Serial Out -PISO).

На слика 1.34. е прикажан поместувачки регистар со сериски влез и паралелен излез.



Слика 1.34. Логичка шема на поместувачки регистар со сериски влез и паралелен излез

Сите четири флип флопови користат заеднички дигитски такт. На почетокот претпоставуваме дека сите флип флопови, од FFA до FFD биле ресетирани (преку влезот Clear) и сите податочни излези од QA до QD се еднакви на нула. Ако во првиот дигитски такт на влез од првиот флип флоп FFA се појави логичка единица тогаш тој ќе се сетира односно QA=1. Во вториот и сите следни дигитски такта претпоставуваме дека влезот на првиот флип флоп ќе биде на ниско ниво. Во вториот дигитски такт првиот флип флоп ќе се ресетира QA=0, а излезот на вториот флип флоп и QB ќе стане логичка единици. Логичката едниница се поместила за едно место во десно. Кога ќе пристигне третиот дигитски такт логичката единица ќе биде во третиот флип флоп. До петтиот дигитски такт логичката единица ќе ги помине сите флип флопови и сите тие повторно ќе имаат нули на своите излези бидејќи после првиот дигитски такт серискиот влез на регистарот остана на константно ниско ниво.

Во табелата 1.14 дадени се содржините на сите четири флип флопови за време на петте дигитски такта. Во четвртиот такт состојбата на излезите е еднаква на четирибитниот податок, внесен во регистарот преку серискиот влез,

бит по бит. Ова значи дека поместувачкиот регистар го претворил серискиот податочен сигнал во паралелен податочен сигнал.

Реден број на дигитски такт	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	0	0	0	0

Табела 1.14 Табела на вистинитост за поместувачки регистар со сериски влез и паралелен влез

Спротивно на поместувачкиот регистар со сериски влез и паралелен излез, поместувачкиот регистар со паралелен влез и сериски излез врши претварање на влезниот паралелен податочен збор во сериски податочен сигнал. Овие поместувачки регистри се користат при мултиплексирање на повеќе различни влезни линии на една сериска линија за брз пренос на податоци.

Поместувачкиот регистар со сериски влез/ сериски излез и регистарот со паралелен влез/паралелен излез се користат за привремено чување на податоци, односно внесуваат време на доцнење. Колку ќе изнесува временската разлика меѓу сигналите на влез и излез зависи од бројот на флип флопови во регистрите и тој број обично изнесува 4, 8, 16, итн.

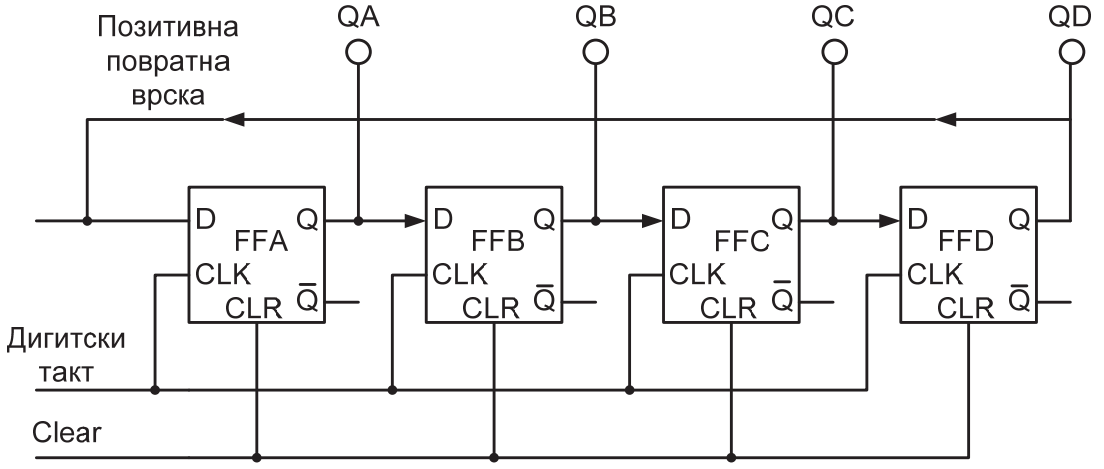
1.5.3. Бројачи

Исто како поместувачките регистри така и бројачите се секвенцијални кола составени од повеќе флип флопови. Постојат различни видови на бројачи: асинхрони или синхрони бројачи, бројачи кон напред или бројачи кон назад и бинарни , декадни или хексадецимални бројачи.

Ако излезот од поместувачкиот регистар се поврзе со неговиот влез (позитивна повратна врска) тогаш се добива **кружен бројач**. На слика 1.35. е прикажана шемата на синхрон кружен бројач составен од четири флип флопови. Кружниот бројач се нарекува синхрон бидејќи сите негови флип флопови користат ист дигитски такт.

Четирибитниот податок постојано кружи помеѓу четирите флип флопови и ова се повторува на секој четврти дигитски такт. Но пред да започне

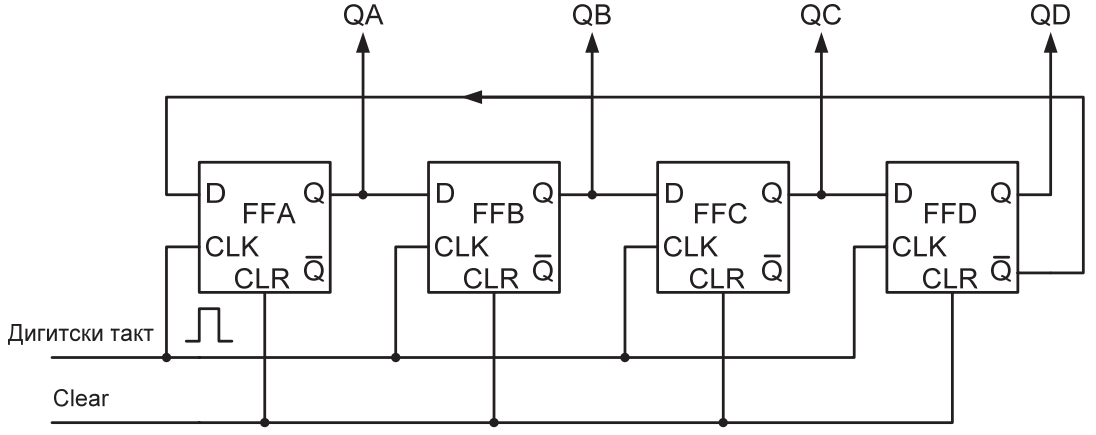
кружењето на податокот истиот треба да биде внесен во бројачот. На почетокот го ресетираме бројачот преку сигналот Clear.



Слика 1.35. Логичка шема на кружен бројач

Потоа на влез од првиот флип флоп се појавува логичка единица , импулс кој трае само еден дигитски такт. Имено се работи за импулс активатор кој е потребен за да кружниот бројач почне да работи.

Ако наместо неинвертирачкиот излез го поврземе инвертирачкиот излез со влезот на кружниот бројач ќе добиеме сосема поинаков бројач таканаречен **Џонсонов кружен бројач**. Неговата шема е прикажана на слика 1.36.



Слика 1.36. Логичка шема на Џонсонов кружен бројач

Наместо четири Џонсоновиот бројач има осум различни состојби. Во горниот наведен пример состојби на кружниот бројач се 1000, 0100, 0010, 0001. Џонсоновиот бројач прво брои нанапред, а после првите четири дигитски такта почнува да брои наназад. Во табелата 1.15. дадени се осумте различни состојби кои постојано се повторуваат после секој осми дигитски такт.

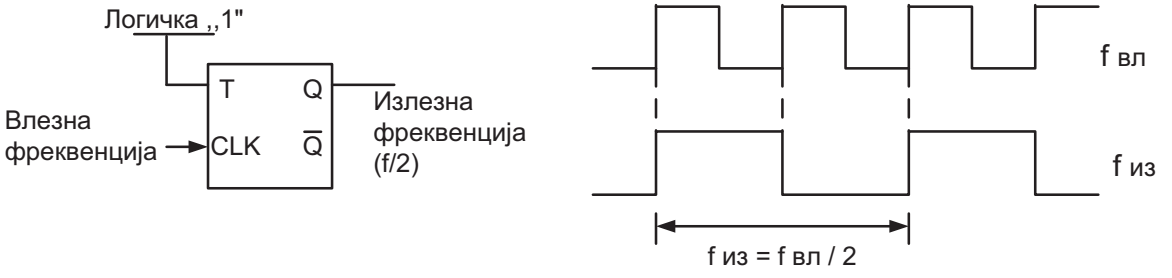
Кај кружниот бројач единицата внесена во регистарот после неговото ресетирање беше единствена и таа постојано кружеше, поместувајќи се од

еден во друг флип флоп. Кај Џонсоновиот бројач во првите четири состојби бројот на единици постојано се зголемува бидејќи на инвертирачкиот излез од FFD имаме логичка единица и таа единица ќе ги сетира сите четири флип флопови. После тоа инвертирачкиот излез станува логичка нула, флип флоповите ќе почнат да се ресетираат, почнувајќи од FFA кон FFD (последните четири состојби).

FFA	FFB	FFC	FFD
0	0	0	0
1	0	0	0
1	1	0	0
1	1	1	0
1	1	1	1
0	1	1	1
0	0	1	1
0	0	0	1

Табела 1.15. Табела на вистинитост за Џонсонов кружен бројач

Најчесто користен флип флоп кај бројачите е Т флип флопот. Тој често се користи како **делител на фреквенција**. Ако Т влезот се поврзе на високо ниво тогаш на неинвертирачкиот излез ќе добиеме дигитски такт со двапати помала фреквенција од онаа на влезниот дигитски такт. Ваков делител на фреквенција е прикажан на сликата 1.37. На истата слика е прикажан влезниот дигитски такт и тактот добиен на неинвертирачкиот излез Q.

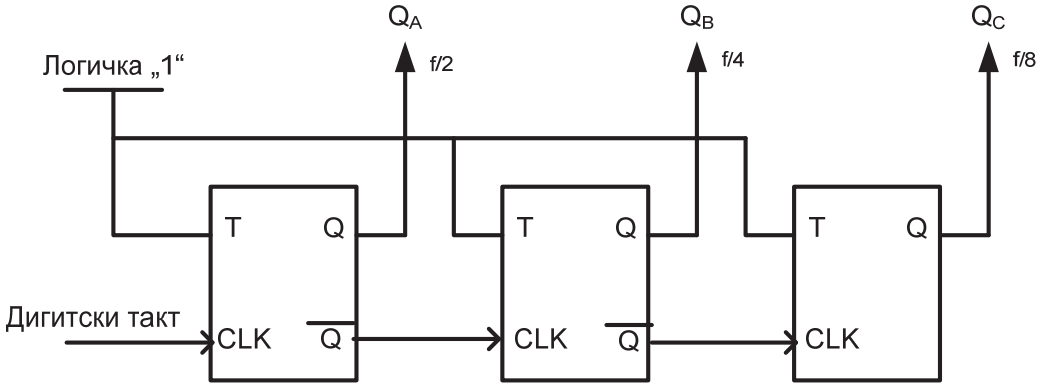


Слика 1.37. Т флип флоп како делител на фреквенција

Со доаѓањето на секој нов растечки раб на влезниот дигитски такт вредноста на излезот Q се менува односно ако вредноста на излезниот сигнал била еден тогаш таа станува нула и обратно. Имено на излез од Т флип флопот добиваме една периода за две периоди на влезниот дигитски такт.
Ако сериски поврземе два Т флип флопови чии Т влезови се врзани на високо ниво тогаш делителот на фреквенцијата ќе изнесува 4. Ако сериски поврземе три Т флип флопови тогаш излезната фреквенција ќе биде осум пати помала од влезната фреквенција итн. Имено делителот на фреквенцијата се пресметува според равенката 2^n каде n е еднаков на бројот на флип флопови. Всушност преку сериско поврзување на повеќе Т флип флопови чии Т влезови

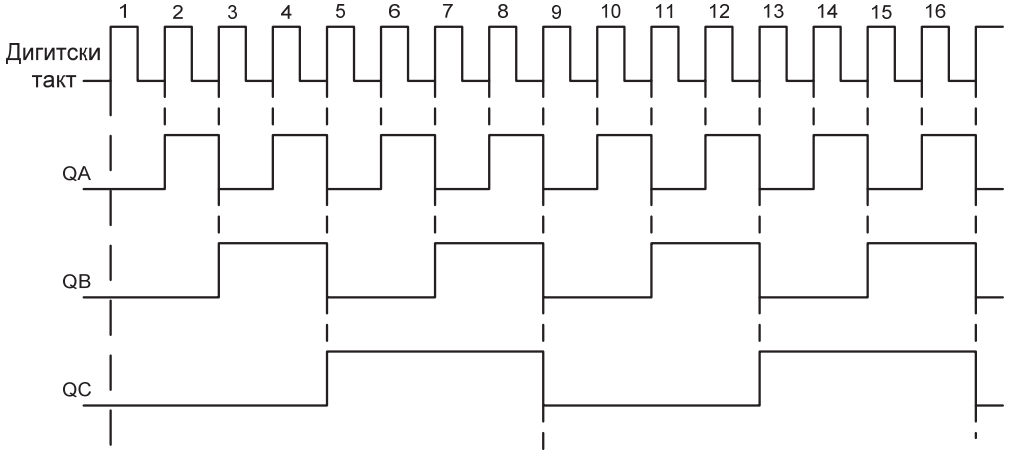
се поставени на високо ниво се добиваат **асинхрони бројачи**. Тие се нарекуваат асинхрони бидејќи нивните флип флопови не користат заеднички дигитски такт. Дигитскиот такт се носи на влез од првиот флип флоп, сигналот од инверирачкиот излез Q на првиот флип флоп се носи на влезот за дигитски такт од вториот Т флип флоп, сигналот од инвертирачкиот излез Q од вториот флип флоп се носи на влезот за дигитски такт на третиот флип флоп итн.

На слика 1.38. е прикажана шемата на три битен асинхрон бројач кој брои кон напред. Претпоставуваме дека за време на првиот дигитски такт сите Т флип флопови се ресетирани. Со доаѓањето на вториот дигитски такт првиот флип флоп се сетира. Со доаѓањето на третиот такт првиот флип флоп се ресетира , а вториот се сетира. Со следниот такт првиот флип флоп по втор пат се сетира , вториот останува сетиран. Кога ќе дојде на ред осмиот такт сите три флип флопови ќе бидат сетирани. После осмиот такт, односно во деветиот дигитски такт сите три флип флопови ќе се бидат ресетирани и постапката ќе почне од почеток.



Слика 1.38. Логичка шема на асинхрон бројач

Промените на состојбата на трите излези од бројачот можат да се прикажат преку временски дијаграм како на слика 1.39.



Слика 1.39. Временски дијаграм за асинхрон бројач

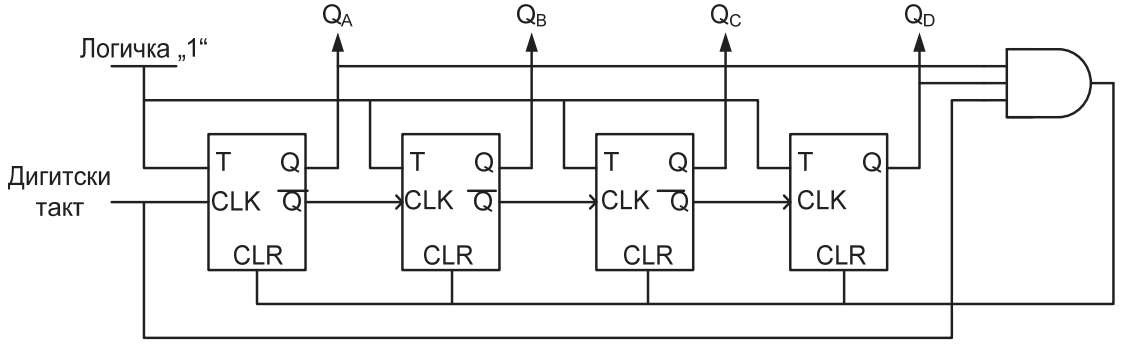
Ако за секој дигитски такт ги запишеме состојбите на излезите QA, QB и QC ќе ја добиеме табелата на вистинитост на бројачот 1.16.

Дигитски такт	Излезна комбинација			Децимална вредност
	QC	QB	QA	
0	0	0	0	0
1	0	0	1	1
2	0	1	0	2
3	0	1	1	3
4	1	0	0	4
5	1	0	1	5
6	1	1	0	6
7	1	1	1	7

Табела 1.16. Табела на вистинитост за трибитен асинхрон бројач

Во табелата на вистинитост освен состојбата на излезите дадена е и децималната вредност која ја даваат сите три излези заедно. Оваа децимална вредност всушност е вредноста на бројачот. Со доаѓањето на секој нов дигитски такт вредноста на бројачот се зголемува за еден. Максималната вредност која може да ја изброи овој бројач е седум. После тоа доаѓа до ресетирање на бројачот односно тој повторно брои од нула. Може да се заклучи дека три битниот бројач има осум различни состојби почнувајќи од 000 до 111. Бројот на состојби на било кој бројач е еднаков на 2^n каде n е број на флип флопови во бројачот, а максималната вредност која може да ја изброи бројачот е за еден помала од бројот на состојби.

На слика 1.40. е прикажан **асинхрон декаден бројач**.



Слика 1.40. Логичка шема на асинхрон декаден бројач

Бројачот прикажан на слика 1.40. е составен од четири Т флип флопови чии Т влезови се поврзани на високо ниво. Ова значи дека делителот на фреквенција ќе изнесува 2^4=16. Но, овој бројач нема да брои од 0 до 15. Бројачот се ресетира односно тој почнува да брои од нула кога состојбата на излезите од четирите флип флопови изнесува 1001=9. Овој бројач се нарекува декаден бројач бидејќи има десет различни состојби, почнувајќи од 0000 до

1001. Ова се всушност BCD кодовите на цифрите од декадниот броен систем, па поради тоа овој бројач се нарекува и BCD бројач.

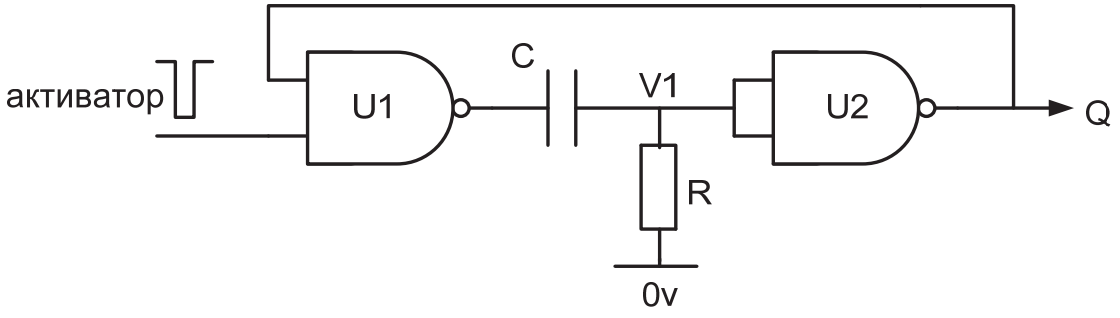
Бројачот се расетира после добивањето на вредноста девет благодарејќи на И колото од слика 1.40. Ова И коло има три влеза: QA, QD и влезниот дигитски такт. Кога сите три влеза ќе станат еднакви на еден И колото ќе даде единица на својот излез и ќе се активира сигналот за ресетирање (clear).

1.6. Мултивибратори

Работата на флип флоповите, регистрите, бројачите и меморискиите чипови не може да се замисли без употреба на дигитски такт. Мултивибраторите се саквенцијални кола коои генерираат дигитски такт. Општата поделба на мултивибраторите е на астабилни и моностабилни. Астабилните мултивибратори генерираат периодични поворки од импулси, а моностабилните мултивибратори генерираат само еден импулс или пауза на својот излез.

1.6.1. Моностабилен мултивибратор

На слика 1.41. е прикажан моностабилен мултивибратор составен од две НИ кола, отпорник R, кондензатор C и позитивна повратна врска. Овој мултивибратор има една стабилна состојба на логичка единица и една преодна состојба на логичка нула. Кога активаторот не се менува мултивибраторот се наоѓа во стабилна состојба. Во случај на промена на активаторот мултивибратототот преминува во преодна состојба, но како што ни кажува самото име оваа состојба не е постојана односно после извесно време тој самиот ќе се врати во стабилна состојба.



Слика 1.41. Шема на моностабилен мултивибратор

Во стабилна состојба низ отпорникот R не тече струја, на влез од колото U2 имаме логичка нула, а на неговиот излез логичка единица. Во стабилна

состојба активаторот е еднаков на логичка единица , па двете единици на влез од колото U1 ќе дадат логичка нула на неговиот излез. Во стабилна состојба двата краја на кондензаторот се наоѓаат на низок потенцијал што значи низ него не тече струја.

Ако на влезот за активаторот се појави логичка нула тогаш излезот на колото U1 ќе стане логичка единица. Бидејќи напонот на краевите од кондензаторот не може моментално да се промени влезот на колото U2 исто така ќе стане логичка единица . Како резултат на ова ќе се промени и излезот на U2 колото тој ќе стане логичка нула. Поради логичката нула на излезот од U2 мултивибраторот ќе остане во оваа состојба дури и кога ќе исчезне паузата на влезот за активаторот.

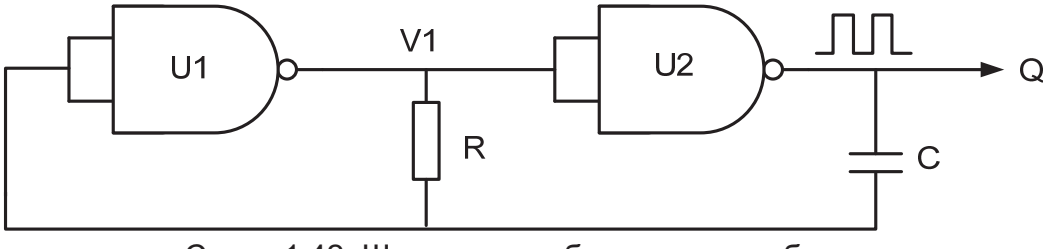
Додека кондензаторот се полни низ него тече струја, а откако тој ќе се наполни струјата престанува да тече и потенцијалот во точката V1 повторно ќе падне на логичка нула. Ова предизвикува излезот на колото U2 повторно да стане логичка единица, а излезот на колото U1 логичка нула. Сега кондензаторот ќе почне да се празни преку отпорникот R. Мултивибраторот конечно ќе се врати во својата стабилна состојба.

Времетраењето на преодната состојба односно времетраењето на паузата на излез од мултивибраторот зависи од константата на RC колото и се пресметува според релацијата T=0,7 RC.

1.6.2. Астабилен мултивибратор

Астабилниот мултивибратор е слободен осцилатор без стабилна состојба. Тој има две преодни состојби: состојба на логичка нула и состојба на логичка единица. Неговиот излез постојано се менува од ниско на високо ниво и обратно и на тој начин тој генерира периодична поворка од правоаголни импулси чија перода ќе зависи од константата на RC колото.

На слика 1.42. е прикажана шемата на астабилен мултивибратор направен од НИ порти. Влезовите на овие НИ порти се кусо врзани, па тие функционираат како инвертори.



Слика 1.42. Шема на астабилен мултивибратор

Ако излезот на второто НИ коло U2 е еднаков на логичка единица тогаш на неговиот влез има логичка нула. Едниот крај на кондензаторот е поврзан со излезот на U2 колото и е на логичка единица, а другиот крај преку отпорникот R е поврзан со влезот на колото U1 и е на логичка нула. Кондензаторот ќе почне да се полни.

Додека кондензаторот се полни потенцијалот во точката меѓу кондензаторот и отпорникот постојано опаѓа. Оваа точка е поврзана со влезот на првото НИ коло U1. Во еден момент влезот на колото U1 ќе стане логичка нула, а неговиот излез ќе се промени во логичка единица. Ова ќе предизвика промена на излезот од колото U2 од логичка единица на логичка нула. Кондензаторот ќе почне да се празни се додека влезот на колото U2 повторно не се промени и циклусот почне одново.

Можеме да заклучиме дека кога кондензаторот се полни на излез од мултивибраторот имаме логичка единица (импулс), а кога кондензаторот се празни на излез имаме логичка нула (пауза). Периодата на добиениот дигитски такт се пресметува според равенката T=2,2RC.

1.7. Полупроводнички мемории

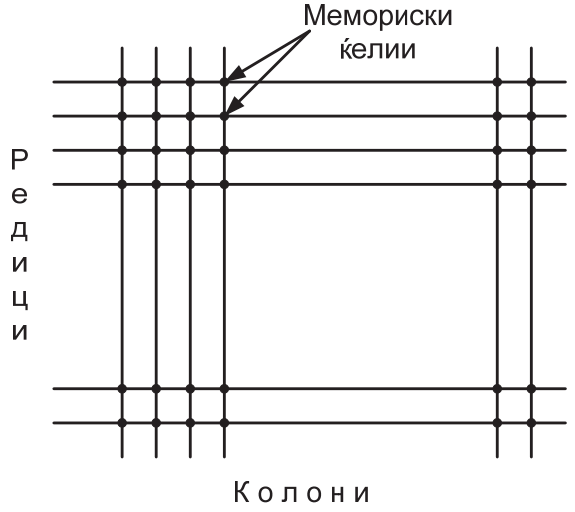
Во зависност од видот на материјалите од кои се изработени мемориите се поделени на полупроводнички и магнетни мемории. ROM и RAM мемориите се полупроводнички мемории. Не може да се замисли компјутер кој во својот состав нема да има RAM и ROM меморија. Полупроводничките мемории се изработени во форма на интегрирани кола. Пред да се запознаеме со функцијата и видовите RAM и ROM мемории ќе ја разгледаме организацијата на мемориските чипови и нивните поважни влезни и излезни пинови.

1.7.1. Организација на мемориски чип

Флип флоповите се најмали мемориски единици кои можат да запаметат информација од еден бит. Регистрите се составени од повеќе флип флопови и тие можат да бидат 4, 8, 16 битни итн. После регистрите следуваат мемориските чипови чии капацитет може да достигне и до неколку мега бајти.

Мемориските чипови се организирани во форма на матрици. Матрицата е структура составена од нормално поставени редици и колони, како што е прикажано на слика 1.43. Во спојните точки на редиците и колоните има по една мемориска ќелија која може да меморира информација од еден бит. Бројот на редици е еднаков на бројот на мемориски локации, а бројот на колони

ни покажува колку битови можат да се меморираат во една мемориска локација. Бројот на редици и колони го дефинираат бројот на адресни и податочни пинови на еден мемориски чип.



Слика 1.43. Структура на мемориски чип

Адресните пинови служат за селекција на една од многуте мемориски локации во состав на еден мемориски чип. Зависноста меѓу бројот на адресни пинови и бројот на мемориски локации е дадена со ралацијата

$$\text{број на мемориски локации}=2^{\text{број на адресни пинови}}$$

Степенот 2^{10} е еднаков на 1024 или скратено се запишува 1К (кило). Степенот 2^{20} е еднаков на 1048576 или скратено се нарекува 1М (мега). Степенот 2^{30} е еднаков на 1073741824 или скратено се запишува 1G (гига). Исто така доколку ги разложиме степените ќе добиеме дека 1G=1К·1К·1К или 1М=1К·1К. За поедноставна пресметка на степените најдобро е експонентот да се разложи на единици и десетки при што степенот на десетки ќе го дадеде префиксот. Ова е прикажано во примерот 1.12.

Пример 1.12: Пресметај го бројот на мемориски локаци во еден мемориски чип ако бројот на адресни пинови изнесува 17.
Решение: $2^{17}=2^{7}\cdot 2^{10}=128\cdot 1024=128К$
Овој мемориски чип содржи 131072 мемориски локации или скратено кажано 128 кило мемориски локации.

Најнезначајниот адресен пин се означува со A_0, а најзначајниот A_{n-1}, каде што индексот n е еднаков на вкупниот број адресни влезови. На пример, ако меморискиот чип поседува 10 адресни влеза, тогаш тие се бележат од A_0 до A_9. Адресните пинови се всушност влезови за адресниот декодер сместен во внатрешноста на меморискиот чип. Адресниот декодер служи да избере една мемориска локација од чипот во зависност од влезната комбинација на

адресни битови. Секоја мемориска локација има своја адреса, која, всушност, претставува единствена комбинација од адресни битови. Адресата е единствена бидејќи не може да се случи во рамките на ист мемориски чип две или повеќе локации да имаат иста адреса.

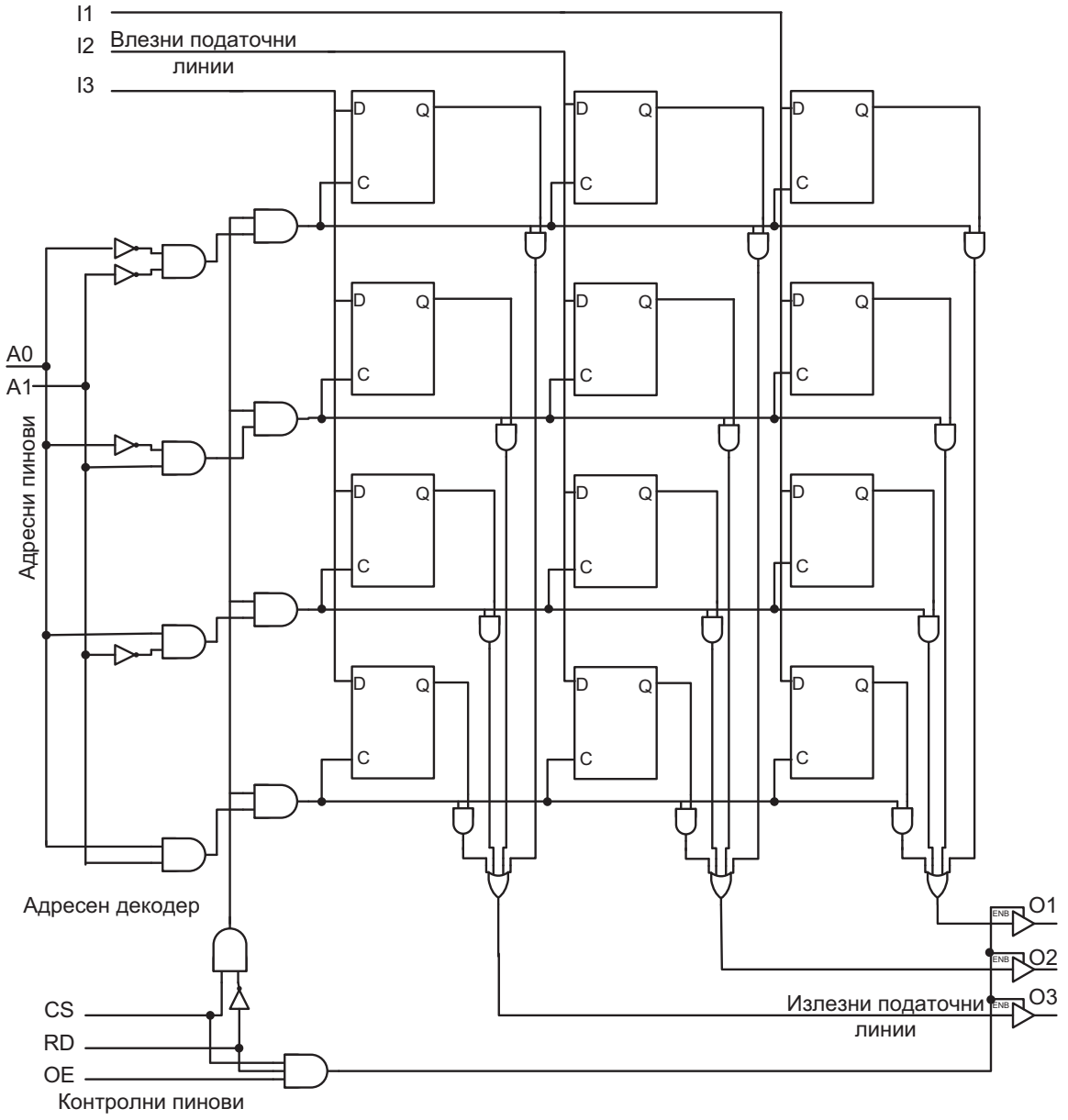
Податочни пинови. Податочните пинови служат за внесување (впишување) и за изнесување (читање) податоци од еден мемориски чип. Податочните пинови можат да бидат само влезни, излезни или двонасочни. Ако се влезни се бележат со буквата I што претставува прва буква од англискиот збор Input што во превод значи влез. Ако се излезни се бележат со буквата O што е прва буква од англискиот збор Output што значи излез. Ако се двонасочни тогаш се бележат со буквите IO или со буквата D што е првата буква од англискиот збор *data* што во превод значи податок. Бројот на податочни пинови на еден мемориски чип зависи од бројот на колони во мемориската матрица, односно широчината на една мемориска локација. Под ширина на мемориска локација се подразбира бројот на битови што можат да се сместат во неа. Најголем дел од мемориските чипови се 8-битни што значи дека секоја мемориска локација може да собере 8 бита или 1 бајт. Покрај 8-битните мемории, постојат 16-битни, 4-битни и еднобитни. Во каталошките опции на мемориските чипови, покрај информацијата за капацитетот, дадена е дополнителна информација за широчината на мемориските локации. На пример, 1Кx8 значи дека чипот има 1К мемориски локации при што секоја локација може да собере 8 бита. Ова значи дека тој чип може да собере вкупно 8К бита. Чипот 16Кx1 има 16К локации и секоја локација собира само по еден бит.

Секој мемориски чип има еден или повеќе **пинови за селекција.** Имено, во сметачот има голем број мемориски чипови. Пиновите за селекција треба да издвојат само еден од нив за да во него се впише или се прочита податок. Овие пинови се обележани со една од ознаките CS (chip select), CE (chip enable) или едноставно S (select). Најчесто RAM меморииите ја користат ознаката $\overline{CS}$ или $\overline{S}$, а ROM кратенката $\overline{CE}$. Може да се забележи дека кратенките $\overline{CS}$, $\overline{CE}$ и $\overline{S}$ се ставени под негација, што значи овие пинови ќе бидат активни кога се на ниско ниво. Кога овие пинови се на високо ниво, меморискиот чип е пасивен, односно од него не може да се изнесуваат или да се внесуваат податоци.

Преку **контролните пинови** се дефинира операцијата што ќе се извршува, односно дали ќе се внесуваат или ќе изнесуваат податоци. ROM има еден контролен пин $\overline{OE}$ (output enable) или $\overline{G}$ (gate). За да се изнесат податоци преку податочните пинови, пиновите $\overline{CE}$ и $\overline{CS}$ мора да бидат на ниско ниво. Ако пинот $\overline{OE}$ е на високо ниво, тогаш податочните пинови ќе бидат онеспособени и за нив велиме дека се во состојба на висока импеданса. RAM меморијата може да има еден или два контролни пина. Кога има еден контролен пин, тогаш тој се бележи со R/$\overline{W}$ (read/$\overline{write}$). Кога овој пин е ниско ниво тогаш се впишуваат

нови податоци, а кога е на високо ниво, се читаат претходно внесените податоци. Ако се користат два контролни пина, тогаш тоа се WE (write enable) за пишување и OE (output enable) за читање. Двата контролни пина не смеат да бидат активни истовремено, а доколку и двата се пасивни, тогаш за податочните пинови велиме дека се во состојба на висока импеданса.

На сликата 1.44. е прикажан логички дијаграм на мемориски чип составен од 12 тактирачки флип-флопови.



Слика 1.44. Организација на мемориски чип 4×3

Секој флип-флоп може да запамти информација од еден бит, што значи целиот мемориски чип може да запамти вкупно 12 бита. 12-те флип-флопови се распоредени во четири редици и три колони што значи тој има четири мемориски локации и во секоја локација може да се запамтат по три бита. Овој

чип има три податочни влезни пинови кои се означуваат со I_1,I_2 и I_3 и три посебни податочни излезни пинови со ознаката O_1,O_2 и O_3.

Адресниот декодер служи за селекција на една од четирите мемориски локации. На табелата 1.17. се прикажани комбинациите од двата адресни влеза, кои обезбедуваат пристап до само една мемориска локација .

A_0	A_1	Пристап
0	0	Мемориска локација број 1
0	1	Мемориска локација број 2
1	0	Мемориска локација број 3
1	1	Мемориска локација број 4

Табела 1.17. Табела на вистинитост за адресниот декодер

Ако влезот RD е на високо ниво, тогаш ќе читаме податоци од некоја мемориска локација, а ако RD=0, тогаш нема да читаме, туку ќе пишуваме нови податоци. Влезот OE врши оспособување на излезните бафери.

1.7.2. RAM мемории

RAM меморијата е привремена меморија. Таа ги губи податоците после исклучувањето на напојувањето. Најголема разлика меѓу RAM и ROM мемориите е таа што RAM меморијата се програмира додека работи компјутерот, а ROM меморијата се програмира пред да се вгради во компјутерот. RAM мемориите се многу брзи мемории. Нивната брзина се изразува преку времето на пристап. Под време на пристап се подразбира времето поминато од моментот кога меморискиот чип ќе ја прими адресната комбинација на влез до моментот на појавување на податочните битови на излез. Кај RAM мемориите времето на пристап изнесува неколку нано секунди.

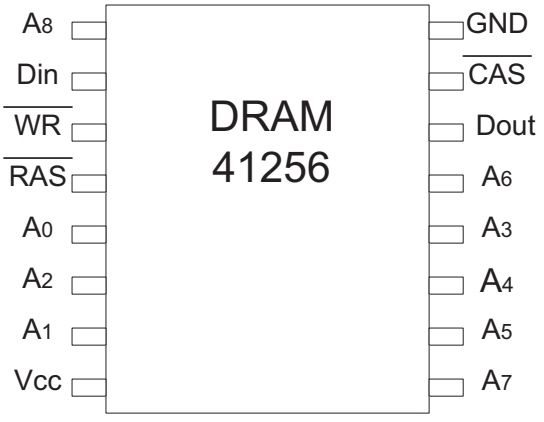
Според начинот на чување и освежувањето на податоците, постојат два вида RAM мемории: DRAM и SRAM мемории.

SRAM (Static RAM) е статичката RAM меморија и во своите ќелии има D флип-флопови. Нејзе не ѝ е потребно освежување и е многу побрза од DRAM меморијата. Поради високата цена на чинење, во сметачот се наоѓа во помали количини, обично како кеш-меморија

DRAM (Dinamic RAM) или динамичката RAM меморија која во своите ќелии содржат еден транзистор и еден кондензатор. Кога кондензаторот е полн, тоа значи состојба на високо ниво (единица), а кога е празен, значи состојба на ниско ниво (нула). Овој вид меморија има многу кратко време на

памтење, бидејќи со тек на време доаѓа до истекување на електрицитетот од кондензаторот. Поради тоа е потребно податоците во DRAM меморијата постојано да се освежуваат (повторно да се подигнува напонот). DRAM меморијата е многу спора, но добри страни се ниската цена и големата густина (голем број бита по еден чип). Постојат неколку видови DRAM мемории. Најстар вид е FPM (Fast Page Mode) DRAM. Тоа е мемориски чип организиран во форма на матрица од битови. За да се стигне до саканиот бит, мора да се знае бројот на редица и бројот на колона. Подоцна се јавува EDO (Extended Data Output) DRAM, кој овозможува второто повикување на меморијата да почне пред да заврши првото. Со ова се зголемува количеството податоци што можат да се извлечат од меморијата во дадено време.

Од SRAM мемориите, ќе го споменеме SRAM 4016. Тој има структура 2К×8, што значи има 11 адресни и 8 податочи пина. SRAM може да има максимален капацитет од 128Кx8. За разлика од SRAM, DRAM е со многу поголеми капацитети од 64Mx1 па нагоре. Недостаток на DRAM мемориите е што бараат многу повеќе адресни пина отколку што предвиделе производителите. Поради тоа, адресните пинови се мултиплексираат. Тоа значи дека низ една иста адресна линија можат да се пренесуваат два адресни бита, на пример, A_0 и A_8 или A_3 и A_{11} ,но не истовремено, туку прво едниот, а потоа другиот.



Слика 1.45. Пин дијаграм на DRAM мемориски чип

На сликата 1.45. е прикажан пин-дијаграмот на чипот DRAM 41256, кој има структура 64Кx1. Од сликата може да се забележи дека овој чип има 8 адресни пина, а за адресирање на 64К мемориски локации потребни се 16 адресни пина. Ова е решено преку воведување на два дополнителни пина: $\overline{CAS}$(column address strobe) $\overline{RAS}$ (row address strobe). Додека е активен пинот $\overline{RAS}$, тогаш на адресните пинови доаѓаат битовите од A_0 до A_7. Потоа специјалниот пин $\overline{RAS}$ се крева на високо ниво и на неговото место се активира пинот $\overline{CAS}$, кој дозволува на истите адресни пина да дојдат битовите од A_8 до A_{15}.

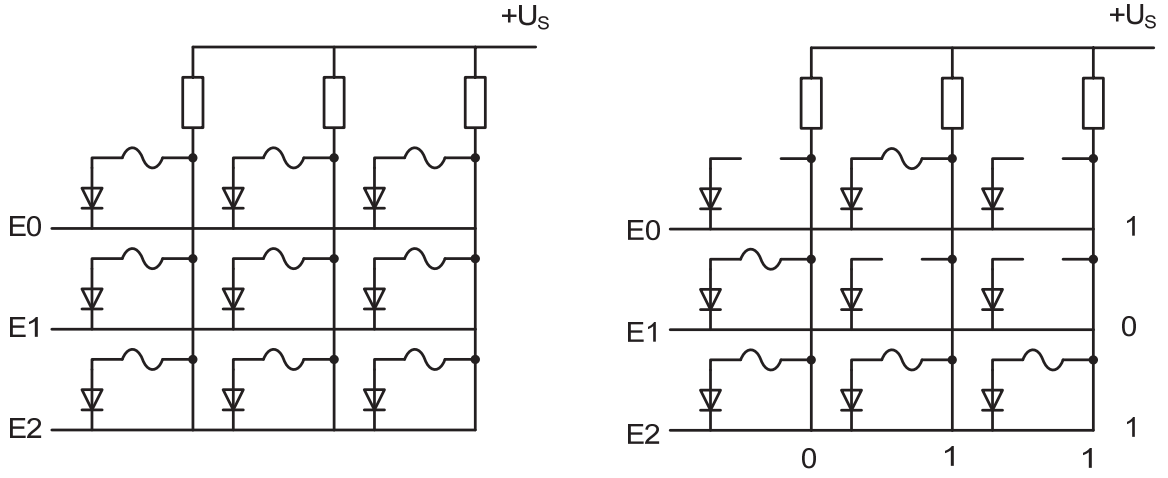
1.7.3. ROM мемории

ROM (Read Only Memory) е меморија за трајно чување податоци. Од неа можат само да се читаат податоци, но не и да се впишуваат нови. По исклучувањето на напојувањето, податоците и натаму остануваат во ROM меморијата. ROM чиповите обезбедуваат поголем степен на сигурност бидејќи меморираните содржини се заштитени од несакани промени. Во ROM меморијата се сместени програми што служат за подигнување на оперативниот систем по вклучувањето на компјутерот. Овие програми се познати под името BIOS (Basic Input Output System).

Постојат различни видови на постапки за внесување на податоци во ROM мемориските чипови. Според тоа постојат различни видови на ROM мемории: маскирачки ROM мемории, PROM, EPROM, EEPROM и флеш мемории.

Маскирачката ROM е меморија чија содржина се впишува во текот на самото производството. За производство на ROM чипови се користат слични технолошки постапки како и за производство на микропроцесорски чипови. Маскрачките ROM чиповите се користат за чување на програми кои масовно се користат и кои не се менуваат.

PROM е програмибилна ROM меморија, која може да ја програмира самиот корисник во зависност од неговите потреби. Структурата на една ваква меморија е прикажана на слика 1.46. Кај диодната PROM меморија во секоја ќелија од мемориската матрица се наоѓа диода, која е сериски поврзана со топлив никел-хром осигурувач.



Слика 1.46. Постапка за впишување на информација во PROM чип

Кога меморијата не е програмирана сите диоди се поврзани со редиците и колоните од матрицата. Корисникот сам ја програмира меморијата така што

предизвикува прегорување на осигурувачите на местата на кои сака состојба на логичка нула. Постапката се врши постапно така што се адресира редица по редица, а на податочните линии кај кои треба да се отстрани диодата се носи негативен импулс. Тогаш низ диодата и осигурувачот ќе протече преголема струја и доаѓа до раскинување на врската меѓу колоната и редицата. PROM меморијата не може повторно да се репрограмира.

EPROM мемориите (Erasable Programmabile Read Only Memory) како мемориски елементи користи MOS транзистори со изолиран гејт. Кога меморијата не е програмирана напонот од +5V на податочната линија е доволен за да се формира канал во MOS транзисторот, па содржината на сите мемориски ќелии е нула. Програмирањето на меморијата се врши така што на податочната линија се носи висок напон (25V) при што состојбата на изолираниот гејт се менува од логичка нула во логичка единица. Вака програмираната меморија не ја менува својата содржина повеќе од десет години. Ако вака програмираниот чип се озрачи со ултравиолетова светлина за време од 20 минути, содржината се губи бидејќи се намалува проводноста на каналот и електроните го напуштаат изолираниот гејт. После ова чипот може повторно да се програмира. Најчесто користена EPROM меморија е чипот со индентификационен број 2716. Овој чип е со структура 2Кx8, што значи содржи 11 адресни пина и 8 податочни пина. Од серијата 27x__ ќе ги споменеме чиповите со броеви: 2704 (512x8), 2708 (1Кx8), 2716 (2Кx8), 2732 (4Кx8), 2764 (8Кx8), 27128 (16Кx8), 27256 (32Кx8), 27512 (64Кx8) и 271024 (128Кx8). Може да се заклучи дека сите овие чипови имаат 8 податочни пина, а бројот на адресни пинови е различен. Ако се помножи бројот на адресни пинови и бројот на податочни пинови се добива број што треба да се додаде на 27 од оваа серија.

EEPROM (Electrically Erasable Programmabile Read Only Memory) исто така како мемориски ќелии користи MOS транзистори со изолиран гејт, само што изолацијата меѓу електродите е помала во однос на MOS транзисторите кај EPRPM мемориите. Програмирањето се врши на сличен начин како и кај EPROM меморијата, но поради помалата изолација се користи помал напон на пробив од +10V. Бришењето на содржината е по електричен пат, така што се користи напон со спротивен поларитет од напонот на впишување. Бришењето се извршува софтверски и може веднаш да се внесат новите содржини. Ова го доведува во прашање називот Read Only Memory, но треба да се земе во предвид дека содржината на EEPROM меморијата се менува многу ретко, на пример еднаш годишно, а содржината на RAM меморијата се менува неколку пати во една секунда.

Најнов облик на полупроводничка меморија е **флеш меморијата**, која така се вика поради брзината со која може да се репрограмира (англиски flash-молња). Флеш меморијата се појавила во средината на осумдесетите години и претставува средина меѓу EEPROM и EPROM меморијата и по својата цена, но и по функционалноста. Исто како и EEPROM, флеш меморијата користи

технологија на електрично бришење. Бришењето на целата флеш меморија трае само неколку секунди. Густината на пакување е иста како кај EPROM (поголема од EEPROM) бидејќи користи само еден транзистор за меморирање на еден бит. Флеш мемориите можат да се репрограмираат во самиот систем, без вадење на чипот од подножјето. При почесто вадење и повторно ставање на чипот во подножјето може да дојде до оштетување на неговите пинови. Флеш мемориите можат да се репрограмираат и до 100000 пати.

Заклучоци:

Претварањето на број од декаден во бинарен броен систем се врши преку делење на бројот со два и запишување на добиениот остаток. Последно добиениот остаток е бит со најголема тежина. Претварањето на број од бинарен во декаден броен систем се врши со собирање на сите цифри од бинарниот број, но претходно помножени со нивните тежини 1, 2, 4, 8, 16, 32, 64, 128 итн,

---

Било која цифра од хексадецималниот броен систем може да се претстави како збир од тежините 8, 4, 2 и 1. На позициите од тежините кои влегуваат во збирот се пишува единица, а на позициите од тежините кои не влегуваат во збирот се пишува нула.

За да на излез од логичкото И коло добиеме логичка единица треба сите влезови да бидат логичка единица. За да на излез од логичкото ИЛИ коло добиеме логичка нула треба сите влезови да бидат логички нули. Ако на влез од инверторот има логичка единица тогаш на излез ќе се добие логичка нула и обратно. За да на излез од логичкото ЕКСИЛИ коло добиеме логичка единица треба на влезовите да има непарен број на единици.

Кај интегрираните кола од TTL фамилијата за логичка нула максималната вредност на влезниот напон изнесува 0,8V, а на излезниот напон 0,45V. За логичка единица минималната вредност на влезниот напон изнесува 2V, а на излезниот напон 2,4V.

Мултиплексерот е коло со 2^n податочни влеза и n контроли влеза со кои се селектира еден од податочните влезови. Селектираниот податочен влез се поврзува со единствениот податочен излез.

Ако на влез од декодерот доаѓа n битен број тогаш со него се селектира еден излез од вкупно 2^n излези.

Флип флопот е најмал мемориски модул во кој може да се запамти информација од еден бит. Работата на флип флоповите може да се анализира табеларно, аналитички и графички.

Кај флип флоповите дигитскиот такт претставува активатор, услов да настане промена на податочните излези под дејство на податочните влезови. Активатор може да биде логичката единица на дигитскиот такт, неговиот растечки или опаѓачки раб.

Основна поделба на регистрите е на стационарни и поместувачки. Постојат четири видови поместувачки регистри: сериски влез-паралелен излез, сериски влез-сериски излез, паралелен влез-сериски излез и паралелен влез-паралелен излез. Најважни контролни сигнали кај стационарниот регистар се сигналот за запишување нови податоци (READ) и сигналот за читање на веќе запишаните податоци (WRITE).

Ако излезот од поместувачкиот регистар се поврзе со неговиот излез (повратна врска) тогаш се добива кружен бројач.

Кај синхрониот бројач сите негови флип флопови користат ист дигитски такт. Кај асинхроните бројачи податочниот излез од претходниот флип флоп се носи на влезот за такт од следниот флип флоп.

Асинхроните бројачи се најчесто изработени од T или JK флип флопови. При тоа податочниот влез на T флип флопот се поврзува со ниво на логичка единица, а двата податочни влезови на JK флип флопот кратко се поврзуваат.

Мемориските чипови се организирани во форма на матрица, мрежа од редици и колони, при што во секоја спојна точка на редиците и колоните има по една мемориска ќелија.

Најважни пинови на мемориските чипови се адресните и податочните пинови. Адресните пинови служат избор на една мемориска локација, а преку податочните пинови се запишуваат нови или читаат веќе внесените податоци.

Постојат два вида на RAM мемории: DRAM и SRAM. DRAM мемориите се изработени од флип флопови и тие се брзи мемории. SRAM меморијата е изработена од кондензатори и нејзе и е потрбно освежување, надополнување на изгубениот електрицитет.

Маскирачката ROM е меморија чија содржина се впишува во текот на самото производство. PROM е програмибилна меморија, која може да ја програмира самиот корисник во зависност од неговите потреби. EEPROM е

меморија чија содржина може да се брише и повторно запишува по електричен пат. Флеш мемориите можат да се репрограмираат до 10000 пати.

Прашања и задачи

1. А) $E3_{(16)}=?_{(2)}$
   Б) $A8_{(16)}=?_{(2)}$
   В) $2B_{(16)}=?_{(2)}$
   Г) $C5_{(16)}=?_{(2)}$

2. А) $49_{(10)}=?_{(2)}$
   Б) $60_{(10)}=?_{(2)}$
   В) $47_{(10)}=?_{(2)}$
   Г) $56_{(10)}=?_{(2)}$

3. А) $00111001_{(2)}=?_{(16)}=?_{(10)}$
   Б) $11100010_{(2)}=?_{(16)}=?_{(10)}$
   В) $01000110_{(2)}=?_{(16)}=?_{(10)}$
   Г) $10100101_{(2)}=?_{(16)}=?_{(10)}$

4. А) $D589_{(16)} + 55CC_{(16)}=?_{(16)}$
   Б) $E5B7_{(16)} + AA33_{(16)}=?_{(16)}$
   В) $12FD_{(16)} + C4C9_{(16)}=?_{(16)}$
   Г) $78D7_{(16)} + 192B_{(16)}=?_{(16)}$

5. Бинарните броеви собери ги логички и аритметички!

   А) $11001110_{(2)} + 00111011_{(2)}=?_{(2)}$
   Б) $11100111_{(2)} + 10010111_{(2)}=?_{(2)}$
   В) $10010011_{(2)} + 11000111_{(2)}=?_{(2)}$
   Г) $01011101_{(2)} + 10110111_{(2)}=?_{(2)}$

6. Нацртај логички дијаграм на декодер со два влеза. Напиши ја табелата на вистинитост.

7. Кои се најпознатите фамилии интегрирани кола и кои се нивните основни карактеристики?

8. Кога се користи постапката на баферирање?

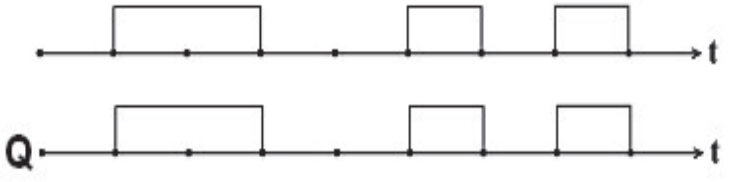
9. Наброј ги видовите на комбинациски кола!

10. Објасни ја намената на декодерот со приоритет на влезовите?

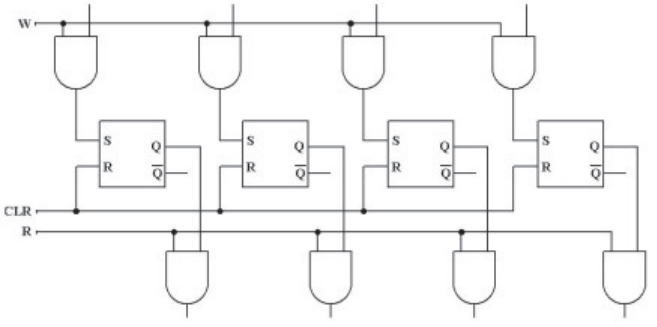
11. Која е разликата меѓу аритметичкото и логичкото собирање?

12. Објасни го називот секвенцијални кола!

13. На сликата е прикажана реакцијата на излезот на еден флип-флоп. За кој флип флоп се работи?



14. Како треба да се побудени контролнителинии R, W и CLR за да може во регистерот да се внесува нова содржина? Колку пати може да се употреби запамтената содржина?

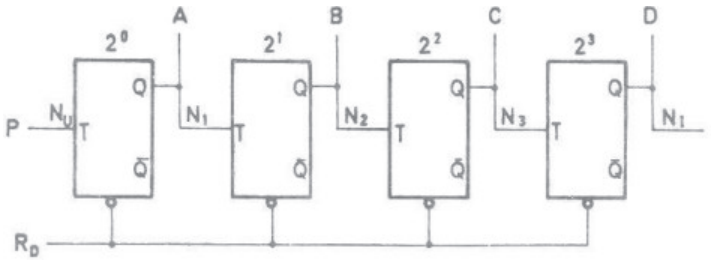


15. Наброј барем 4 пина на мемориски чип со структура 4 редици x 3 колони. Напиши го целиот назив на овие пинови (на англиски јазик) и нивните кратенки.

16. Познат е капацитетот на меморискиот чип. Колку адресни влеза ќе има тој?

А) 256KB
Б) 1GB
В) 32KB
Г) 2MB

17. Колку импулси му се потребни на прикажаниот бројач, почнувајќи од почетната состојба на бројачот, за да ја постигне највисоката вредност во опсегот на броење? Што ќе се појави на излезите од бројачот после десеттиот влезен импулс?



18. Позната е структурата на меморискиот чип. Колку битови може да меморира чипот? Колку бајта може да меморира тој?

    А) 16 редици x 2 колони
    Б) 4 редици x 6 колони
    В) 8 редици x 8 колони
    Г) 16 редици x 4 колони

19. Каква функција има декодерот кај меморискиот чип со структура 4 редици x 3 колони?

20. Кои се составните делови на еднобитната аритметичко-логичка единица?

21. За што служи декодерот кај еднобитната аритметичко-логичка единица?

22. Нацртај мултиплексер со 4 кориснички влеза. Напиши ја табелата на вистинитост за истиот.

23. Изврши споредба меѓу астабилниот и моностабилниот мултивибратор?

24. Кои се најважните пинови на еден мемориски чип?

25. Мемориските чипови се организирани во форма на матрици. Објасни!

26. Спореди ги SRAM и DRAM мемориските чипови!

27. Објасни ја постапката на програмирање на PROM чиповите!

28. Познати се вредностите на влезните сигнали на еднобитната аритметичко логичка единица. Да се одредат вредностите на излезните сигнали и видот на извршената операција?

Fo	F_1	Вид операцијата	A	B	$П_{вл}$	Излез	$П_{из}$
1	1		1	1	1		
1	0		0	0	1		
1	0		1	0	0		
1	0		1	0	0		
1	1		1	0	0		

29. Објасни ја функцијата на адресниот декодер во составот на мемориските чипови!

2. Основи на микрокомпјутерите

2.1. Вовед во микрокопјутерски системи

Под микрокомпјутерски систем подразбираме систем составен од централна процесорска единица, работна RAM меморија и трајна меморија за чување на програмите и добиените резултати од процесирањето. Овие компоненти се неопходни и без нив не може да се врши обработка на податоци. Кога би се обиделе да ги разделиме поимите **компјутерски систем** и **микрокомпјутерски систем** тогаш би рекле дека компјутерските системи се добиени од микрокомјутерските системи преку приклучување на нови хардверски и софтверски компоненти на истите. На пример со додавање на монитор како излезен уред и тастатура како влезен уред, микрокомпјутерскиот систем прераснува во персонален компјутер. Друг пример за микрокомпјутерски систем претставуваат микроконтролерите. Тоа се специјално дизајнирани интегрирани кола кои во себе содржат централна процесорска единица, RAM и ROM . Сите три компоненти се ставени во еден чип. Но како што не можеме да ги користиме нашите персонални компјутери без тастатура, глушец и монитори, така и микроконтролерите не можат да функционираат сами за себе, без да приклучиме сензори на нивните влезови и извршни уреди на нивните излези.

Префиксот микро многу често се користи во компјутерската техника. Такви се зборовите микрокомпјутер, микропроцесор, микроконтролер. Со префиксот микро се истакнувааат нивните мали димензии во однос на нивната огромна функционалност.

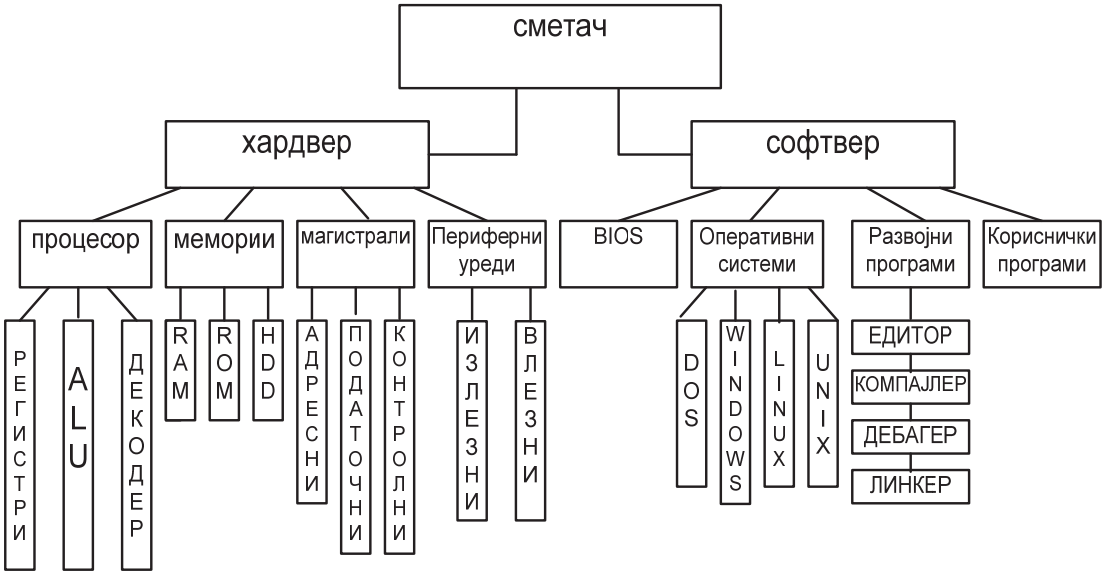
Денес компјутерските системи се користат насекаде и без нив не може да се замисли модерниот живот. Не постои електричен уред што не содржи мини-компјутер во себе. Тука се мисли на телефоните, телевизорите, камерите, микробрановите печки, ЦД-плеерите, играчките и многу други уреди за домашна употреба. Малку посложени компјутери се уредите за видеоигри. Кај нив графиката е понапредна, но софтрверски се многу ограничени. Обично, содржат еден микропроцесор, неколку MB меморија, дисплеј (ако не постои можност за поврзување со телевизор). Најголем дел од луѓето кога ќе го

слушнат поимот компјутер, мислат на персонален компјутер (PC-Personal Computer). За разлика од претходните компјутери, персоналните компјутери имаат многу поголем капацитет на меморија, хард-дискови и можност за приклучување на голем број периферни уреди. Тука за првпат се среќаваме со поимот оперативен систем што, всушност, претставува посредник - врска меѓу човекот и компјутерот. По персоналните компјутери доаѓаат таканаречените сервери. Тоа се моќни компјутери со огромни брзини, кои можат да содржат еден или повеќе микропроцесори. Најчесто се користи за управување со телефонскиот сообраќај, интернет-мрежата и други локални или глобални компјутерски мрежи. Поголемите организации, како банките, универзитетските библиотеки и другите позначајни институции располагаат со компјутери - банки на податоци. Тие се со доста големи димензии и најголем дел од времето и енергијата се трошат на организирање на огромниот мемориски простор. Како најмоќни и најскапи се суперкомпјутерите кои располагаат не само со огромен мемориски простор туку и со супер брзи процесори.

Се разбира, ние не сме во можност да ги проучиме сите овие компјутерски системи. Ќе ги проучиме микропроцесорите на наколку генерации персонални компјутери, нивните составни делови, функционалноста, мемориската организација и поврзувањето со влезно-излезните уреди.

2.2. Компјутерска организација

Компјутерот има сложена структура и таа е речиси иста кај најголем број компјутери. Постојат две основни компоненти: **хардвер** и **софтвер.** На сликата 2.1. прикажана е блок-шема на компјутер.



Слика 2.1. Блок шема на основен модел на компјутер

Под хардвер се подразбираат механички делови од кои е направен компјутерот. Составни хардверски компоненти на компјутер се: микропроцесор, мемории, магистрали и периферни уреди. **Микропроцесорот** ги обработува податоците, **мемориите** ги чуваат, а **магистралите** ги пренесуваат. **Периферните уреди** се познати и под името влезно-излезни единици. Периферните уреди ги претвораат корисничките информации во дигитални сигнали, нули и единици, но и обратно. Кориснички информации се оние информации што ги користи човекот, како што се: звук, слика, текст, видео. Секоја од основните хардверски компоненти ќе биде детално објаснета.

Софтверот претставува збир од програми кои овозможуваат користење на хардверот. Хардверот и софтверот се условно поврзани. На почетотокот, хардверот бил многу поскап од софтверот, но денес состојбата е спротивна. Особено важен момент во развојот на софтверот е појавата на **оперативниот систем**. Тој претставува сума од програми кои овозможуваат огромните хардверски ресурси на компјутерот да бидат достапни до човекот и да му обезбедат поудобна работа. Оперативнот систем претставува дел од софтверот, кој е најблиску до хардверот. Со потсредството на оперативниот систем сите останати програми пристапуваат до хардверот. Најмногу користен оперативен систем е Windows каде човекот му задава команди на компјутерот преку отворање прозорчиња. За компјутери врзани во мрежа, потребни се посебни оперативни системи. Овде посебно внимание се посветува на комуникацијата (размена на информации) меѓу компјутерите и заштитата на податоците од недозволено користење. Кај нас доста актуелен оперативен систем за компјутери врзани во мрежа е UNIX.

Под **развоен систем** се подразбира оној дел од софтверот што служи за изработка на апликативни програми. Можеме да кажеме дека развојниот систем е алатка што служи за правење нов софтвер. Заради подобро сфаќање на развојниот систем, ќе објасниме неколку видови програми што спаѓаат во оваа група :

- едитори-програми кои овозможуваат внесување, менување, обликување, памтење и печатење текст на програмата;
- преведувачи-ги преведуваат програмите напишани во програмски јазици од повисок ред во машински јазик.
- дебагери-програми за отстранување грешки. Се поставуваат контролни точки на одделни места во програмата и во текот на програмата се застанува на овие точки и се испитува состојбата на програмата.
- поврзувачи-програми што вршат поврзување на новиот софтвер со стариот што е многу важно за негово извршување.

Апликативните програми се изработуваат со цел влезните податоци, користејќи го компјутерот, да се трансформираат во излезни податоци. Тие може да бидат напишани во машински јазик или во програмски јазик од повисок

ред. Апликативните програми ги изработуваат корисниците сами или ги купуваат како готови продукти (програми за уредување текстови, програми за книговодство, програми за банки, компјутерски игри и др.). Некои апликативни програми служат и за натамошно усовршување на постојниот развоен систем.

2.3. Основна организација на микропроцесор

Микропроцесорот е ,,**мозокот**" на компјутерот. Тој има две основни функции: извршува програми и управува со другите уреди на матичната плоча.

Слика 2.2. Надворешен изглед на микропроцесор

Микропроцесорот ги ,,гледа" програмите како множество од бајти наредени во последователни мемориски локации. Микропроцесорот ги **извршува програмите** бајт по бајт. Откако ќе го обработи тековниот бајт, тој го чита следниот бајт од RAM меморијата, го обработува и оваа постапка се повторува сè додека процесорот не дојде до крајот на програмата.

Управувачката функција го прави микропроцесорот master, господар на матичната плоча. Тој прибира информации од сите уреди во компјутерот, ги обработува, ги процесира и потоа иницира одредени активности, со цел да се обезбеди сигурна работа на целиот компјутер.

Инструкциското множество претставува множество од сите инструкции кои може да ги извршува микропроцесорот. Сите инструкции на микропроцесорите можат да се поделат во четири основни групи: инструкции за пренос, аритметички, логички инструкции и инструкции за контрола на програмата.

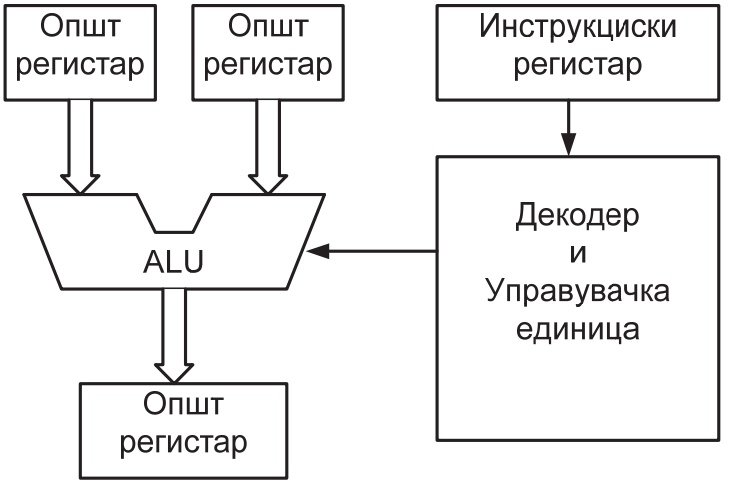
- Инструкциите за пренос служат за: пренос од еден во друг регистар во внатрешноста на микропроцесорот (move), пренос од

меморија во регистар или спротивно (load, store) и читање и пишување од периферните уреди (in, out).

- Аритметички инструкции се инструкциите за собирање (add), одземање (substract), множење (multiply) и делење (divide).
- Во групата на логички инструкции спаѓаат : логичко множење или операција на И коло (and), логичко собирање или операција на ИЛИ коло (or) и пресметка на прв комлемент (complement).
- Контролните инструкции го менуваат текот на програмата. Со нив наместо инструкцијата што следува може да се изврши било која друга инструкција. Во оваа група спаѓаат инструкциите за скок (Jump) и инструкции за повикување на подпрограма (Call). Овие инструкции можат да бидат условни и безусловни. На пример инструкцијата за скок може да се изврши само ако резултатот од претходно извршената инструкција е еднаков на нула.

На слика 2.3. прикажана е основната организација на микропроцесорот. **Составни делови** на еден микропроцесор се: регистри, аритметичко-логичката единица и управувачката единица со декодерот.

Регистрите се брзи мемориски локации во внатрешноста на микропроцесорот. Се делат на општи регистри и регистри со специјална намена. Општите регистри ги содржат податоците што треба да се обработат (пренесат, соберат, поделат, комплементираат, ротираат итн). Секој регистар со специјална намена си има своја точно дефинирана функција. На оваа група припаѓа инструкцискиот регистар. Тој го содржи операцискот код. Овој код служи за претставување на инструкциите во машински јазик. Операцискиот код е единствена комбинација од нули и единици и служи за идентификација и препознавање на инструкцијата.



Слика 2.3. Блок шема на основен модел на микропроцесор

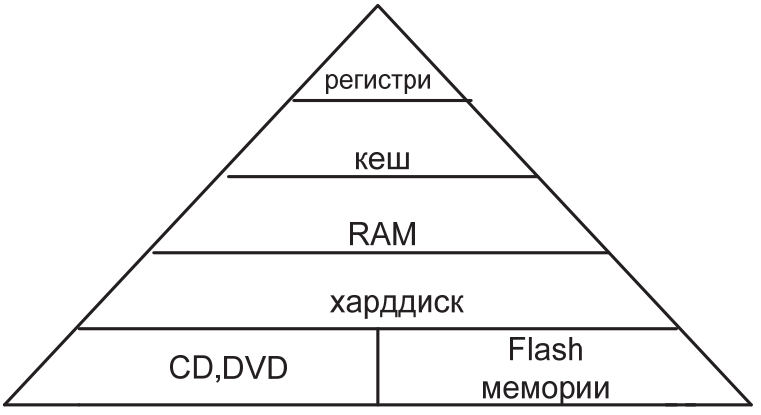
Во аритметичко-логичката единица се извршуваат сите аритметички и логички инструкции. Да се потсетиме, наједноставни аритметички инструкции се: собирањето, одземањето, делењето, множењето, а наједноставни логички инструкции се: И, ИЛИ и негација.

Декодерот го декодира операцискиот код. Тој е директно поврзан со управувачката единица. Управувачката единица е составена од милиони логички кола. Таа прима податоци од сите уреди и потоа одлучува, менува содржина на регистри, менува логичка состојба на пинови, повикува програми, активира периферни уреди итн.

2.4. Мемориска организација

Сите податоци и програми што ги извршува процесорот мора да бидат сместени во меморииte. Најважни **карактеристики на меморииte** се нивниот капацитет и брзината. Капацитетот на меморииte се мери во бајти (B) или поголеми единици за капацитет се: килобајти (KB), мегабајти (MB), гигабајти (GB), терабајти (TB). Брзината на меморијата се мери преку времето на пристап. Тоа е времето потребно меморијата да се пребара и да го пронајде саканиот податок.

Компјутерите содржат различни видови на мемории. Мемориската организација подразбира поделба на функциите за секој вид меморија и начинот на пристап до саканиот податок. На сликата 2.4. е прикажана пирамидата од мемории. Како се симнуваме надолу така се зголемува капацитетот на меморииte, а се намалува нивната брзина на работа. Поголемата брзина на работа (помалото време на пристап на податоците) повлекува зголемување на цената по еден бајт. Во пирамидата, меморииte се подредени хиерархиски, односно според нивната важност. Колку меморииte се поблиску до врвот толку повеќе се поблиску до микропроцесорот.



Слика 2.4. Пирамида на мемории

Регистрите се многу брзи мемориски локации во внатрешноста на самиот микропроцесор. Тие служат за привремено чување на податоците пред тие да бидат обработени, но и за сместување на резултатите веднаш по нивното добивање. Регистрите се најбрзи мемориски локации во компјутерот, но нивниот капацитет изнесува само неколку бајти.

Многу често се случува микропроцесорот да чека (wait) да се донесат потребните податоци од меморијата за да истите ги обработи. Со цел да се намали времето на чекање, се користи **кеш-меморија.** Кеш-меморијата е многу брза меморија. Заради побрзо извршување на тековната програма, потребните податоци од побавните меморији однапред се донесуваат и се сместуваат во кеш-меморијата. Во кеш-меморијата може да се чуваат податоци што почесто ги користи микропроцесорот, а не може да ги смести во своите регистри. Кеш-меморијата може да биде во самиот мемориски чип или надвор од него. Кеш-меморијата е брза RAM меморија, но со помал капацитет.

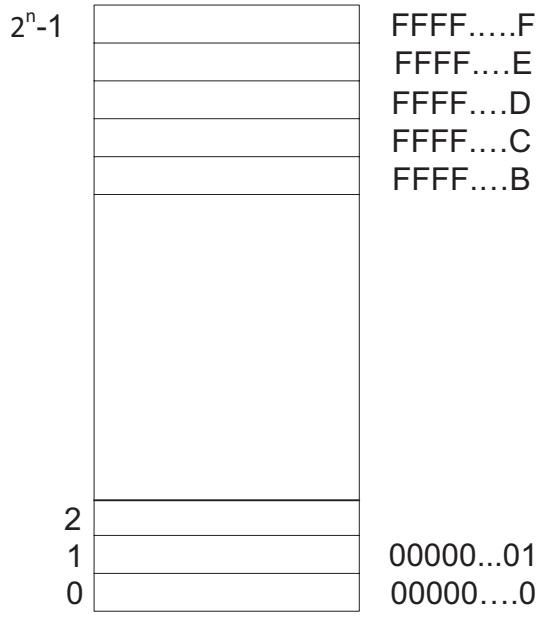
RAM е кратенка од англиските зборови Random Access Memory, што во превод значи **меморија со случаен пристап**. Имено, процесорот има ист пристап до сите локации од меморијата или ниедна локација не е со помал или со поголем приоритет. RAM е меморија од која може истовремено да се читаат, но и да се пишуваат нови податоци. Тоа е меморија со променлива содржина и претстатува **работна меморија**, еден вид работен лист на кој микропроцесорот привремено ги запишува сите податоци што ќе му бидат потребни за успешно реализирање на дадената програма. Програмите се најчесто снимени на некој од уредите за перманентна меморија (диск, дискета), но во моментот кога се работи со нив, тие мора да бидат ископирани во RAM меморијата. RAM меморијата е позната и под името примарна меморија, поради нејзината важност за микропроцесорот. RAM е **привремена меморија**. Доколку напојувањето се исклучи, податоците од RAM не можат да се обноват по повторното вклучување. Поради тоа, пред да го исклучиме компјутерот, податоците од RAM треба да се сместат на хард-дискот преку кликнување на иконата SAVE. Ова е прикажано на слика 2.5.

Слика 2.5. Функција на RAM-от како работна меморија на микропроцесорот

Микропроцесорот е многу побрз уред од мемориите. Пребарувањето на податоците понекогаш може да биде сложена и бавна постапка. Врз брзината на работата посебно влијание имаат RAM меморијата и кеш-меморијата. Микропроцесорот обработува програми бајт по бајт. Притоа, инструкциите ги зема од RAM меморијата. RAM меморијата го ,,храни" процесорот со

информации, а RAM-от се храни од хард-дискот. Сите програми се сместени на хард-дискот и тие се пренесуваат во RAM кога треба да бидат обработени. Доколку некоја програма е преголема и не може да ја собере во предвидениот простор во RAM-от, тогаш таа се дели на повеќе делови и така се обработува дел по дел. Доколку RAM е поголем, тогаш тој одеднаш ќе земе поголем дел од корисничката програма (сместена на хард-дискот) и истиот ќе се обработи во микропроцесорот. Со ова се намалува бројот на преноси од RAM до хард-дискот, а тоа значи и заштеда на време. Проширувањето на RAM меморијата со нови локации бара поголем број адресни линии во магистралите за да се овозможи адресирање на нови локации.

На слика 2.6. преставена е **мемориската мапа на RAM** меморијата. Секоја редица претставува една мемориска локација. Вкупниот број на мемориски локации зависи од бројот на адресни линии и се пресметува според равенката: вкупен број на мемориски локации = 2^n, каде што n е број на адресни линии. Бидејќи броењето започнува од нула последната локација ќе биде со реден број 2^n-1. Адресата на првата, најдолната локација од RAM меморијата е еднаква на n нули, а последната, најгорната локација има адреса составена од n единици. На слика 2.6. адресите на мемориските локации не се дадени во бинарен, туку во хексадецимален броен систем.



Слика 2.6. Мемориска мапа на RAM меморија

На пример ако бројот на адресни битови изнесува 10 тогаш капацитетот на таа меморија ќе биде 1К, локациите ќе бидат нумерирани од 0 до 1023, почетната адреса ќе биде 0000000000B=000H, а крајната 1111111111B=3FFH.

Ваквата мемориска мапа овозможува полесно разбирање на работата на микропроцесорот и извршувањето на програмите. Микропроцесорот

програмите ги извршува бајт по бајт. Откако ќе заврши со тековниот бајт, се повикува RAM меморијата да го пронајде следниот. Најчесто бајтите од програмата се наредени еден врз друг, во последователни мемориски локации (редиции) од RAM меморијата.

RAM меморијата е примарна меморија, а хард-дискот и другите надворешни мемории се секундарни. За разлика од RAM-от, кеш меморијата и регистрите, **секундарната меморија** претставува трајна меморија. Бидејќи секундарната меморија е со многу голем капацитет многу е важно пребарувањето на истата. Пребарувањето треба да биде брзо, но едноставно. Ќе го споменеме механизмот на виртуелни страници. Секундарната меморија се дели на страници, а примарната на рамки. Страниците и рамките се со ист капацитет. Во зависност од потребите се врши пренос на страниците во рамките, при што мора да се води сметка која страница во која рамка се запишува. Со виртуелната меморија ќе се запознаеме детално подоцна.

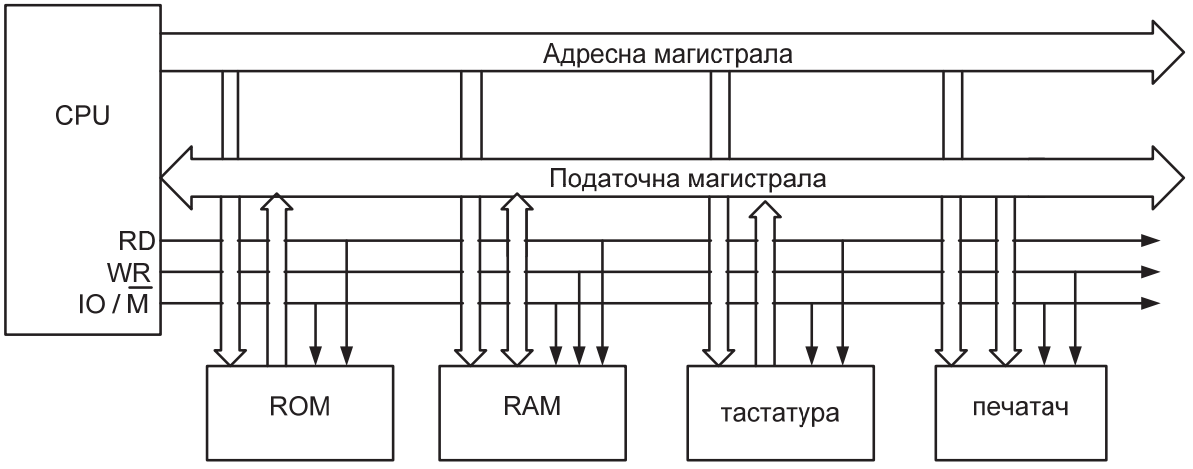
2.5. Работа на компјутер

Магистралите се бакарни водови на матичната плоча и служат за пренос на податоци. Првите персонални компјутери имале една единствена магистрала наречена системска магистрала. Таа се состоела од 50 до 100 бакарни жици вградени во матична плоча, со конектори поставени на еднакви растојанија за приклучување на мемориските чипови и на влезно-излезните уреди. Модерните компјутери содржат повеќе магистрали со специјална намена, на пример, една магистрала за поврзување на процесорот со меморијата и друга магистрала за поврзување на процесорот со влезно-излезните уреди.

На слика 2.7. е прикажано како магистралите ги поврзуваат различните делови од компјутерот, како што се микропроцесорот, RAM, ROM и неколку периферни уреди. За да се одвива преносот на податоци без грешки, се користат специјално дефинирани правила, до кои мора да се придржуваат сите уреди во компјутеркиот систем. Ова множество од правила се нарекува магистрален протокол (bus protocol).

Некој од уредите што се приклучени на магистралата можат да иницираат трансфер на податоци и овие уреди се викаат господари – masters. Другите уреди се пасивни и можат да примаат или да даваат податоци по туѓо барање и овие уреди се викаат робови – slaves. На пример, кога централниот процесор ќе му нареди на диск-контролерот да прочита блок на податоци, тогаш процесорот е господар, а контролерот е роб.

Основни карактеристики на една магистрала се работната фреквенција и широчината. Под широчина на магистрала подразбираме од колку бакарни водови е составена, т.е. колку битови може да се пренесат истовремено. Работната фреквенција е од редна величина на MHz и ни покажува колку битови се пренесуваат по еден бакарен вод во една секунда. Ако се помножат работната фреквенција и широчината на магистралата, се добива **пропусниот опсег на магистралата**. Единица мерка за пропусен опсег е број на бити / секунда.



Слика 2.7. Начин на поврзување на составните делови на компјутерот

Според видот на сигналите што се пренесуваат, магистралите можат да се групираат во 3 основни групи: адресни линии (ADDRESS), податочни (DATA) и контролни (CONTROL).

Адресната магистрала ги пренесува адресните битови од микропроцесорот до мемориите и периферните уреди. Адресата е единствена комбинација од нули и единици и служи за препознавање, идентификација на само една мемориска локација или периферен уред. Кога меморијата ќе ја добие адресата од микропроцесорот, таа се пребарува за да ја пронајде саканата мемориска локација.

Податочните линии служат за пренос на корисничка информација од еден до друг уред во компјутерот. Податочната магистрала е двонасочна. Кога се чита податок од меморијата, податочната магистрала пренесува податоци од меморијата кон микропроцесорот. Кога се впишува податок во меморијата, преносот на податоци се врши во насока од микропроцесорот кон меморијата. Магистралите можат да бидат 8, 16, 32 и 64-битни. Поголема широчина значи и поголема брзина на работа. Така, на пример, ако треба да се пренесе 32-битен податок по 8-битна магистрала, тогаш се потребни 4 циклуси за пренос на податоци.

Контролните или управувачките линии се најмногубројни и за секоја управувачка линија постои специјална ознака, што, всушност, е скратеница од англискиот збор за дадениот контролен сигнал. Контролните линии служат за дефинирање на операцијата што треба да се изврши. Некои контролни битови се пренесуваат од микропроцесорот кон други уреди и претставуваат еден вид наредби. Други контролни битови се пренесуваат од други уреди кон микропроцесорот и претставуваат еден вид повратна информација за тоа како се извршени претходно зададените наредби. Дали ќе се пишува или ќе се чита податок од меморијата, одлучуваат две контролни линии од контролната магистрала: $\overline{RD}$ и $\overline{WR}$ **.** Тие се кратенки од зборовите READ (читај) и WRITE (пишувај). Негацијата врз нив значи дека се активни на логичка нула. Не смее двете линии да бидат активни истовремено (или ќе читаме или ќе пишуваме). Друга поважна контролна линија е линијата со ознака $IO/\overline{M}$ **.** Оваа ознака е кратенка од Input Output/$\overline{\text{Memory}}$ што во превод значи влезно излезни уреди/меморија. Кога овој сигнал $IO/\overline{M}$ =1, тогаш микропроцесорот комуницира со некоја влезно-излезна единица (периферен уред), а кога $IO/\overline{M}$ = 0, тогаш микропроцесорот комуницира со некој мемориски модул.

Пример 2.1: Познати се контролните сигнали $IO/\overline{M}$ =0, $\overline{RD}$ =0 и $\overline{WR}$ =1. Со кој уред комуницира микропроцесорот и каква операција извршува?

Решение: Микропроцесорот чита ($\overline{RD}$ =0) податок од некој мемориски модул ($IO/\overline{M}$ =0)

Магистралите се вградени во самата матична плоча на компјутерот и нивниот број е неменлив. Зголемувањето на бројот на линии по магистрала предизвикува големо зголемување на цената на компјутерот. Мора да се прави компромис меѓу цената и широчината на магистралите. На пример, колку повеќе адресни линии имаме на располагање толку поголем мемориски простор може директно да се адресира. Ако магистралата има n адресни линии, тогаш може да се адресираат вкупно 2^n мемориски локации. Со секоја нова генерација на компјутери бројот на линии по магистрала расте, како адресните така и податочните и контролните. Доста користена постапка е **мултиплексирањето.** Мултиплексирањето значи пренос на податочни и адресни битови низ една иста магистрала, но не истовремено, туку во различни временски интервали (прво едните, а потоа другите видови сигнали). Постои мултиплекс на адресна и податочна магистрала. По податочната магистрала се пренесуваат податочни и адресни битови. Кој од овие битови ќе се пренесува, одлучува контролниот сиганал ALE (Address Letch Enable). Кога овој сигнал е на високо ниво, тогаш низ магистралата се пренесуваат адресни битови, а кога сигналот ALE е на ниско ниво, тоа значи пренос на податочни битови. Кога треба да се прочита некој податок од меморијата, секогаш прво се пренесува адресата, а потоа откако ќе се пребара меморијата, податокот се пренесува по

истата магистрала. Значи, доколку податочната магистрала не се мултиплексира во периодот додека се пренесува адресата таа би била неискористена.

Заклучоци:

Составни хардверски компоненти на компјутер се: микропроцесор, мемории, магистрали и периферни уреди. Микропроцесорот ги обработува податоците, мемориите ги чуваат, а магистралите ги пренесуваат. Периферните уреди се познати и под името влезно-излезни единици. Периферните уреди ги претвораат корисничките информации во дигитални сигнали, нули и единици, но и обратно.

Микропроцесорот е ,,мозокот" на компјутерот. Тој има две основни функции: извршува програми и управува со другите уреди на матичната плоча.

Составни делови на еден микропроцесор се: регистри, аритметичко-логичката единица и управувачката единица со декодерот.

Управувачката единица прима податоци од сите уреди и потоа одлучува, менува содржина на регистри, менува логичка состојба на пинови, повикува програми, активира периферни уреди итн.

Најважни карактеристики на мемориите се нивниот капацитет и брзината. Капацитетот на мемориите се мери во бајти (B) или поголеми единици за капацитет се: килобајти (KB), мегабајти (MB), гигабајти (GB), терабајти (TB). Брзината на меморијата се мери преку времето на пристап.

RAM е кратенка од англиските зборови Random Access Memory, што во превод значи меморија со случаен пристап. Имено, процесорот има ист пристап до сите локации од меморијата или ниедна локација не е со помал или со поголем приоритет.

RAM е меморија од која може истовремено да се читаат, но и да се пишуваат нови податоци. Тоа е меморија со променлива содржина и претстатува работна меморија, еден вид работен лист на кој микропроцесорот привремено ги запишува сите податоци што ќе му бидат потребни за успешно реализирање на дадената програма.

RAM е привремена меморија. Доколку напојувањето се исклучи, податоците од RAM не можат да се обноват по повторното вклучување.

Вкупниот број на мемориски локации зависи од бројот на адресни линии и се пресметува според равенката: вкупен број на мемориски локации = 2^n, каде што n е број на адресни линии. Адресата на првата, најдолната локација од RAM меморијата е еднаква на n нули, а последната, најгорната локација има адреса составена од n единици.

Основни карактеристики на една магистрала се работната фреквенција и широчината. Под широчина на магистрала подразбираме од колку бакарни водови е составена, т.е. колку битови може да се пренесат истовремено. Работната фреквенција ни покажува колку битови се пренесуваат по еден бакарен вод во една секунда. Ако се помножат работната фреквенција и широчината на магистралата, се добива пропусниот опсег на магистралата.

Прашања и задачи

1. Накратко објасни ја функцијата на микропроцесорот, мемориите, магистралите и периферните уреди!
2. Објасни го значењето на оперативниот систем за работата на микрокомпјутерите!
3. Од кои делови е составена една развојна програма? Објасни ја функцијата на секој од тие делови!
4. Која е функцијата на микриороцесорот во компјутерот?
5. Наброј ги најважните делови на еден микропроцесор и објасни ја нивната функција!
6. Што подразбираме под поимот инструкциско множество на микропроцесор?
7. Кои се основни карактеристики на меморискиот модул?
8. RAM е привремена работна меморија со случаен пристап. Објасни!
9. Кои се предностите и недостатоците на меморијата DRAM во однос на меморијата SRAM?

10. Зошто големината на RAM влијае врз брзината на работа на микропроцесорот?

11. Нацртај ја мемориската мапа на RAM меморијата и објасни го начинот на адресирање!

12. Како бројот на адресни влеза влијае врз капацитетот на еден мемориски чип?

13. Објасни ја намената на податочните пинови на мемориски чип! Кога податочните пинови на меморијата се влезни, а кога излезни?

14. Набroj неколку пинови за селекција на мемориски чип!

15. Кои контролни сигнали се користат за контрола на мултиплексот на адресната магистрала и мултиплексот на податочната магистрала?

16. Кои сигнали се пренесуваат по адресната, податочната и контролната магистрала?

17. Кои се основните карактеристики на една магистрала?

18. Пресметај го пропусниот опсег на шестнаесет битна магистрала со работна фреквенција од 512 MHz?

3. Општа архитектура на микропроцесор

3.1. Историски развој на микропроцесорите

Брзиот развој на компјутерите е условен од брзиот развој на технологијата на интегрирани кола. Микроелектрониката е гранка од електрониката што се занимава со дизајнирање, проектирање и производство на интегрирани кола -чипови. Чипот претставува силициумска плочка во која преку постапката дифузија се внесуваат атоми од бор и арсен и на таков начин се добиваат p и n слоеви. Дебелината на овие слоеви и концентрацијата на атоми во нив мора да бидат строго пресметани доколку сакаме да добиеме чип со саканите перформанси. Основни карактеристики на едно интегрирано коло се: сигурноста на интегрираното коло, степенот на интеграција по единица површина, брзина на работа, начинот на поврзување со другите интегрирани кола, односно интерфејсот итн.

Под микропроцесор се подразбира чип, кој ја обединува работата на целиот компјутерски систем. Тој претставува мозок на компјутерот и од неговите перформанси зависи моќноста на самиот компјутер. Дури и името на компјутерот произлегува од името на неговиот микропроцесор.

Историјата на микропроцесорите е кратка, но многу бурна. Уште во 1960 година претседателот на „Интел“, Гордон Мур, претпоставил закон, кој се докажа дека е точен во наредните три децении, а гласи дека моќта на компјутерите и сложеноста на интегрираните кола ќе се удвојува на секои две години, а нивните цени ќе се преполовуваат. На табелата 3.1. е прикажан хронолошкиот развој на микропроцесорите и на нивните перформанси.

Во 1971 година беше произведен првиот покомерцијален микропроцесор 8008, кој всушност, претставуваше 8-битна верзија на 4004. Две години подоцна „Интел“ го презентира модерниот осумбитен микропроцесор 8080. Паралелно со „Интел“ и другите светски познати производители на електронски

компоненти се претставија на пазарот со слични осумбитни микропроцесори, како што микропроцесорот е MC 6899 на производителот „Моторола“ или Z-8 на компанијата „Зилог“. На пример, „Зилог“ произведува микропроцесори ориентирајќи се на микроконтролери.

Процесор	Година на производство	Брзина на MIP	Фреквенција	Број на транзистори	Големина на регистри	Големина на адресен простор	Големина на магистрали	Кеш-меморија
8086	1978	0,8	8MHz	29K	16	1M	16	/
Интел 286	1982	2,7	12.5MHz	134K	16	16MB	16	/
Интел 386	1985	6,0	20MHz	275K	32	4GB	32	/
Интел 486	1989	20	25MHz	1,2M	32	4GB	32	8KB L1
Пентиум	1993	100	60MHz	3,1M	32	4GB	64	16KB L1
Пентиум про	1995	440	200MHz	5,5M	32	64GB	64	16KB L1 256K или 512KB L1
Пентиум II	1997	466	266MHz	7M	32	64GB	64	32KB L1 256/512KB L2
Пентиум III	1999	1000	500MHz	8.2M	32GP 128 SIMD-FP	64GB	64	32KB L1 512KB L2

MIP-Milions of instruction per second

Табела 3.1. Хронолошки развој на микропроцесорите

Процесорот 8080 ги извршува инструкциите десет пати побрзо од 8008, односно 50.000 инструкции во секунда (20μs по инструкција). За разлика од 8008, 8080 е целосно TTL компатабилен, со што интерфејсот станува многу поедноставен и поевтин. 8080 има 64KB меморија што е четири пати поголема во однос на 8008. Исто така, овој микропроцесор е вграден во првиот персонален компјутер, кој излезе на пазарот под името MITS Altair 8800 во 1974 год.

Во 1977 година „Интел“ го претставува **8085**, последниот осумбитен микропроцесор. Денес постојат над 200 милиони вакви микропроцесори вградени во најразлични електронски уреди, а неговото производство ќе продолжи и во иднина. Друга компанија што има продадено 500 милиони осумбитни микропроцесори слични на 8085 е „Зилог“ и тоа под називот Z-80.

Во 1978 година беше произведен микропроцесорот **8086,** а една година подоцна 8088. Двата чипа се 16-битни микропроцесори со брзина на извршување од 2.5MIPs. микропроцесорската меморија се зголемува дури за 16 пати и изнесува 1MB. Новина е малата инструкциска кеш-меморија од 4 или од 6 бајти, која дозволува однапред пренос на неколку инструкции од меморијата до микропроцесорот пред тие да бидат извршени. Инструкциското множество е огромно, составено од 20.000 различни инструкции. Некои од нив

се доста сложени и поради тоа за овие микропроцесори велиме дека се CISC (Complex Instruction Set of Computers) технологија. микропроцесорот 80286 е речиси идентичен со 8086 и со 8088, со таа разлика што кај него можеме да адресираме простор од 16MB.

Новите софтверски апликации бараа побрзи микропроцесори, повеќе меморија и пошироки магистрали. Поради тоа, во 1986 година беше произведен првиот Интелов 32-битен микропроцесор **80386**. Овој микропроцесор содржи 32-битна податочна магистрала и 32-битна адресна магистрала, што дозволува директно адресирање на 4GB. Исто така, квалитетната графика бара брз и моќен микропроцесор. Модерните монитори содржат над 300.000 пиксели. Кај нив графиката е регулирана преку специјален софтвер, како што е GUI (Graphical User Interface) и модерниот VGA (Variable Graphics Array). Новина кај 80386 е специјалната хардверска компонента MMU (Memory Management Unit) за управување со работата на меморијата. Дотогаш овој проблем се решавал софтверски.

Во 1989 година „Интел“ го претставува микропроцесорот **80486**. Овој микропроцесор значи обединување на микропроцесорот 80386, нумеричкиот копроцесор 80387 и 8KB кеш-меморија во едно интегрирано коло. Работната фреквеницијa на овој микропроцесор изнесува 66MHz. Но трансферот сè уште е бавен и се извршува со 33MHz. Да напомене дека верзијата 80486DX4 со работната фреквенција од 100MHz, брзина на трасфер 33MHz и 16KB кеш-меморија има речиси исти перформанси како и микропроцесорот Пентиум со фреквенција 60MHz.

Пентиум-процесорот излезе на пазарот во 1993 година. Во тој период различните верзии на Пентиум-процесорот можеа да обработуваат од 110 до 150MIPs. Други потипични карактеристики се поделените податочна кеш-меморија и инструкциската кеш-меморија, 4GB RAM меморија, брзина на податочна магистрала 60-66MHz. Зголемената брзина на работа дозволува виртуелниот софтвер да стане пореален. Инструкциското множество е проширено со нови инструкции наречени мултимедијално проширување или MMX инструкции. Во однос на претходните генерации компјутери Пентиумот видно се разликува по тоа што содржи два микропроцесора за обработка на цели броеви, со што истовремено и независно можат да се извршуваат две инструкции. Овој начин на работа се нарекува суперскаларна технологија.

Кај **Пентиум Про** микропроцесорот новина се двете кеш-мемории: 16KB L1(level one) меморија составена од 8KB податочна и 8KB инструкциска меморија. Друга значајна промена е појавата на три извршни единици за паралелно извршување на три инструкции. За Пентиум Про микропроцесорот подобар оперативен систем е Windows NT во однос на Windows 95. Да напоменеме дека Windows NT користи 32-битен код, додека Windows 95 16-битен код.

Со појавата на **Пентиум II**, компанијата „Интел“ презентираше сосема нов правец во производство на микропроцесори. Наместо микропроцесорот како интегрирано коло да е директно поврзан на матичната плоча, тој се поставува во специјално дизајнирано пластично куќиште. Исто така, и 4КВ кеш-меморија е вградена во истото интегрирано коло како и микропроцесорот, со што се добива поголема ефикасност во работата, односно поголема брзина. Во 1998 година „Интел“ ја зголеми брзината на пренос на магистралите од 66MHz на 100MHz. На овој начин беше елиминиран ефектот на тесно грло, кој се јавува кога магистралите се многу побавни од микропроцесорот. Во 1998 година „Интел“ ја претставува и новата верзија на Пентиум II, наречена Пентиум II Хеон. Основна разлика на овие два компјутера е зголемената L2 кеш-меморија од 512КВ на 1 или 2 МВ. Исто така, микропроцесорот Хеон е дизајниран за да функционира заедно со 4 вакви микропроцесори во еден ист систем, што има извесни сличности со микропроцесорот Пентиум Про.

Процесорот **Пентиум III** се карактеризира со максимална работна фреквенција 1GHz, 32KB L1 кеш-меморија и 256KB или 512KB L2 кеш-меморија со брзина на магистралите 100 MHz.

Во 2000 година излезе **Пентиум IV,** кој можеше да работи со 1,3 или 1,4 GHz фреквенција. Интересно е што L1 кеш-меморијате се намалува од 32 на 8KB, додека L2 останува со иста големина. Наместо алуминиумски контакти, почнале да се користат бакарни. Бакарот е подобар проводник и требало да ја зголеми фреквенцијата на микропроцесорот. Брзината на магистралите се зголемува од 133 на 200MHz.

3.2. Видови на податоци

Информациите што ги користи човекот се претвораат во дигитален облик пред да бидат пренесени или обработени во некој дигитален систем. Дигитализираните податоци претставуваат бинарни кодови, односно низи од нули и единици. Групата од битови со кои се претставува т.е. кодира некој податок се вика коден збор. Меѓу податоците и кодните зборови постои еднозначна коресподенција односно не може да се случи два податоци да имаат ист коден збор. Податоците кои ги обработуваат микропроцесорите може да се поделат во две основни групи: карактери и броеви.

Под **карактери** се подразбираат бројките , буквите и алфанумеричките знаци кои се користат за пишување на текст. Секое копче од тастатурата претставува два или повеќе карактери. Карактерите се кодираат според однапред дефинирани правила при што секој карактер се претсавува со единствена комбинација од седум нули и единици. Најчесто користен код за

карактери е **ASCII** кодот. Целосниот назив на овој код е American Standard Code for Interchange Information што во превод значи Американски стандарден код за размена на информации. Овој код служи за пренос на сите карактери од еден уред до друг во внатрешноста на компјутерот или за внесување или изнесување податоци преку периферните уреди (на пример, тастатура за влез и принтер за излез). ASCII кодот е 7-битен што значи дака со него можат да се претстават 2^7=128 знаци. На табелата 2.2. се претставени ASCII кодовите.

код	знак	код	знак	код	знак	код	знак	код	знак	код	знак
20	празно	30	0	40	@	50	P	60	`	70	p
21	!	31	1	41	A	51	Q	61	a	71	q
22	“	32	2	42	B	52	R	62	b	72	r
23	#	33	3	43	C	53	S	63	c	73	s
24	$	34	4	44	D	54	T	64	d	74	t
25	%	35	5	45	E	55	U	65	e	75	u
26	&	36	6	46	F	56	V	66	f	76	v
27	‘	37	7	47	G	57	W	67	g	77	w
28	(	38	8	48	H	58	X	68	h	78	x
29	)	39	9	49	I	59	Y	69	i	79	y
2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
2C	,	3C	<	4C	L	5C	/	6C	l	7C	|
2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
2F	/	3F	?	4F	O	5F	_	6F	o	7F	

Табела 3.2. ASCII кодови за алфанумеричките знаци

Кодовите од 0 до 1F (хексадецимално) претставуваат контролни знаци што служат за уредување текст пред тој да биде пренесен.

Пример 3.1: На слика 3.1. ни е преставен ASCII кодот за зборот ,,Mikro”

алфанумерички знаци	M	i	k	r	o
бинарно	01001101	01101001	01101011	01110010	01101111
хексадецимално	4D	69	6B	72	6F

Слика 3.1 Претставување на зборот Mikro со ASCII код

За меморирање на зборот Mikro ќе бидат употребени шест мемориски локаци, односно за секоја буква од зборот по една мемориска локација. Зборовите составени од карактери всушност претставуваат низи од податоци. Подоцна кога детално ќе го проучиме инструкциското множество ќе видиме дека постојат специјални инструкции за низи со кои значително се поедноставува нивната обработка.

За претставување на **броевите** во бинарен облик се користат два вида на кодови: тежински и редоследен. Пример за тежински код би бил веќе опишаниот природен бинарен броен систем. Претварањето од декаден броен систем во бинарен го разгледавме на почетокот од првата тема. Со пресметка на втор комплемент од позитивниот бинарен број се добиваше бинарната вредност на негативниот број. Кога се работи со **броеви со предзнак** најзначајниот бит од бројот претставува бит за знак. Ако овој бит е нула тогаш се работи за позитивен број, а кога тој е еден тогаш се работи за негативен број. Ако работиме со броеви без предзнак тогаш со осум битови можат да се престават 256 броеви, од нула до 255. Ако работиме со броеви со предзнак тогаш со осум битови можат да се престават броевите од -128 до +127. На сликата 3.2. се преставени осумбитните броеви со предзнак и без предзнак. Во редиците од табелите се прикажани хексадецималните вредности на секој од броевите, а од страна дадени се нивните вредности во декаден броен систем.

Слика 3.2. Претставување на броеви со и без предзнак во хексадецимален броен систем

Податоците кои ги обработува микропроцесорот може да се поделат и според нивната **големина**. 8-битните податоци се викаат бајти. 16-битните броеви се познати под името зборови. Со еден збор можат да се престават броевите од нула до +65535 без предзнак и броевите од -32768 до +32767 со предзнак. 32-битните податоци се познати под името двојни зборови и за нивно меморирање ќе бидат потребни четири мемориски локации.

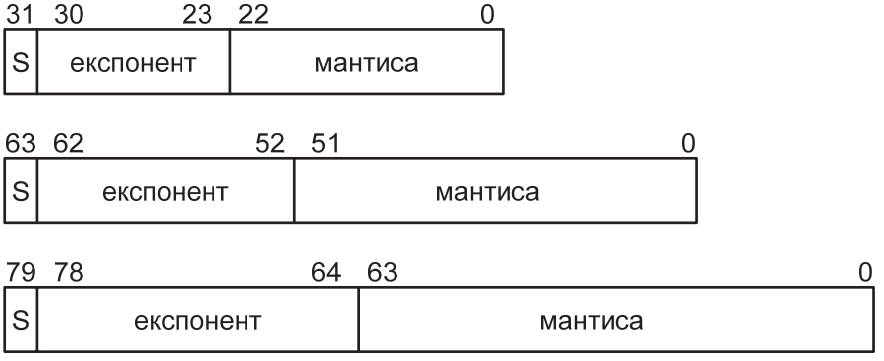
BCD кодот е многу често користен бинарен код. Кај овој код кодните зборови не се добиваат преку делење со два и запишување на остатокот. Кај BCD кодот секоја цифра од декадниот број се запишува со 4 бинарни цифри. Со 4 бита можеме да направиме 2^4=16 комбинации, но од нив користиме само 10 комбинации во претставување на декадните броеви од 0 до 9. Шест

комбинации останууваат неискористени. Оваа постапка речиси е идентична со постапката за премин од хексадецимален во бинарен броен систем, со тоа што таму се користат сите 16 комбинации. Во пример 3.2 е претставен декадниот број 1944 во BCD код и бинарен броен систем, користејќи 16 бита и во двата случаја.

Пример 3.2:
Декаден броен систем: 1944
Бинарен броен систем: 00000111100110000
BCD код: 0001 1001 0100 0100

Со 16 бита во BCD код може да се претстават броевите од 0 до 9999, а во бинарен броен систем може да се претстават броевите од 0 до 65536. Очигледно е дека бинарниот броен систем е поефективен од BCD кодот, но електричните кола за претворање од декаден броен систем во BCD код се многу поедноставни и побрзи од колата за претворање во бинарен.

Реалните броеви се познати под името **броеви со подвижна запирка**. Сите модерни микропроцесори користат специјален нумерички копроцесор и специјални инструкции за обработка на броевите со подвижна запирка. На слика 3.3. прикажани се трите формати на броевите со подвижна запирка во бинарен броен систем. Тие се составени од три дела: бит за знак-S (sign), експонент и мантиса.



Слика 3.3. Формати на броеви со подвижна запирка

Ако бројот е негативен тогаш битот за знак е еден, а ако е позитивен тогаш битот за знак е нула. Постапката за добивање на експонентот и мантисата е следна:

1. Претварање на бројот во бинарен броен систем
2. Нормализација на бинарниот број. Запирката се поместува во лево сè додека пред запирката не остане само најзначајниот бит од бинарниот број. Степенот е еднаков на бројот на поместувања на запирката во лево. Битовите после запирката ја претставуваат мантисата. На мантисата може да се додаваат нули од десната страна за да биде задоволен соодветниот формат.

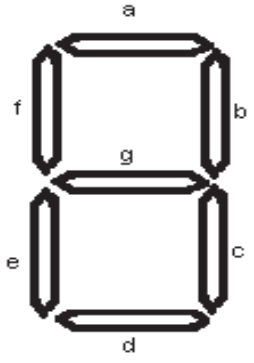
3. Експонентот се добива со додавање на степенот во бинарен облик на основата на форматот. За првиот краток формат основа е бројот 7FH, за вториот формат 7FFH и за третиот формат 7FFFH. Да напоменеме дека колку е поголем форматот толку повеќе се зголемува прецизноста со која се прикажува реалниот број.

Пример 3.3: Реалниот број 100,25 ќе го преставиме како број со подвижна запирка.

1. 100,25=1100100,01
2. $1100100,01=1,10010001\times2^{6}$
3. 110+01111111=10000101

Решение: бит за знак=0, експонент=10000101,
мантиса=10010001000000000000000

За визуелно претставување на броевите од декадниот броен систем се користи **седумсегментен код**. Секое поле од седумсегментниот дисплеј е составен од 7 сегменти, како што е прикажано на слика 3.4.а.



\	x	a	b	c	d	e	f	g	\
0	0	1	1	1	1	1	1	0	= 7EH
1	0	0	1	1	0	0	0	C	= 30H
2	0	1	1	0	1	1	0	1	\
3	0	1	1	1	1	0	0	1	\
4	0	0	1	1	0	0	1	1	\
5	0	1	0	1	1	0	1	1	\
6	0	1	0	1	1	1	1	1	\
7	0	1	1	1	0	0	0	0	\
6	0	1	1	1	1	1	1	1	\
9	0	1	1	1	1	0	1	1	\

а) б)

Слика 3.4. Седумсегментен дисплеј и кодовите за цифрите од декадниот броен систем

Секој сегмент е составен од повеќе лед диоди и нивниот број е променлив. Сегментот ќе светне ако до сите негови лед диоди испратиме информација од логичка единица. Позициите на седумте сегменти се бележат со буквите a, b, c, d, e, f, g. На сликата 3.4.б. се дадени седумсегментните кодови за десетте цифри од декадниот броен систем од 0 до 9.

Пример 3.4: Кои сегменти треба да светнат за да се прикажат броевите 2 и 7?
Решение: За да се прикаже бројот 2 потребно е да светат сегментите a, b, g, e и d , а за бројот 7 сегментите a, b и c.

3.3. Глобална архитектура на микропроцесор

Микропроцесорот е составен од дигитални кола: логички кола, декодери, регистри, кола за одземање и собирање, мемории итн. Овие дигитални кола ја формираат архитектурата на микропроцесорот. Од архитектурата зависи функционалноста, способноста на микропроцесорот да извршува различни видови инструкции. Микропроцесорите од различни производители и генерации имаат различна архитектура, но 80% од хардверските компоненти се среќаваат речиси кај сите видови микропроцесори. Микропроцесорската организација не значи само именување на составните делови туку и познавање на нивната функција и нивно поврзување во функционална целина. Пред да се запознаеме со глобалната архитектура на еден општ микропроцесор да го објасниме **извршувањето на инструкциите** преку нивно разложување на повеќе фази. Инструкцијата ќе ја разложиме на пет фази:

- Пренос на операциски код од меморија во инструкцискиот регистар од микропроцесорот.
- Декодирање на операцискиот код во управувачката единица.
- Пренос на податоци од меморијата во некој од општите регистри на микропроцесорот.
- Ефективно извршување на инструкцијата во аритметичко логичката единица.
- Запишување на добиениот резултат во некој општ регистар од микропроцесорот или мемориска локација.

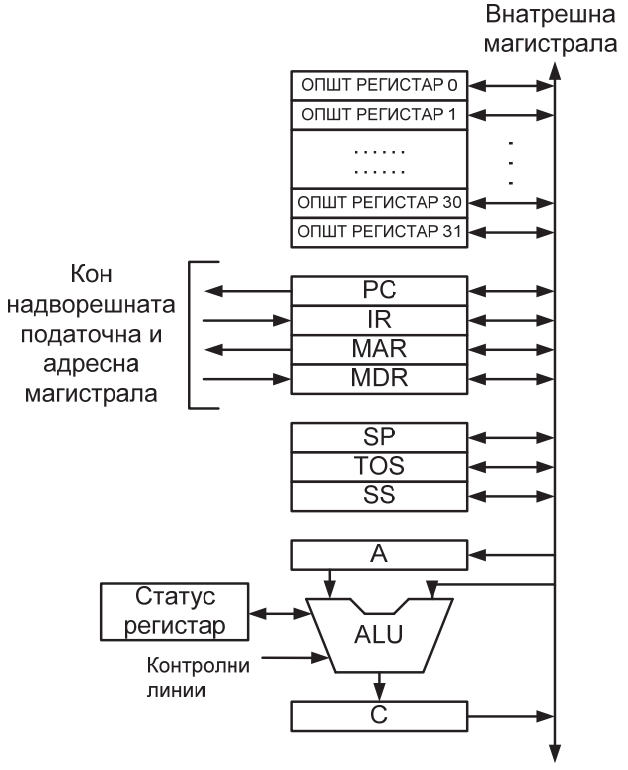
Секоја од петте фази активира одреден дел од хардверот на микропроцесорот, односно е извршена хардверска поделба на функциите. Разложувањето на инструкциите може да продолжи, при што е најдобро да нема хардверско мешање на истите. На ваков начин се подобрува брзината на работа, слично како кај производните ленти во фабриките.

Но, откако го разложивме извршувањето на инструкциите на повеќе фази почнуваат да се поставуваат многу прашања. Како микропроцесорот ја пронаоѓа следната инструкција за извршување? Како препознава за која инструкција станува збор? Како ја извршува инструкцијата во аритметичко логичката единица? Како знае каде да ги смести податоците за обработка и добиените резултати? Одговор на овие и други прашање ќе добиеме после анализата на архитектурата на микропроцесорот. На слика 2.3. е прикажана основната блок шема на општиот микропроцесор. Да се потсетиме дека составни делови на секој микропроцесор се: регистрите, аритметичко логичката единица и управувачката единица.

Заради поголема едноставност посебно ќе ја разгледаме архитектурата на секој составен дел и ќе ја објасниме нивната поврзаност.

3.3.1.Регистри

Регистрите се брзи мемориски локации во внатрешноста на микропроцесорот и служат за чување на податоци, инструкции и информации за состојбата на микропроцесорот. **Основна поделба** на регистрите е на општи регистри и регистри со специјална намена. На слика 3.5. прикажани се позначајните регистри и поврзувањето со внатрешната магистрала на микропроцесорот, надворешната податочната и адресна магистрала.

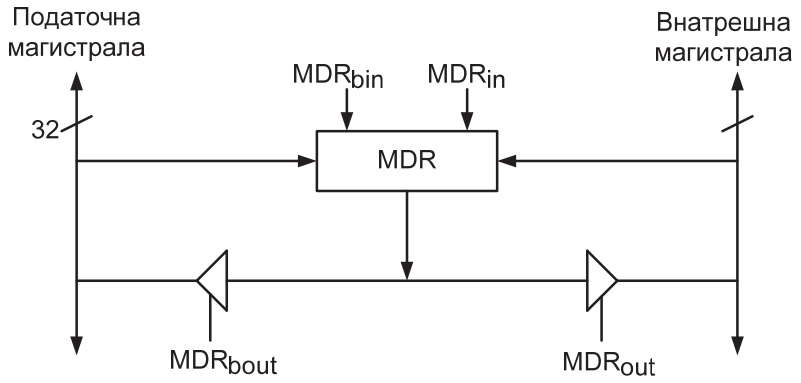


Слика 3.5. Регистри во општ модел на микропроцесор

Регистрите со специјална намена можат да бидат кориснички (пристап има програмерот) или системски. Друга **поделба на регистрите со специјална намена** е според функцијата. Тука би истакнале три групи: регистри за читање и пишување податоци во меморија (MAR, MDR), регистри за пренос на инструкција (PC,IR) и регистри за работа со стек меморија (SP, TOS, LV). За секој регистар постојат контролни сигнали кои овозможуваат поврзување со внатрешната магистрала или надворешната адресна и податочна магистрала. Контролните сигнали за поврзување со магистралите имаат префикс in или out во зависност од тоа дали сакаме да запишеме или прочитаме податок од регистарот, а доколку до овие префикси стои и буквата b тоа значи дека се работи за надворешна системска магистрала (англиски bus=магистрала).

На слика 3.6. е прикажано поврзувањето на регистарот MDR со магистралите и неговите контролни сигнали. Гледаме дека е извршено

баферирање на податочните линии при што се употребени бафери со три состојби.



Слика 3.6. Контролни сигнали за меморискиот податочен регистар

Сега ќе ја објасниме функцијата на поважните регистри со специјална намена.

- Не постои микропроцесор кој во својот состав не содржи регистар **програмски бројач** (PC=Program Counter). Тој ја содржи адресата на следната инструкција. Претпоставуваме дека инструкцијата е голема 32 бита. За да ја добиеме адресата на следната инструкција треба да додадеме четири на адресата на инструкцијата која моментално се извршува. Ова е така бидејќи една мемориска локација памти информација од еден бајт, па за меморирање на 32-битна инструкција ќе бидат потребни четири мемориски локации. Програмскиот бројач има три контролни сигнали: PC_{bout}, PC_{out} и PC_{in}. Кога микропроцесорот ја бара следната инструкција содржината на програмскиот бројач се носи на адресната магистрала преку активирање на сигналот PC_{bout}. Содржината на програмскиот бројач може да ја менува само управувачката единица при што новата содржина се пренесува преку внатрешната магистрала и за да таа се впише треба да се активира сигналот PC_{in}. Да напоменеме дека содржината на програмскиот бројач може да се пренесе истовремено до надворешната адресна магистрала и внатрешната магистрала преку активирање на сигналите PC_{bout} и PC_{out} .
- Содржната на регистарот PC се пренесува до системската адресна магистрала, се чека меморијата да ја смести инструкција на податочната магистрала и потоа инструкцијата се запишува во **инструкцискиот регистар** IR (Instruction Register). Поради едноставното поврзување со магистралите се користат два контролни сигнали IR_{bin} и IR_{out} при што едниот се користи кога треба да се запише инструкцијата од меморијата во регистарот, а другиот кога треба да се пренесе инструкцијата до управувачката единица.
- MAR (Memory Address Registar) или **меморискиот адресен регистар** ја чува адресата на податокот кој треба да се обработи во микропроцесорот. Некои инструкции во себе содржат адреса на

мемориска локација од која треба да се земе податокот за обработка. Оваа адреса се внесува во меморискиот адресен регистар преку активирање на контролниот сигнал MAR_{in}, а кога адресата треба да се пренесе до меморијата истата се испраќа по адресната магистрала преку активирање на контролниот сигнал MAR_{bout}. Содржината на меморискиот адресен регистар може да се пренесе на внатрешната магистрала преку активирање на сигналот MAR_{out}.

- MDR (Memory Data Register) или **меморискиот податочен регистар** го чува податокот кој треба да се прочита или запише во меморијата. Поврзувањето на овој регистар со магистралите е двонасочно, па поради тоа ќе ни требаат четири контролни сигнали: MDR_{bin}, MDR_{bout}, MDR_{in}, MDR_{out}. Ова е прикажано на слика 3.6.
- Со регистрите за работа со стек меморија ќе се запознаеме подоцна кога детално ќе биде објаснета стек меморијата и постапките за читање и пишување во неа.

Регистрите со општа намена се поврзани само со внатрешната магистрала. Моделот на микропроцесорна слика 3.5. содржи 32 регистри со општа намена кои се обележани со буквите од G_o до G_{31} каде што буквата G е прва буква од англискиот збор general што во превод значи општ. За секој општ регистар постојат по два контролни сигнали кои се бележат со G_{xin} и G_{xout}. Ове контролни сигнали служат за оспособување на влезовите односно излезите на општите регистри. Преку активирање на овие контролни сигнали микропроцесорот бира од кој регистар ќе чита податок и во кој регистар ќе запишува податок. Употребата на овие контролни сигнали ќе ја илустрираме преку инструкцијата за собирање

add R_d,R_{s1},R_{s2}

Со оваа инструкција се собираат содржините на општите регистри R_{s1} и R_{s2} и збирот се запишува во регистарот R_d. Буквата s доаѓа од англискиот збор source што во превод значи извор, а буквата d доаѓа од англискиот збор destination што во превод значи цел, дестинација. На пример, сакаме да ги собереме содржините на регистрите со реден број 5 и 7 и збирот да го сместиме во регистарот број 9. Ова значи дека треба да се изврши следната инструкција

add G9, G5, G7

Бидејќи имаме само една внатрешна магистрала, содржината на еден изворен регистар треба да се префрли во привремениот регистар A. Потоа содржината на другиот регистар ја пренесуваме на внатрешната магистрала A и истите ги собираме. **Активирањето на контролните линии** можеме да го поделиме во три фази:

1. Го активираме сигналот G_{5out} за да ја пренесеме содржината на регистарот G5 на внатрешната магистралата. Истовремено го активираме сигналот A_{in} за да содржината од внатрешната магистралата ја пренесеме во регистарот A.

2. Сега содржината на регистарот G7 треба да се пренесе до внатрешната магистралата преку активирање на сигналот G_{7out}. Следен чекор ќе биде активирање на контролните сигнали за аритметичко логичката единица. Состојбата на овие контролни сигнали треба да одговара на петата редица од табелата 3.4. Преку активирање на сигналот C_{in} збирот влегува во регистарот C.
3. Последен чекор е запишување на вредноста на регистарот C во регистарот G_9. Овој пренос е овозможен преку истовремено активирање на сигналите C_{out} и G_{9in}.

Во табелата 3.3. е прикажан редоследот на активирање на контролните сигнали за секој чекор.

чекори	Контролни сигнали
1	G_{5out}; A_{in}
2	G_{7out}; ALU:F_o, F_1, ENA, ENB; C_{in}
3	C_{out}; G_{9in}

Табела 3.3. Контролни сигнали за операцијата собирање на содржината на два општи регистри

3.3.2. Аритметичко логичка единица

Микропроцесорот е ,,мозокот" на компјутерот, а **аритметичко логичката единица е ,,мозокот" на микропроцесорот.** Преку оваа единица се извршуваат сите инструкции од инструкциското множество. Посложените операции не се изведуваат директно туку со комбинирање на повеќе едноставни.

Начинот на поврзување на аритметичко логичката единица со регистрите на микропроцесорот е прикажана на слика 3.5. Од сликата гледаме дека аритметичко логичката единица не е поврзана со надворешните магистрали на микропроцесорот туку само со внатрешната магистрала, преку која прима податоци од регистрите. Исто така, на сликата се прикажани двата привремени, работни регистри на аритметичко логичката единица, A и C. Регистарот A се наоѓа над единицата и служи за привремено чување на податокот пред тој да влезе во единицата и се обработи. Регистарот C служи за привремено чување на добиениот резултат пред истиот да биде сместен во некој од општите или специјалните регистри на микропроцесорот.

Аритметичко логичката единица е поврзана директно со еден специјален регистар таканаречен **статус регистар**. Битовите од овој регистар се нарекуваат **знаменца** (англиски flag). Знаменцата се индикатори на состојба односно истите даваат информација за состојбата на битовите во последно добиениот резултат. За да се ,,крене" односно активира некое знаменце потребно е да биде исполнет одреден услов. Со секоја следна генерација на

микропроцесори бројот на знаменца се зголемува, но како најважни кои се среќаваат речиси кај сите микропроцесори ќе ги споменеме:

- Знаменцето нула Z (англиски-zero) се активира кога ќе се добие резултат еднаков на нула.
- Знаменцето S (англиски-sign) се активира кога ќе добиеме негативен резултат
- Знаменцето C (англиски-carry) се активира кога имаме пренос на највисоката позиција во бинарниот број. На пример ако собираме осумбитни броеви ова знаменце ќе се активира ако имаме пренос од осмата на деветата позиција. Кај осумбитните податоци девета позиција не постои , па битот за пренос ќе се запише во знаменцето C.
- Знаменцето O (англиски-overflow) се активира при пречекорување односно кога ќе се добие број поголем или помал од предвиденото. Оваа знаменце се користи само при работа со броеви со предзнак. Ако ја погледнеме слика 3.2. ќе заклучиме дека ова знаменце се активира ако добиеме резултат поголем од +127 или помал од -128. Кај броевите без предзнак индикатор за настанато пречекорување е знаменцето за пренос C.

Со структурата на аритметичко логичката единица веќе се запознавме во првата тема кога зборувавме за комбинациски кола и истата беше прикажана на слика 1.20. Беа објаснети функциите на сите логички кола, двата податочни влеза A и B и контролните сигнали за избор на операција F_0 и F_1. Но, покрај овие два **контролни сигнали** постојат уште четири: ENA (enable A), ENB (enable B), INVA (invert A) и INC (increase). Сигналите ENA и ENB служат за оспособување на податочните влезови. Со сигналот INVA се инвертира битот на влезот A. Сигналот INC служи за додавање на единица на бинарниот број. Со шесте контролни сигнали можат да се направат 64 различни комбинации, но не се користат сите. Некои од поинтересните комбинации се прикажани во табелата 3.4. Во наведените функции аритметичкото собирање е најчесто користена операција. На пример кај операцијата A+1, ENB е еднаков на нула бидејќи податочниот влез B воопшто не ни е потребен, но сигналот INC е еднаков на еден бидејќи со него додаваме единица. Ако сигналите INVA и INC се еднакви на еден тогаш пресметуваме втор комплемент од вредноста на A односно ја пресметуваме негативната вредност на A, -A. Ако го оспособиме податочниот влез B тогаш ќе ја добиеме функцијата B+(-A)=B-A, одземање на првиот податочен влез од вториот. Можеби некој ќе се запраша зошто се потребни првите две функции од табелата 3.4. кога на излез од аритметичко логичката единица се добива истото она што е и на најзиниот влез. Оваа функција многу често се користи при преносот на податоци од еден во друг регистар бидејќи регистрите не се директно поврзани меѓу себе. На пример, податокот не може директно да помине од еден во друг општ регистар, туку меѓу нив посредува аритметичко логичката

единица. Содржината на првиот регистар се носи на влез од аритметичко логичката единица, а излезот од аритметичко логичката единицата се носи во вториот регистар што претставува дестинација. Од првите две редици на табелата гледаме дека за да пресликаме еден од податочните влезови на излез од аритметичко логичката единица се користи операција логичко собирање (ИЛИ) при што само еден од сигналите за оспособување е активен.

F0	F1	ENA	ENB	INVA	INC	функција
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	Негација A
1	0	1	1	0	0	Негација B
1	1	1	1	0	0	A+B
1	1	1	1	0	1	A+Б+1
1	1	1	0	0	1	A+1
1	1	0	1	0	1	B+1
1	1	1	1	1	1	B-A
1	1	0	1	1	0	B-1
1	1	1	0	1	1	-A
0	0	1	1	0	0	A $\vee$B
0	1	1	1	0	0	A$\wedge$ B
0	1	0	0	0	0	0
0	1	0	0	0	1	1
0	1	0	0	1	0	-1

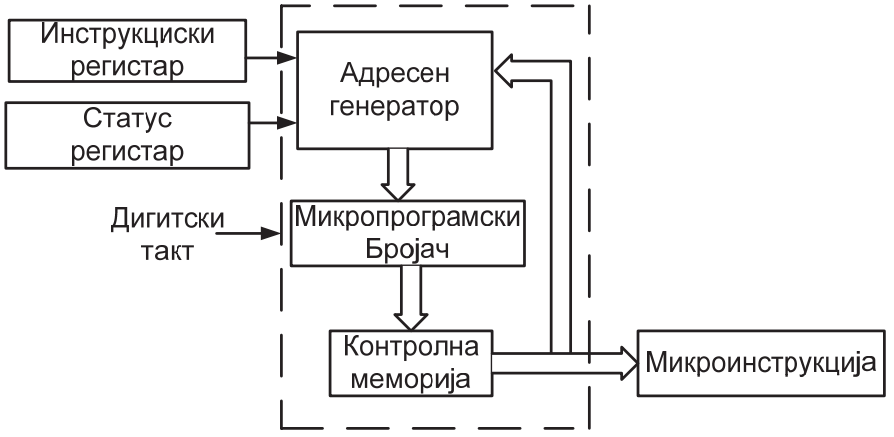
Табела 3.4. Табела на вистинитост за еднобитна аритметичко логичка единица

3.3.3. Управувачка единица

Најважна функција на управувачката единица е извршување на инструкциите една по една се додека не заврши започнатата програма. Управувачката единица го врши извршувањето на инструкцијата преку генерирање на контролни сигнали за регистрите, аритметичко логичката единица, меморијата или периферните уреди. На слика 3.7. е прикажана основната блок шема на управувачката единица.

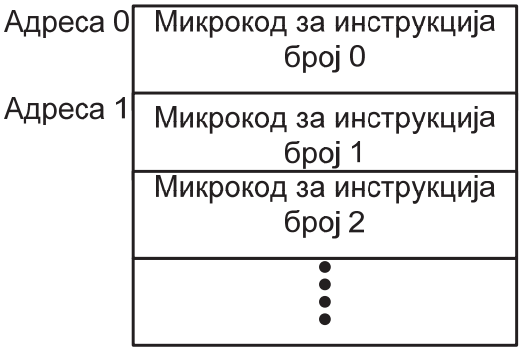
На влез од управувачката единица се носи содржината на инструкцискиот регистар, статус регистарот и дигитскиот такт. Првите осум битови од инструкцијата претставуваат операциски код и тој служи за препознавање на инструкцијата. Да се потсетиме на поделба на инструкциите во четири групи: инструкции за пренос, аритметички, логички инструкции и инструкции за контрола на текот на програмата. Контролните инструкции можат да бидат условни. За да тие се извршат потребна е информација за состојбата

на знаменцата во статус регистарот. Содржината на инструкцискиот регистар и статус регистарот претставуваат всушност влезови на адресниот генератор, кој генерира адреси за пристап до микропрограмата.



Слика 3.7. Управувачка единица на општ модел на микропроцесор

Најважен дел од управувачката единица е **микропрограмата.** Микропрограмата се наоѓа во внатрешноста на микропроцесорот, во посебна ROM меморија, наречена контролна меморија. Микропрограмата е составена од микрокодови. Бројот на микрокодови е еднаков на бројот на инструкции во инструкциското множество на микропроцесорот. За секоја инструкција има по еден микрокод. Микрокодовите се составени од микроинструкции. Бројот на микроинструкции во еден микрокод е еднаков на бројот на фази, чекори на кои е поделено извршувањето на инструкцијата. За секој чекор има по една микроинструкција. Ваквата линеарна организација на микропрограмата е прикажана на слика 3.8.



Слика 3.8. Организација на микропрограмата

Микропроцесорот генерира контролни сигнали. За секој контролен сигнал постои посебен бит во **микроинструкцијата**. Контролниот сигнал и неговиот бит во микроинструкцијата имаат иста вредност, логичка нула или единица. Ако имаме 32 општи регистри и ако за секој регистар требаат по две контролни сигнали тоа значи дека 64 битови од микроинструкцијата ќе бидат за контрола на општите сигнали. Во состав на микроинструкцијата влегуваат и битовите за контрола на аритметичко логичката единица и регистрите со

специјална намена. Микринструкцијата може да има повеќе од 90 битови. Да истакнеме дека сите микроинструкции се со иста големина. Колку е поголема микроинструкцијата толку е побрзо извршувањето на инструкциите. Се разбира поголемата микроинструкција значи поголем број контролни линии односно трошење на хардвер.

Во табелата 3.3. беа дадени контролните сигнали за сите чекори при извршувањето на инструкцијата add G9, G5, G7. Бидејќи има три чекори значи дека микрокодот за оваа инструкција ќе содржи три микроинструкции. Во првата микроинструкција на високо ниво ќе бидат само битовите G_{5out} и A_{in}, во втората G_{7out}, ALU, F_o, F_1, ENA, ENB, C_{in} и во третата C_{out} и G_{9in}. Сите остананти битови во микроинструкцијата ќе бидат еднакви на нула.

Времетраењето на една микроинструкција изнесува еден дигитски такт. Поради тоа е потребно дигитскиот такт да се донесе на влез од управувачката единица. Преку него всушност се врши броење на микроинструкциите од микропрограмата.

3.4. Заеднички карактеристики на микропроцесор

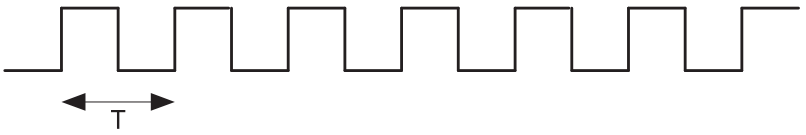
Хардверски микропроцесорот може да извршува многу едноставни операции, но со огромна брзина. Сложените инструкции се добиваат со комбинирање на повеќе едноставни инструкции. Брзината на микропроцесорот се изразува во MIPS (милиони инструкции во една секунда). Но, тој е само една од компонентите што придонесуваат за вкупната брзина на целиот компјутер. За да микропроцесорот брзо ги извршува инструкциите, тие треба брзо да му се доставуваат, а тоа зависи од брзината на работата на магистралите и на мемориите. Ќе наведеме неколку фактори што влијаат врз брзината на работата на микропроцесорот, но има и многу други за кои ќе стане збор натаму.

Работна фреквенција

Наједноставно би било кога брзината на работата на микрокомпјутерот би се изразувала во број на извршени операции во 1 секунда. Меѓутоа, проблем е што различни операции траат различно време. Брзината на работата на микропроцесорот се изразува во херци [Hz].

Имено, во самиот микропроцесор (но може и надвор од него) има вградено кварцен кристал, кој постојано со константна брзина генерира правоаголни импулси. Овој кристал се вика генератор на такт и ги дава дигитските тактови во микропроцесорот. Од фреквенцијата на овие импулси

зависи брзината на работата на микропроцесорот, а со тоа и на самиот компјутер.



Слика 3.9. Излезен сигнал од генераторот на дигитски такт

Основна карактеристика на периодичната поворка правоаголни импулси е периодата (времетраење на една осцилација). Периодата се пресметува како реципрочна вредност од фреквенцијата. Фреквенцијата, всушност, ни покажува колку периоди има во една секунда.

Кај осумбитните микропроцесори кристалот емитува од 1.000.000 од 4.000.000 импулси во секунда (од 1MHz до 4MHz). Кај Пентиум 3 оваа фреквенција изнесува 1GHz. Денешните микропроцесори имаат работна фреквенција од 2 до 4 GHz. Веќе истакнавме дека различни операции траат различно време. На пример, за да се изврши трансфер на еден осумбитен податок од RAM меморијата до микропроцесорот потребни се три такта, а за да се изврши аритметичко собирање на два броја потребни се четири такта. Битно е да се запамти дека времетраењето на сите операции се изразува во цел број на дигитски такта. Можеби дадена операција трае 3.2 такта, но микропроцесорот смета дека на таа операција се потрошени 4 такта, што значи 0.8 такт е залудно потрошен. Овој пренос се вика синхрон пренос. Доколку секој уред би имал свој посебен начин на мерење на времетраењето на операциите (се мисли на различно времетраење на тактовите), тогаш многу лесно може да настане хаос. Микропроцесорот треба да памти колкава е брзината на секој уред и да ги синхронизира, што значи големо губење време.

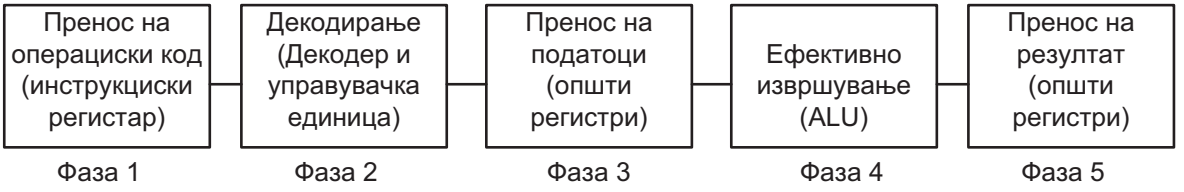
Големина на податоци

Според големината на обработуваните податоци, микропроцесорите се делат на 8, 16, 32, 64 и 128-битни микропроцесори. Микропроцесорите што обработуваат поголеми податоци се помоќни. Но, се разбира, за да се постигне тоа, потребни се вложувања во хардверските ресурси. Така, на пример, кај осумбитните микропроцесори постојат 8 пина за пренос на податоци, на матичната плоча има 8 податочни водови, секој општ регистар во микропроцесорот содржи по 8 флип-флопа. Кај 16-битните микропроцесори сè е двапати поголемо. Ако, на пример, 16-битниот микропроцесор има само 8 податочни пина, тогаш 16-битните податоци ќе треба да се пренесуваат на двапати, а тоа значи губење време.

Проточен концепт (PIPELINE)

Концептот Pipeline значи проточно, последователно извршување на повеќе инструкции во микропроцесорот, но секоја инструкција треба да биде во

различна фаза. На сликата 3.10. се прикажани петте фази на извршување на една инструкција и хардверските компоненти потребни за секоја фаза.



Слика 3.10. Фази на извршување на инструкција во микропроцесорот

На сликата 3.11. со квадратчиња се означени инструкциите, а бројката внатре е реден број на инструкцијата, од каде лесно се воочува последователноста на извршувањето на инструкцијата.

Слика 3.11. Зависност на фазите од времето кај проточниот концептот

За да се изврши инструкцијата број 2, прво мора да се изврши инструкцијата број 1. Пред инструкцијата број 3 се извршува инструкцијата број 2 итн. Во првиот дигитски такт инструкцијата број 1 е во фаза 1, односно инструкцијата се пренесува од меморијата до микропроцесорот. Инструкциите минуваат од фаза во фаза сè додека не биде извршена и последната фаза - запишување на добиениот резултат. Од сликата 3.11. се гледа дека секоја фаза трае еден такт (види генератор на такт). Значи, за 9 такта ќе бидат извршени 5 инструкции. Ако не беше pipeline за извршување на една инструкција ќе беа потребни 5 такта, а останатите инструкции ќе чекаа. За да се извршат 5 инструкции, ќе требаше вкупно: 5 такта ·5 инструкции = 25 такта.

CISC и RISC

Првите микропроцесори биле произведени во CISC (Complex Instructions Set Computer). Оваа концепција се карактеризира со многу големо множество од инструкции и микропроцесорот треба да извршува многу сложени операции. Преку бројот на инструкции што може да ги извршува еден микропроцесор се согледувала неговата моќност. Од појавата на

микропроцесорот па сè до денес нивната брзина постојано се зголемува. Меѓутоа, зголемувањето на брзината и зголемувањето на бројот на операции се спротивни барања. Се поставува прашањето - што треба да се жртвува за да се добие подобар микропроцесор. Со детална анализа е заклучено дека кај повеќето микропроцесори најмногу се користат околу 20 % од целото множество инструкции. Од ваквата анализа произлегува идејата да се направат многу брзи микропроцесори со мало инструкциско множество. Така е создаден посебен правец во изградбата на микропроцесорите заснован врз RISC (Redused Instruction Set Computer) концепција. Кај оваа концепција преку едноставни микропроцесори (обично со мал број транзистори) се постигнуваат многу големи брзини. RISC концептот на програмерот му остава поголем простор за креативност - од попрости инструкции да создава посложени. Не станува збро за просто намалување на бројот на инструкции, туку за нивна рационализација.

Денес се смета дека постојат неколку принципи врз кои се заснова градбата на RISC микропроцесорите:

- Намалување на бројот на тактови за извршување на една инструкција;
- Користење фиксен формат на инструкцијата за да се поедностави декодирањето;
- Намалување на бројот на инструкции за пристап до главната меморија;
- Намалување на бројот на кодови кои ги препознава микропроцесорот.

Голем дел од овие принципи се реализирани на хардверско ниво. RISC концептот овозможува поинтензивно користење на паралелните линии и се отстранети голем дел од колата за декодирање.

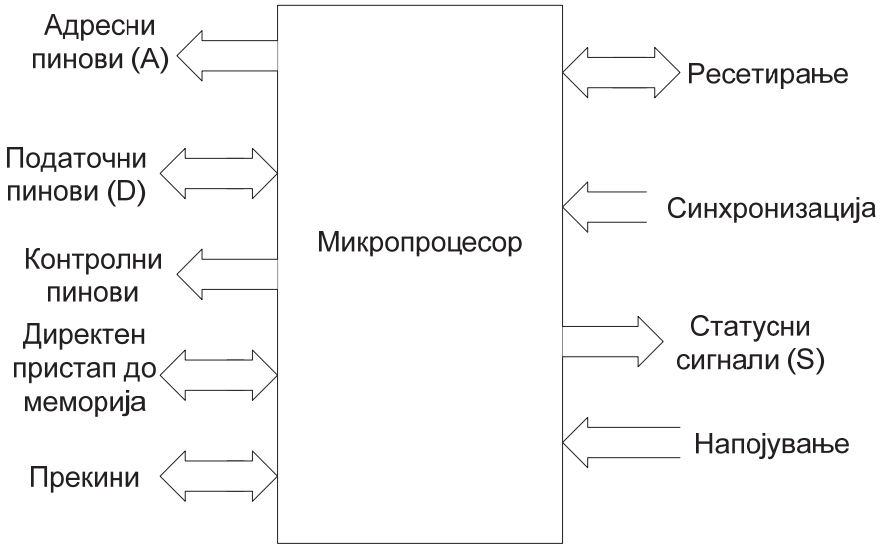
3.5. Пин-дијаграм на микропроцесор

Пиновите се продолжетоци на внатрешните водови на интегрираното коло и потоа тие се поврзуваат со водовите на матичната плоча. На сликата 3.12. е прикажан пин-дијаграм на општ модел на микропроцесор. Пин-дијаграмот го дава распоредот на пиновите и нивните ознаки. Ознаките на пиновите се кратенки од англиски зборови, со кои накратко се опишува нивната функција.

Пиновите се поделени во неколку групи.

- Пиновите со ознака А се **адресни пинови**. Тие се секогаш излезни пинови за микропроцесорот, а влезни пинови за мемориските чипови. Да се потсетиме дека адресата е единствена бинарна комбинација за избор на една локација од меморијата или периферен уред. Од вкупниот број адресни линии зависи големината на адресниот просор на

микропроцесорот. Тој се пресметува според релацијата 2^n каде што n е вкупен број адресни пинови.



Слика 3.12. Пин дијаграм на општ модел на микропроцесор

- Пиновите со ознака D се **податочни пинови**. Буквата D е првата буква од англискиот збор DATA, што во превод значи податок. Податочните пинови се влезно-излезни што зависи од тоа дали микропроцесорот чита или запишува податоци. Бројот на податочни пинови може да изнесува 8, 16, 32 или 64 што зависи од големината на обработуваните податоци.

- Контролните пинови се разноимени и секој од нив има различна функција. Пиновите $\overline{WR}$ (write) и $\overline{RD}$ (read) служат за избор на операција меѓу пишување и читање. Нивната негација ни покажува дека тие се активни на логичка нула. Пинот IO/$\overline{M}$ (input output/$\overline{memory}$) служи за избор на уред за комуникација, меѓу влезно-излезните уреди или мемориски уред.

- Освен контролни пинови, постојат и **статусни пинови** кои се бележат со S_0 и S_1. Тие даваат информација за значењето на пренесуваните податоци и за уредот од кој се пренесуваат.

- Пинови за **прекини** се INTR (interrupt) и INTA (interrupt acknowledge). Прекините се посебно организирани механизми за комуникација на периферните уреди со микропроцесорот. Доколку се појави грешка во работењето на периферниот уред, тогаш тој испраќа барање за прекин до микропроцесорот преку пинот INTR. Доколку микропроцесорот дозволи да биде прекинат во извршувањето на тековната програма, тогаш тој го активира пинот за потврда на прекин INTA. Потоа следува сервисирање на појавената грешка.

- **HOLD и HLDA** (Hold Acknowledge) се пинови за барање и потврда на директен пристап до меморија (DMA-Direct Memory Access). За да се ослободи од преголемиот број управувачки функции, микропорцесорот му дозволува на еден специјален контролер да управува со арбитражата на магистрали и директно да пристапува до меморијата.

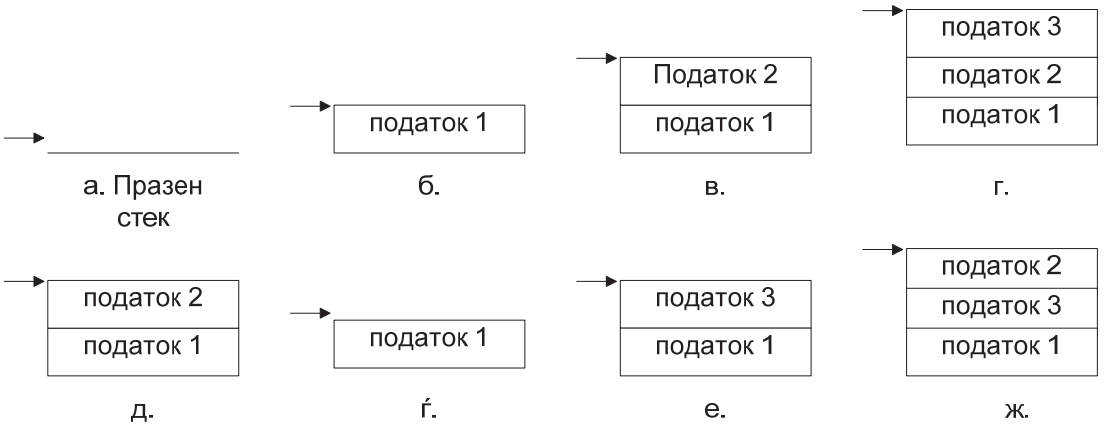
- Сигнали за управување со самиот микропорцесор. Во оваа група спаѓа сигналот RESET и под негово дејство микропроцесорот ја прекинува тековната работа и доаѓа до ресетирање на сите негови регистри, односно бришење на нивната содржина.

- Сигнали за синхронизација. Под синхронизација се подразбира усогласување на брзините на работа. По правило, микропроцесорот е најбрза компонента и тој мора да ги чека другите побавни компоненти, како што е меморијата. Во оваа група спаѓа сигналот CLK (Clock) кој претставува дигитски такт. Да напоменеме дека секоја операција што се извршува во микропроцесорот трае цел број од дигитскиот такт.

- Напојување. Во оваа група сигнали спаѓаат еднонасочниот напон Vcc и заземјувањето што се бележи со GND (Ground).

3.6. Употреба на стек меморија

Стекот е посебно организиран мемориски простор во внатрешноста на микропроцесорот која се полни и празни преку најгорната локација, врвот. Стек меморијата е позната и под кратенката LIFO (Last In First Out) што во превод значи **податокот кој последен влегол, прв ќе излезе**. Стек меморијата е повеќеслојна меморија бидејќи запишувањето на новите податоци може да се спореди со пластење на слоеви. Секој нов податок се запишува на врвот на стекот. Но, исто така микропроцесорот може да го прочита само податокот од најгорната локација на стекот (врвот). Ако сакаме да земеме некој податок сместен во средината на стекот треба да ги испразниме сите мемориски локации над саканиот податок, со што автоматски саканиот податок (мемориска локација) ќе стаса на врвот од стекот. На слика 3.13. е демонстрирано полнењето и празнењето на стек меморијата со податоци. На почетокот стекот е празен (слика 3.13.а.). Потоа го внесуваме првиот податок (слика 3.13.б.). Стрелката од страна го покажува податокот кој во дадениот момент е достапен за микропроцесорот, односно покажува на најгорната локација од стек меморијата. Вториот податок се сместува над првиот и стрелката се крева за едно место погоре (слика 3.13.в.). Потоа го внесуваме и третиот податок (слика 3.13.г.).



Слика 3.13. Полнење и празнење на стек меморијата

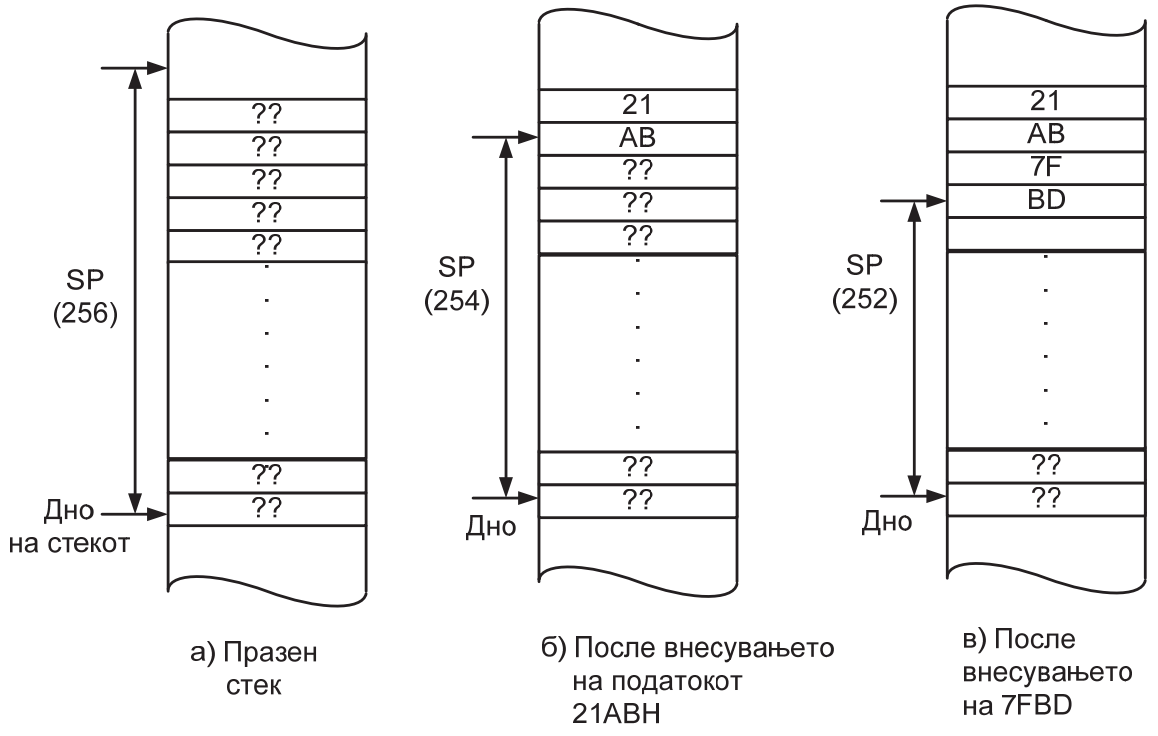
Сега сакаме вториот податок и третиот податок да си ги заменат местата. Тоа не може да се постигне со едноставно вметнување на третиот податок над првиот туку мора да се извади прво третиот податок (слика 3.13.д.), па вториот (слика 3.13.ѓ.), а потоа повторно да се внесат, но по спротивен редослед, прво третиот(слика 3.13.е.), а потоа вториот (слика 3.13.ж.).

Најважен регистар кој се користи при работа со стек меморијата е **стек покажувачот, SP (Stack Pointer)**. Овој регистар ја содржи адресата на последно внесениот податок во стек меморијата, односно адресата на моментално единствениот достапен податок од стекот. Регистарот TOS (Top of Stack) всушност го содржи податокот кој го покажува стек покажувачот. Кај некои микропроцесори постои специјален регистар кој ја памти адресата на најдолната локација од стек меморијата односно неговото дно. Таков регистар е регистарот стек сегмент (SS-Stack Segment), кој за првпат се среќава кај микропроцесорот 8086.

Но наместо вредноста на стек покажувачот да се зголемува со внесувањето на секој нов податок, неговата вредност се намалува. Истото се случува и при празнење на стекот, но во спротивна насока. При празнење на стекот, вредноста на стек покажувачот не се намалува туку се зголемува. Ова е така бидејќи новиот податок не се става над претходно внесениот, туку под него. Ова значи дека сликата 3.13. треба да ја свртиме наопаку, за да таа одговара на вистинската состојба. Сликата 3.13. ја претставивме така бидејќи се чинеше дека така е поедноставо за објаснување. Како се менува вредноста на стек покажувачот при полнење и празнење на стекот е објаснето преку слика 3.14.

Дното и врвот на стекот се неменливи. Со секое полнење и празнење на стекот се менува неговата ,,длабочина", која се мери од врвот на стекот до локацијата на последно внесениот податок. Вредноста на стек покажувачот не ја претставува ,,длабочината" туку растојанието од дното на стекот до

локацијата на последно внесениот податок. Стрелката која одговара на положбата на стек покажувачот се симнува надоле кога стекот се полни , а се крева нагоре кога стекот се празни.



а) Празен стек

б) После внесувањето на податокот 21ABH

в) После внесувањето на 7FBD

Слика 3.14. Промена на вредноста на регистарот стек покажувач (SP-Stack Pointer)

Стек меморијата се користи за чување на податоци и при работа со потпрограми. Бројот на општи регистри не е секогаш доволен и може да се случи сите општи регистри да бидат зафатени при извршувањето на програмата. Ако некој од општите регистри содржи константа која нема да се користи во блиско време, туку подоцна тогаш содржината на тој регистар може де се заштити преку нејзино пренесување во стек меморијата. Кога микропроцесорот ќе ја побара константата истата ќе биде вратена назад од стек меморијата во некој општ регистар.

Стек меморијата се користи за заштита на вредноста на програмскиот бројач и статус регистарот кога се врши премин од главната програма во потпрограма. Имено кога ќе започне потпрограмата овие два специјални регистри ќе се наполнат со други вредности кои ги диктира самата потпрограма. Проблемот се јавува кога ќе треба да се вратиме од потпрограмата во главната програма. Ако претходно не се заштити содржината на програмскиот бројач микропроцесорот нема да знае на која адреса да се врати во главната програма.

Заклучоци:

Ако работиме со броеви без предзнак тогаш со осум битови можат да се престават 256 броеви, од нула до 255. Ако работиме со броеви со предзнак тогаш со осум битови можат да се престават броевите од -128 до +127.

BCD кодот е многу често користен бинарен код. Кај овој код кодните зборови не се добиваат преку делење со два и запишување на остатокот. Кај BCD кодот секоја цифра од декадниот број се запишува со 4 бинарни цифри.

Современите процесори содржат аритметичко логичка единица за обработка на цели броеви и единица за обработка на реални броеви односно таканаречени броеви со подвижна запирка.

За преставување на карактери во бинарен облик се користи ASCII кодот. 7-сегментниот код се користи за визуелно прикажување на броеви од декаден броен систем на 7-сегментен дисплеи.

Извршувањето на инструкциите во микропроцесорот се дели на повеќе фази од кои позначајни се: пренос на операциски код, декодирање, пренос на операнди од меморијата во општите регистри, ефективно извршување во аритметичко логичката единица и запишување на добиениот резултат.

Регистрите со специјална намена можат да бидат кориснички (пристап има програмерот) или системски. Друга поделба на регистрите со специјална намена е според функцијата. Тука би истакнале три групи: регистри за читање и пишување податоци во меморија (MAR, MDR), регистри за пренос на инструкција (PC,IR) и регистри за работа со стек меморија (SP, TOS, LV). За секој регистар постојат контролни сигнали кои овозможуваат поврзување со внатрешната магистрала или надворешната адресна и податочна магистрала.

Знаменцата се индикатори на состојба односно истите даваат информација за состојбата на битовите во последно добиениот резултат. Знаменцето нула Z (англиски-zero) се активира кога ќе се добие резултат еднаков на нула. Знаменцето S (англиски-sign) се активира кога ќе добиеме негативен резултат Знаменцето C (англиски-carry) се активира кога имаме пренос на највисоката позиција во бинарниот број. Знаменцето O (англиски-overflow) се активира при пречекорување односно кога ќе се добие број поголем или помал од предвиденото. Оваа знаменце се користи само при работа со броеви со предзнак.

Микропроцесорот управува со работата на аритметичко логичката единица преку генерирање на соодветни контролни сигнали. Контролните сигнали F_0 и F_1 служат за избор на операција. Сигналите ENA и ENB служат за оспособување на податочните влезови. Со сигналот INVA се инвертира битот на влезот A. Сигналот INC служи за додавање на единица на бинарниот број.

Најважен дел од управувачката единица е микропрограмата. Микропрограмата е составена од микрокодови. За секоја инструкција има по еден микрокод. Микрокодовите се составени од микроинструкции. Бројот на микроинструкции во еден микрокод е еднаков на бројот на фази, чекори на кои е поделено извршувањето на инструкцијата. За секој контролен сигнал постои посебен бит во микроинструкцијата.

Времетраењето на инструкциите се мери во цел број на дигитски такта. Генераторот на дигитски такт претставува кварцен кристал вграден во самиот микропроцесорски чип, но може да биде и надвор од него.

Концептот на проточно извршување на инструкции значи истовремено извршување на повеќе инструкции, но секоја инструкција е во различна фаза и за секоја фаза е потребна различна хардверска компонента.

CISC концептот се карактеризира со многу големо инструкциско множество кое во себе содржи и сложени инструкции. RISC концептот значи рационализација на инструкциското множество со цел да се намали времетраењето на инструкциите. Адресните пинови се излезни пинови од микропроцесорот и служат за избор на една мемориска локација од работната меморија. Податочните пинови се влезно-излезни што зависи од тоа дали микропроцесорот чита или запишува податоци.

Пинот IO/$\overline{M}$ (input output/$\overline{memory}$) служи за избор на уред за комуникација, меѓу влезно-излезните уреди или мемориски уред.

Пинови за прекини се INTR (interrupt) и INTA (interrupt acknowledge). Прекините се посебно организирани механизми за комуникација на периферните уреди со микропроцесорот.

Стекот е посебно организиран мемориски простор во внатрешноста на микропроцесорот која се полни и празни преку најгорната локација, врвот. Стек меморијата е повеќеслојна меморија бидејќи запишувањето на новите податоци може да се спореди со пластење на слоеви. Стек меморијата се користи за чување на податоци и при работа со потпрограми.

Стек покажувачот, SP (Stack Pointer) ја содржи адресата на последно внесениот податок во стек меморијата, односно адресата на моментално единствениот достапен податок од стекот.

Прашања и задачи:

1. Наброј ги микропроцесорите произведени од компанијата „Интел“ според нивниот хронолошки развој?

2. Престави го зборот processor со ASCII код!

3. Пресметај го опсегот на декадни броеви со и без предзнак кои можат да се прикажат во бинарен броен систем ако се користат 16-битни податоци!

4. Наброј ги сегментите кои ќе светнат ако се прикаже бројот четири на седумсегментниот дисплеј!

5. Кој код се чува во инструкцискиот регистар?

6. Како се поделени регистрите во внатрешноста на микропроцесорот?

7. Објасни ја функцијата на меморискиот адресен и меморискиот податочен регистар!

8. Наброј и објасни ги контролните сигнали на еднобитната аритметичко логичка единица!

9. Наброј и објасни ја функцијата на знаменцата во статус регистарот!

10. Каде се чува микропрогамата на микропроцесорот и за што служи истата?

11. Која е разликата меѓу инструкција и микроинструкција?

12. Наброј ги фазите на кои е поделено извршувањето на една инструкција во микропроцесорот!

13. 8086 микропроцесорот е 16-битен микропроцесор. Објасни што значи овој исказ?

14. Кои се предностите и недостатоците на RISC концептот во однос на CISC концептот?

15. Објасни го проточното извршување на инструкциите во микропроцесорот?

16. Објасни ја функцијата на адресните и на податочните пинови на микропроцесорот!

17. Во каква логичка состојба ќе бидат контролните пина $\overline{WR}$, $\overline{RD}$, IO/$\overline{M}$ ако микропроцесорот запишува податок во некој периферен уред?

18. Објасни за што служат пиновите HOLD и HLDA!

19. Зошто доаѓа до губење на резултатот од извршената инструкција запишан во општите регистри на микропроцесорот 8085?

20. Каква функција има програмскиот бројач?

21. Наброј две инструкции за кои адресата на следната локација од RAM меморијата не се добива со додавање единица на содржината на програмскиот бројач.

22. Објасни каква функција има сигналот CLK во работата на микропроцесорот.

23. Како е организирана стек-меморијата?

24. За што служи регистарот стек-покажувач?

25. Објасни каква примена има стек меморијата!

4. Микропроцесорски системи

4.1. Основни поими при поврзувањето на микропроцесорот

Под поимот **интерфејс** се подразбира начин на поврзување, приспособување, со цел да се постигне целосна компатибилност меѓу уредите во составот на еден систем. Перформансите на еден уред може да бидат сосема задоволувачки, но истиот уред да не може да се вгради, бидејќи неговите влезно-излезни карактеристики не се усогласени со карактеристиките на веќе вградените уреди во системот. Усогласувањето, поврзувањето, односно интерфејсот не се остварува само преку електронски, хардверски компоненти, туку честопати е потребен и специјален софтвер со кој ќе се надминат постојните разлики меѓу уредите.

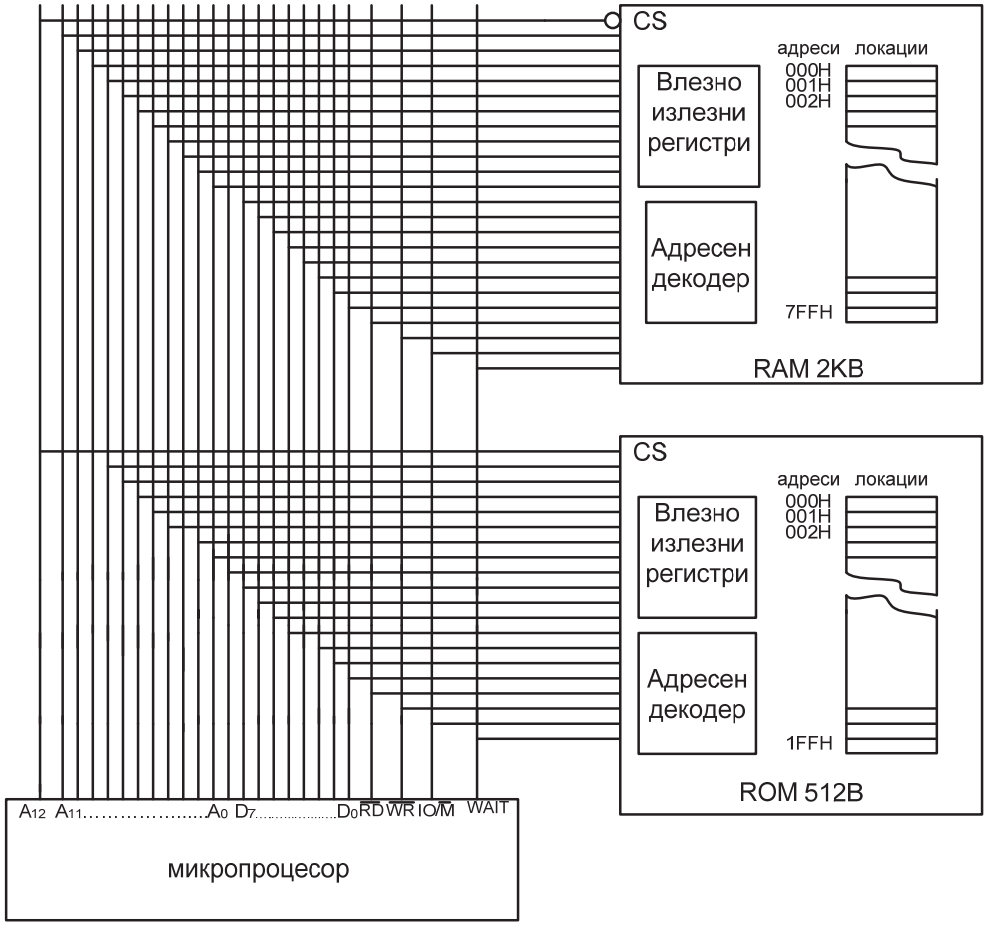
Синхронизацијата е многу важна за реализација на успешен интерфејс. Под синхронизација се подразбира усогласување на брзините на двата уреда. За таа цел се користат најразлични видови бројачи. На пример, уредот што ги испраќа податоците е четири пати побрз од уредот што ги прима. На првиот уред му треба еден дигитски такт за да ги испрати податоците, а на вториот 4 такта за да ги прими. Дигитскиот такт од предавателот се носи на влез од бројач, кој на својот излез ќе даде еден такт по 4 такта на влезот. Имено, бројачот претставува регистар чија соджина се зголемува за еден со секој влезен такт. Во овој случај кога содржината на бројачот ќе стане $100_2=4_{10}$, бројачот ќе го активира својот излез и тогаш сигналот се носи до приемникот кој ќе ги прими податоци. Микропроцесорот е побрз уред од меморијата. Ако не постои синхронизација, тогаш меморијата нема да може да ги прими сите податоци што ги испратил микропроцесорот и ќе дојде до губење на податоци. Затоа, во временскиот систем на микропроцесорот мора да се воведат неколку дигитски такта чекање за да ѝ се даде доволно време на меморијата да ги зачува испратените податоци.

Под **сигнализација** се подразбира размена на контролни сигнали пред да започне преносот на корисничка информација, со цел да се обезбеди сигурен пренос. На пример, брз периферен уред испраќа податоци до микропроцесорот, но микропроцесорот извршува програма од поголем приоритет, па може да дојде до губење на корисничката информација. Поради тоа, периферниот уред треба да побара дозвола за испраќање податоци или ако не е возможно тоа, испратените податоци привремено да се сместат во некој мемориски модул и да се причека микропроцесорот.

Микропроцесорот и сите интегрирани кола кои се поврзуваат со него мора да ги исполнуваат условите за **TTL компатибилност** односно логичките нивоа за влез и излез да бидат усогласени. На пример, ако побудниот сигнал е премногу мал, нема да дојде до активирање на влезниот пин, а ако е преголем може да дојде до оштетување на самото интегрираното коло.

4.2. Поврзување со RAM и ROM мемории

На слика 4.1. е прикажано поврзувањето на ROM и RAM мемориите со микропроцесорот.



Слика 4.1. Поврзување на микропроцесор со RAM и ROM мемории

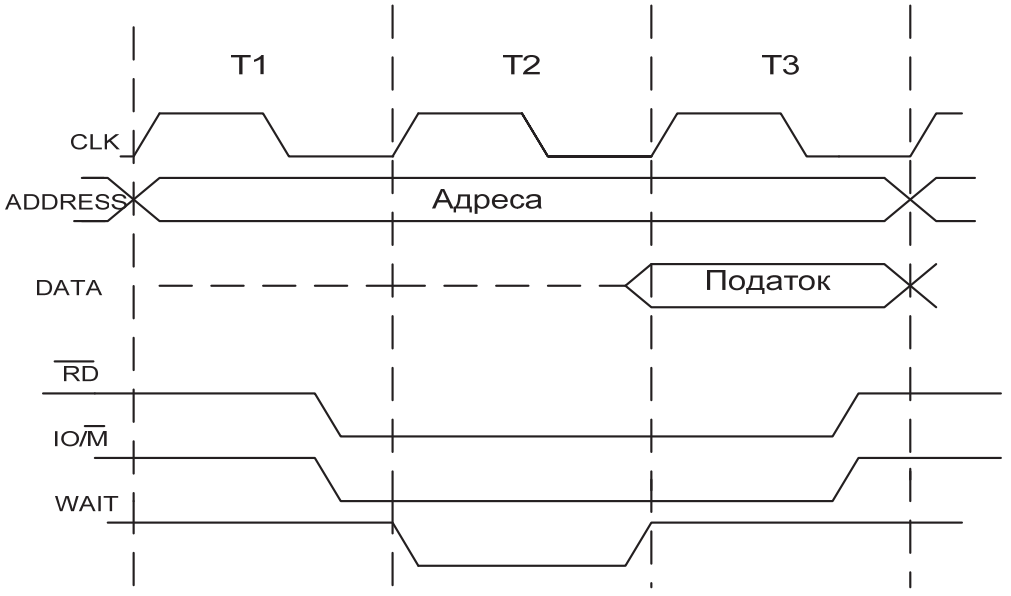
Пиновите на мемориските чипови се поврзани со пиновите на микропроцесорот прку водовите на матичната плоча. Да се потсетиме водовите на матичната плоча се нарекуваат магистрали и истите беа поделени во три групи: адресни, податочни и контролни.

Адресните линии се секогаш влезни за мемориите, а излезни за микропроцесорот. Адресата претставува уникатна бинарна комбинација за избор на една локација од RAM или ROM мемориите. Во примеров капацитетот на RAM меморијата изнесува 2KB за што се потребни 11 адресни линии, од A_0 до A_{10} (2^{11}=2KB) и овие адресни битови ги формираат адресите од 000H до 7FFH. ROM меморијата има капацитет 512B за чие адресирање се потребни 9 адресни линии, од A0 до A8. Адресниот простор на ROM меморијата се простира од 000H (000000000B) до 1FFH (111111111B). Дали микропроцесорот ќе чита од RAM или ROM меморијата зависи од најзначајната адресна линија A_{12}. Оваа линија ги активира пиновите $\overline{CS}$ преку кои се пристапува до една од двете мемории. Со изборот на еден чип од множеството на чипови на ROM или RAM мемориите ќе се запознаеме подоцна преку постапката на адресно декодирање. Податочните линии се двонасочни, влезни или излезни, во зависност од тоа дали читаме или пишуваме од меморијата. Контролните сигнали IO/$\overline{M}$, $\overline{RD}$ и $\overline{WR}$ се излезни за микропроцесорот, а $\overline{WAIT}$ е излезен сигнал за меморијата.

На слика 4.2. е прикажан временски дијаграм за операцијата читање податоци од меморија, при што имаме еден такт состојба на чекање. Микропроцесорот работи во синхрон мод на работа. Синхронизацијата е многу битен процес во преносот на податоци. Синхронизацијата се остварува преку дигитскиот такт на микропроцесорот - сигналот clock. Периодата на дигитскиот такт е најмал временски интервал што го препознава микропроцесорот. За да се прочита еден бајт од меморијата, потребни се три дигитски такта. Гледаме дека растечките и опаѓачките рабови на сигналите не се вертикални, бидејќи никој електричен сигнал не може да ја промени состојбата од логичка нула на единица моментално.

Во почетокот на првиот дигитски такт T_1 микропроцесорот ја испраќа соодветната адреса до меморијата преку адресните линии. Адресата е комбинација од повеќе битови, но не се прикажува нивната состојба, туку се користат две линии кои се вкрстуваат кога ќе се смени адресата. Исто се прикажани и промените на податочните линии, а за сите останати сигнали се користи по една линија. По адресата, микропроцесорот испраќа два контролни сигнала IO/$\overline{M}$ и $\overline{RD}$. Меморијата ги добива овие сигнали уште за време на првиот такт. Но, таа не е во состојба веднаш да го изврши поставеното барање. На меморијата ѝ треба повеќе време за да се пребара, да го пронајде и да го испорача бараниот податок. Значи, оваа работа не може да се изврши за време од еден такт, па поради тоа се активира сигналот $\overline{WAIT}$, што значи

состојба на чекање. Дури во третиот такт ќе бидат испорачани податоците од меморијата во микропроцесорот преку податочните линии, контролните сигнали ќе се деактивираат, а адресните линии ќе бидат ослободени.



Слика 4.2. Временски дијаграм за операција читање од меморија

Синхрониот пренос има голема предност поради неговата голема едноставност и прецизност во работењето. Но, тој има и големи недостатоци што се однесуваат, пред сè, на брзината на работа. На пример, времетраењето на секој машински циклус е изразено во мултипли (цели броеви) од дигитски такта. Иако назначената операција може да биде извршена за време од 3.1 такта, за неа ќе бидат употребени 4 такта, бидејќи делењето на тактовите е строго забрането. Уште полошо е што одредени подобрувања во технологијата не можат да бидат искористени ако тие не се вклопат во воспоставениот тајминг на работа. Неколку години подоцна можат да бидат произведени мемориски чипови кои наместо 40ns ќе имаат време на пристап 20ns. Тоа автоматски би требало да значи елиминирање на тактот за чекање, но ова нема да се случи, бидејќи микропроцесорот и останатите уреди очекуваат операцијата читање од мемориска локација да трае три такта и тоа не може лесно да се измени.

4.3.Виртуелна меморија

Со развојот на софтверот програмите станувале сè пообемни и главната RAM меморија била премала за да обезбеди ефективно работење на системот. Со воведување на поимот виртуелна меморија се врши проширување на главната (примарна) меморија преку надворешната (секундарна) меморија.

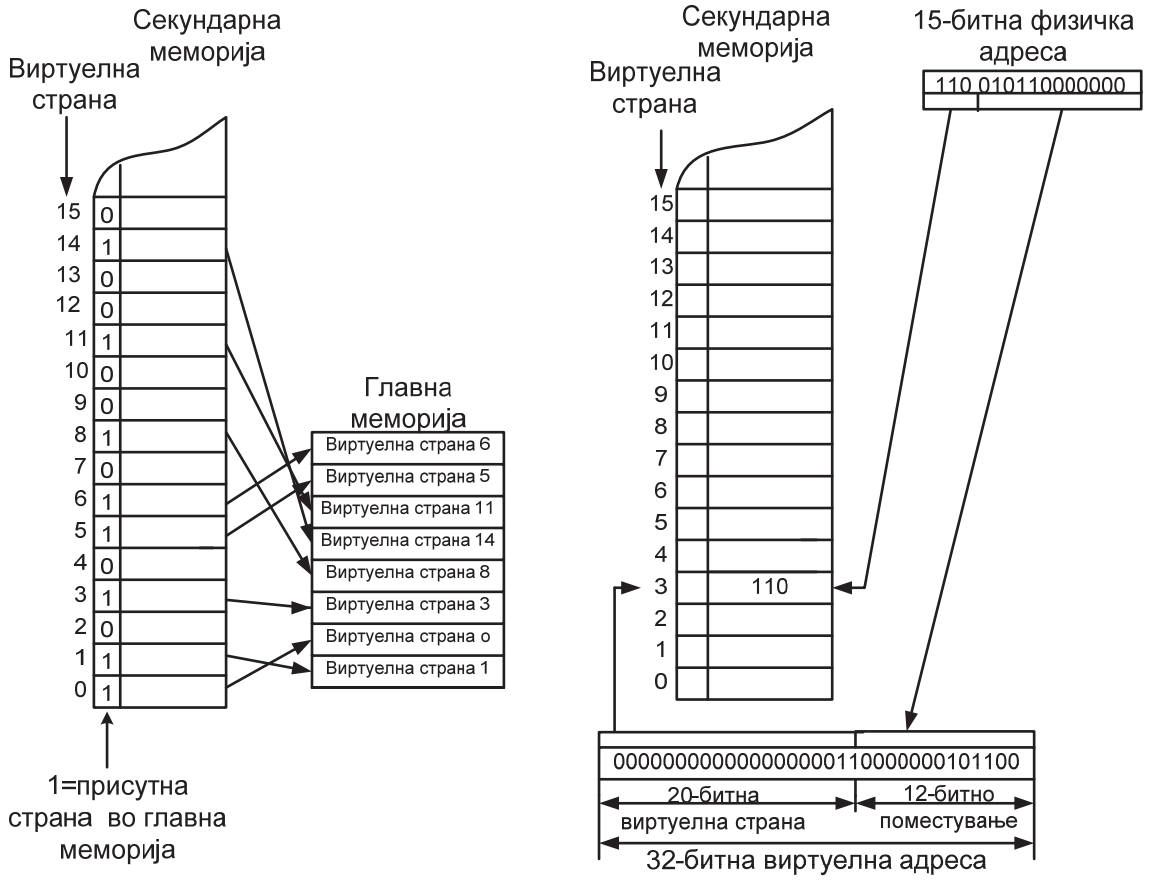
Програмата што треба да се изврши во микропроцесорот се дели на повеќе блокови и секој блок се извршува независно од другите. Откако ќе се изврши првиот блок од програмата, тој се враќа од главната меморија во секударната, а од секундарната во главната меморија се префрла следниот блок, па по него третиот и така натака додека не се изврши целата програма. Во почетокот делењето на програмата на вакви блокови било задача на програмерот што многу ја комплицирало работата. Денес делењето на програмите на блокови и нивното пренесување од еден до друг мемориски простор се врши автоматски преку специјален контролен чип наречен Memory Menagment Unit (MMU). Сè до појавата на виртуелната меморија, адресниот простор бил или поголем или еднаков со меморискиот простор. Со појавата на вирутелната меморија ова се менува, односно адресниот простор станува помал од меморискиот. Да се потсетиме дека адресниот простор зависи од бројот на битови што се содржат во адресното поле или поточно од широчината на адресните магистрали по кои ќе се пренесат овие битови. Меморискиот простор зависи директно од бројот и капацитетот на вградените мемориските чипови.

На сликата 4.3. се претставени главната и секударната меморија и преминот од вирутелен во физички мемориски простор. Од слика 4.3.а. гледаме дека главната меморија е поделена на 8 рамки, при што секоја рамка може да собере 4КВ=2^{12}, а целата главна меморија е 2^{15}=32КВ=8·4КВ (4К-големина на рамка, 8-број на рамки). Секундарната или виртуелната меморија е поделена на страни со големина 4КВ, а целата вирутелна меморија е голема 4GB = 2^{32}.

Значењето на адресните битови е прикажано на слика 4.3.б. Адресите на мемориските локации во главната меморија се викаат физички адреси и тие се состојат од 15 бита. Адресите на локациите во хард-дискот се викаат виртуелни адреси и содржат 32 бита. Во зависност од потребите на микропроцесорот, од виртуелната меморија се земаат саканите страни и се стават во рамките на главната меморија. Да нагласиме дека рамките и страните се со иста големина, со тоа што рамките се фиксни, а се менуваат само страните во нив. Виртуелната адреса е адреса што ја задава програмерот и таа се чува во влезниот регистар на MMU чипот, а физичката адреса се чува во излезниот регистар и според неа микропроцесорот го наоѓа саканиот податок во RAM-от. Првите 20 бита од виртуелната адреса го даваат бројот на страната во виртуелната меморија. Со 20 бита може да се направат 2^{20} различни комбинации, а тоа е, всушност, бројот на страни што може да ги содржи секундарната меморија.

Последните 12 бита од виртуелната адреса служат за идентификација на мемориската локација во рамките на самата избрана страна. Кај физичката адреса значењето на последните 12 бита е исто како и на последните 12 бита од виртуелната адреса, а првите 3 бита ни покажуваат во која рамка од

главната меморија е сместена избраната страна. Со 3 бита може да се направат 2^3=8 комбинации, што одговара на бројот на рамки во главната меморија. Во MMU чипот постои посебна табела на страни во која се чуваат два вида информации: дали соодветната страна е сместена во некоја рамка од главната меморија (0-не,1-да) и редниот број на рамката.



а. Премин од главна во секундарна меморија

б. Генерирање виртуелна адреса од физичка адреса

Слика 4.3.

Постојат најразлични алгоритми за внесување нови страни во главната меморија. Ако сите рамки во главната меморија се полни со страни, се поставува прашањето која страна да се извади за да се стави нова на нејзино место. Најдобро би било да се знае какви побарувања на страни ќе има микропроцесорот во блиска иднина, па тие што ќе бидат најрано побарани, да се стават во рамката, а оние што ќе бидат најдоцна побарани, да се отстранат за да се ослободи место за нови страни. Но, вакви претпоставки е многу тешко да се направат. Доста често користен алгоритам е LRU (Least Recently Used), според кој, од главната меморија се вади онаа страна што била неодамна користена. Кај овој метод се јавуваат одредени аномалии и една таква е прикажана на табелата 4.1.

Виртуелна страна 7	Виртуелна страна 7	Виртуелна страна 7
Виртуелна страна 6	Виртуелна страна 6	Виртуелна страна 6
Виртуелна страна 5	Виртуелна страна 5	Виртуелна страна 5
Виртуелна страна 4	Виртуелна страна 4	Виртуелна страна 4
Виртуелна страна 3	Виртуелна страна 3	Виртуелна страна 3
Виртуелна страна 2	Виртуелна страна 2	Виртуелна страна 2
Виртуелна страна 1	Виртуелна страна 1	Виртуелна страна 0
Виртуелна страна 0	Виртуелна страна 8	Виртуелна страна 8
а	б	в

Табела 4.1. LRU алгоритам за замена на страни

Имено, микропроцесорот има потреба од внесување на првите осум страни од виртуелната меморија (0-7). Потоа треба да се внесе деветата страната со реден број 8. Но, за неа нема место. Неодамна користена страна е онаа со реден број 0 и таа се вади за да на нејзино место дојде страната со број 8. Апсурд е што во наредниот чекор микропроцесорот пак ќе ја побара страната број 0 и за да ја внесеме треба да ја извадиме страната број 1. Ова значи дека во претходниот чекор замената на страната 0 било на некој начин грешен потег, но тоа ни го налага LRU алгоритамот. Друг користен алгоритам е FIFO (First In First Out), или во превод „кој прв влегол прв и ќе излезе“. Овде критериум за заменување на страните не е најдолгото време поминато од последната употреба, туку најдолгото време поминато од внесувањето на таа страна во главната меморија.

За програмерот многу битен фактор е брзината со која ќе се заменуваат страните во главната меморија. Исто така, битен параметар е големината на страните, односно рамките. Тие не треба да се премногу мали ни премногу големи. Ако се премногу големи, тогаш во RAM меморијата ќе бидат сместени мал број страни и тие треба да бидат почесто заменувани за да бидат задоволени барањата на микропроцесорот. Доколку страните се премали, тогаш се зголемува бројот на трансфери што треба да се извршат за да се пополни целата RAM меморија, но при поголем број рамки во RAM меморијата ќе имаме и поголем избор за вадење на некоја стара страна и нејзино заменување со нова. Во секој случај, треба да се најде компромисно решение, имајќи предвид дека големината на страната и нивниот број во RAM меморијата се во обратно пропорционален однос.

4.4. Организација на кеш меморија

Кеш меморијата посредува меѓу микропроцесорот и главната меморија. Според намената кеш мемориите можат да бидат инструкциски и податочни. Во инструкциската кеш меморија се чуваат инструкциите кои се однапред донесени од главната меморија и чекаат да бидат повикани за извршување од страна на микропроцесорот. Податочната кеш меморија ги чува податоците кои микропроцесорот почесто ги повикува, а за истите нема место во општите регистри на микропроцесорот.

Во пософистицираните мемориски организации постојат неколку нивоа на кеш меморија. Постојат три нивоа на кеш меморија. Кеш меморијата од прво ниво се наоѓа во самиот микропроцесорски чип, кеш меморијата од второ ниво се наоѓа во самото пакување на микропроцесорот и кеш меморијата од трето ниво се наоѓа на матичната плоча.

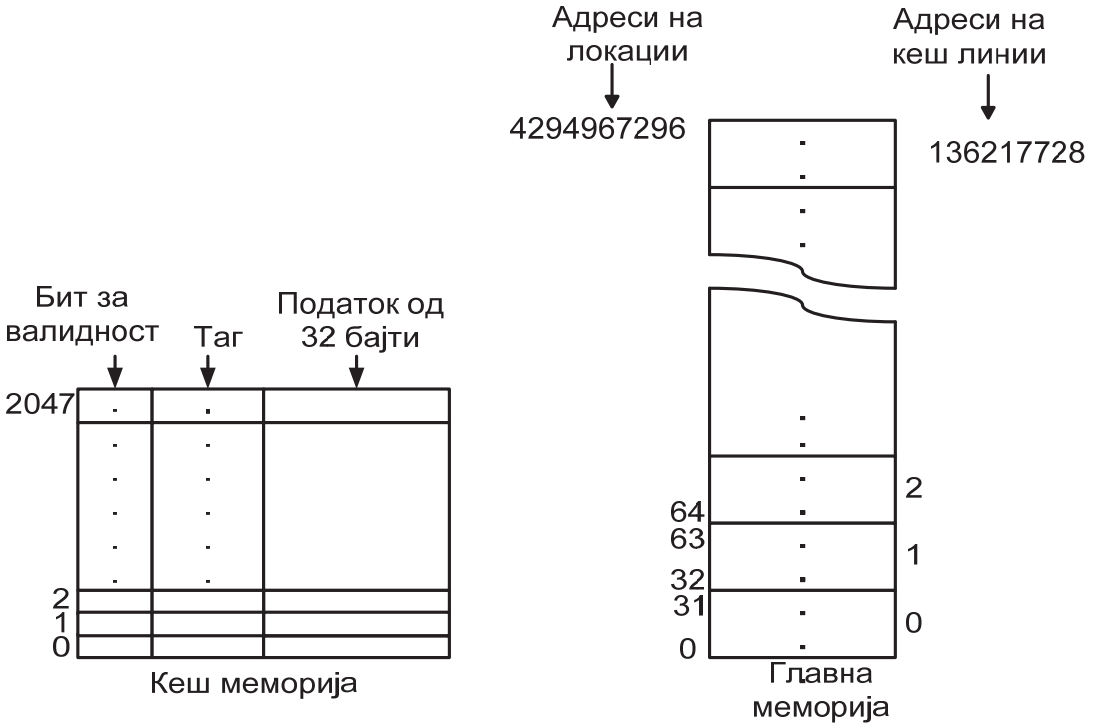
Имплементацијата на сите видови кеш меморија се врши на следниов начин. Главната меморија е поделена на блокови со константна големина, наречени кеш линии. Кеш линиите се составени од 4 до 64 последователни бајти. Ако линиите се составени од 32 бајти тогаш нултата линија ќе ги содржи бајтите со адреси од 0 до 31 , првата линија бајтите со адреси од 32 до 63, втората од 64 до 95 итн. Некои од овие линии се наоѓаат во кеш меморијата. Пред да пристапи микропроцесорот до меморијата, контролерот на кеш меморијата проверува дали бараниот податок е во кеш меморијата. Доколку саканиот податок е во кеш меморијата микропроцесорот нема да пристапи до главната меморија. Во тој случај адресната и податочната магистрала ќе останат слободни и истите можат да се искористат за некоја друга операција како на пример директен пренос на податоци од меморијата без посредство на микропроцесорот. Но доколку саканиот податок не се наоѓа во кеш меморијата тогаш некоја од претходно внесените кеш линии треба да се извади и на нејзино место треба да се запише бараната линија.

На слика 4.4. е прикажана главната меморија со капацитет од 4GB, која е поделена на блокови од по 32 бајти. На истата слика е прикажана кеш меморијата со 2048 влезови. Секој влез од кеш меморијата може да собере една кеш линија од по 32 бајти, што значи капацитетот на кеш меморија ќе изнесува 2К·32В=64КВ.

Секој влез од кеш меморијата се состои од три дела.

1. Битот за валидност ни покажува дали влезот на кеш меморијата е наполнет после стартувањето на системот или не. Кога системот ќе се стартува сите влезови во кеш меморијата се маркирани како невалидни.

2. Таканаречената поле таг (англиски tag=означување) го содржи редниот број на кеш линијата од главната меморија од каде што е земен податокот.
3. Податочното поле го содржи податокот пренесен од кеш линијата во кеш влезот.



Слика 4.4. Приказ на главната меморија (поделена на блокови) и кеш меморијата со 2048 влезови

На слика 4.5. прикажана е виртуената адреса со чија помош кеш контролерот го пронаоѓа саканиот податок. Гледаме дека виртуелната адреса е поделена на три дела, исто како и влезовите на кеш меморијата.

1. Полето таг одговара на истоименото поле од кеш влезот
2. Полето за линија го дава редниот број на кеш влезот во кој се наоѓа саканиот податок
3. Полето за бајт го дава редниот број на бајтот во избраниот кеш влез.

Слика 4.5. Виртуелна адреса за пронаоѓање на податок од кеш меморија

Големината на секое поле од виртуелната адреса се одредува на следниов начин. Бидејќи главната меморија е со капацитет 4GB виртуената адреса ќе биде 32 битна. Бројот на битови во полето за бајт зависи од големината на кеш линијата што секако е еднаква со големината на кеш влезовите. На пример ако големината на кеш линијата изнесува 32 бајти тогаш полето за бајт ќе биде големо пет битови (2^5=32). Големината на полето за линија зависи од тоа колку влезови содржи кеш меморијата. Во нашиот пример тој број изнесуваше 2048=2^{11} што значи полето за линија ќе содржи 11 битови. Останатите 16 битови од 32 битната виртуелна адреса го претставуваат полето на тагот.

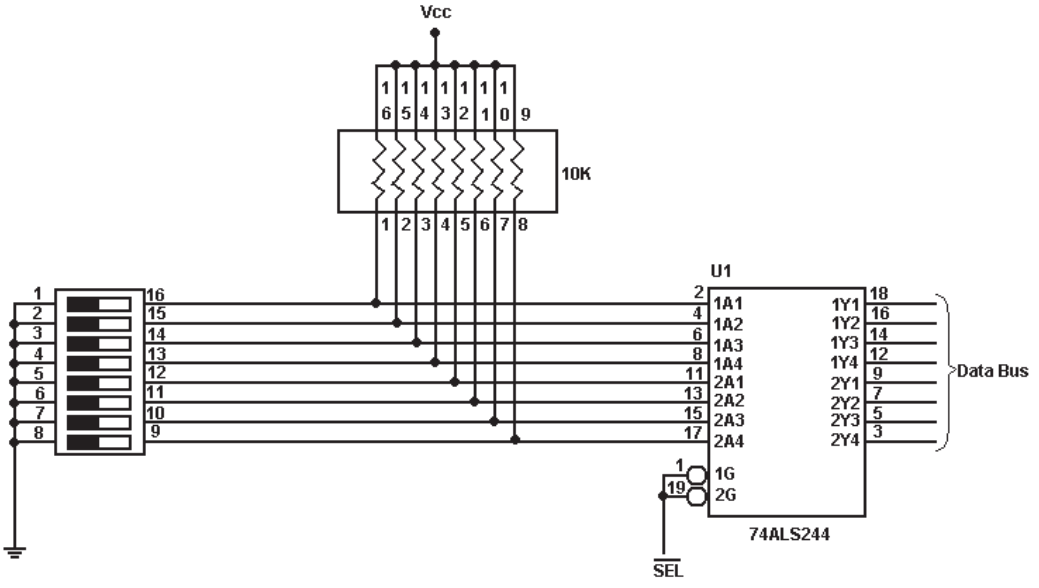
Кога микропроцесорот генерира адреса, се издвојува полето за линија и истото се користи за пронаоѓање на еден од 2048-те кеш влезови. Ако битот за валидност е еднаков на единица тогаш се споредуваат полето за таг од виртуелната адреса и полето за таг од кеш влезот. Ако тие се поклопуваат тогаш велиме дека кеш меморијата го содржи саканиот податок и ваквата ситуација се нарекува кеш погодок. Саканиот податок ќе се прочита од кеш меморијата и ќе се одбегне преносот од главната меморија. Полето за бајт ни овозможува да издвоиме само еден бајт без да ја пренесуваме целата содржина на кеш влезот. Ако битот за валидност е еднаков на нула или полињата за таг не се совпаѓаат тогаш велиме дека саканиот податок не се наоѓа во кеш меморијата. Ваквата ситуација е позната под името кеш промашување. Во тој случај треба да се оди до главната меморија, од таму да се земе саканата кеш линија и да се внесе во кеш меморијата.

И покрај ваквата малку долга постапка на пребарување на кеш меморијата, таа има многу мало време на пристап кое е од ред на величина наносекуда. Главен проблем претставуваат кеш промашувањата бидејќи тогаш треба да се извршат пребарувања на две мемории, кеш меморијата и главната меморија.

4.5. Поврзување со влезно/излезни уреди

Влезно излезните уреди служат за комуникација на компјутерот со надворешниот свет. Тие можат да бидат влезни (тастатура, глушец) , излезни (монитор, печатач) или влезно излезни (CD дискот). Влезно излезните уреди се познати и под името периферни уреди. Покрај за комуникација со надворешниот свет некои од нив се користат и за чување на податоци. Незгодно е што за пренос на податоци влезно излезните уреди ги користат адресната и податочната магистрала исто како и мемориските уреди. За да подобро го разбереме преносот на податоци меѓу влезно излезните уреди и микропроцесорот ќе разгледаме два многу едноставни примери.

На слика 4.6. е даден пример за поврзување со осум паралелни прекинувачи. Прекинувачите се поврзани со податочната магистрала преку бафер 74ALS244. На влез од баферот се приклучени прекинувачите, а на излез осумте линии од податочната магистрала. Кога некој од прекинувачите е затворен по податочната линија се испраќа логичка нула бидејќи прекинувачот го врзува влезниот пин на баферот на маса. Кога прекинувачот е отворен тогаш се испраќа логичка единица. Во првата тема, кога зборувавме за основни логички кола истакнавме дека баферот не врши обработка на сигналите туку тој врши харверско поврзување на периферните уреди со податочната магистрала.

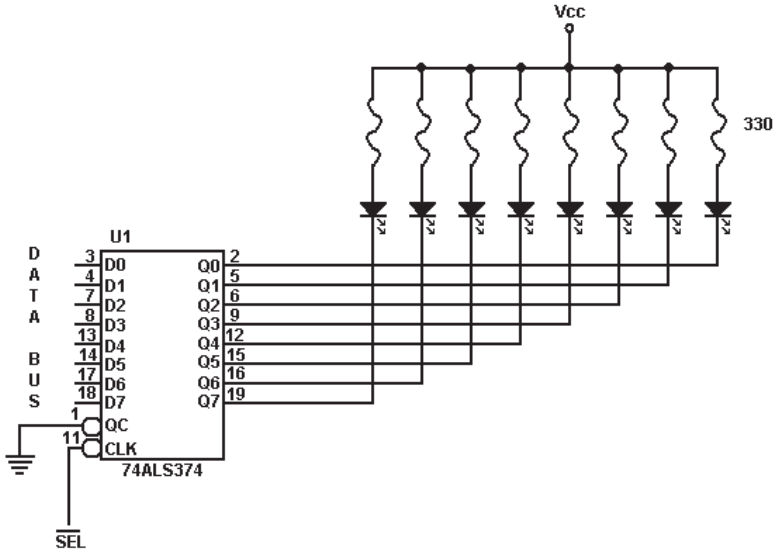


Слика 4.6. Поврзување на микропроцесор со 8 паралелни прекинувачи

Дали баферот ќе ги пропушта битовите кои ги испраќаат прекинувачите зависи од состојбата на сигналот $\overline{SEL}$. Ако овој сигнал е на високо ниво баферот ќе биде во состојба на висока импенданса, а ако е на ниско ниво тогаш ќе се активитаат двата пина за селекција на баферот 1G и 2G (G – gate). Вредноста на сигналот $\overline{SEL}$ се добива со декодирање на интерфејс адресата на периферниот уред за што ќе стане збор подоцна.

На слика 4.7. е прикажан пример за поврзување со излезен уред. Осумте паралелно поставени лед диоди светат кога микропроцесорот ќе испрати логичка 0. Меѓу податочната магистрала и LED диодите е поставен специјален уред, таканаречен леч (latch) слично како баферот кај влезниот интерфејс. Лечот е уред составен од тактирачки флип-флопови чија задача е да го продолжат времетраењето на побудниот сигнал. Сигналите што ги испраќа микропроцесорот траат само 1 μs. Ова време не е доволно за да се побудат лед диодите. Затоа веднаш после пристигнувањето на сигналот на влез од тактирачкиот флип флоп дигитскиот такт се спушта на ниско ниво, се

онеспособуваат влезовите, но испратениот сигнал останува на излезот од флип флопот.

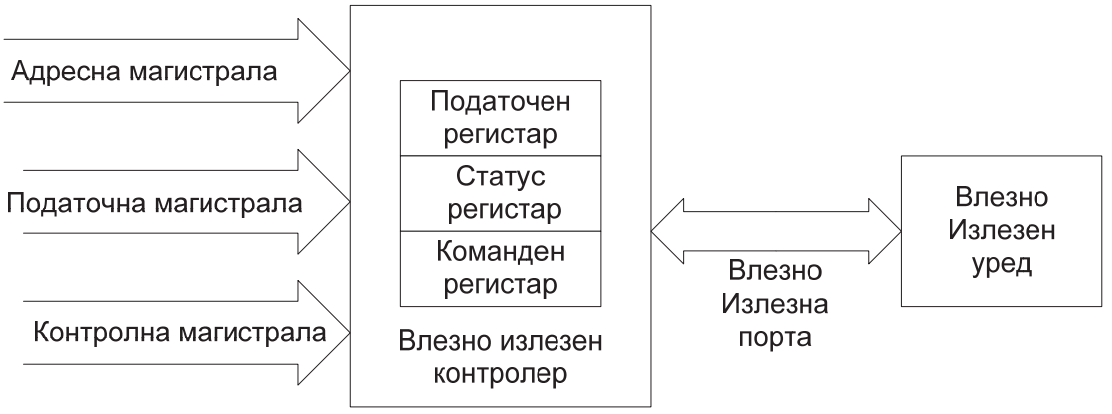


Слика 4.7. Поврзување на микропроцесор со 8 паралелни лед диоди

Од овие два едноставни примери може да се види дека влезно излезните уреди не се директно поврзани на системската магистрала. Колата кои посредуваат во комуникацијата меѓу периферните уреди и системската магистрала се нарекуваат влезно/излезни контролери или интерфејс контролери. Постојат повеќе добри причини за употреба на овие контролери. Пред сè тие му ја олеснуваат управувачката функција на микропроцесорот. Микропроцесорот ќе биде послободен и побрзо ќе ги извршува корисничките програми. Интерфејс контролерите генерираат едноставни контролни сигнали за управување со периферните уреди. За испраќање на сигналите по системската магистрала се троши многу малку енергија. Ако не се користат интерфејс контролери и истите не ги засилат испратените сигнали тогаш каблите за поврзување на периферните уреди со компјутерот би биле долги само неколку сантиметри.

Влезно/излезните контролери содржат три видови на внатрешни регистри: податочни, командни и статусни. Регистрите на влезно излезниот контролер се прикажани на слика 4.8. Податочните регистри ги чуваат податоците кои се прочитани или треба да се запишат во некој од периферните уреди. Статус регистарот содржи информација за моменталната состојба на периферните уреди. Значењето на овие регистри ќе го објасниме преку пример за поврзување на компјутерот со печатач. Пред да испрати микропроцесорот карактер до печатачот тој го проверува статус регистарот на контролерот кој управува со печатачот. Четвртиот бит од статус регистарот дава информација за тоа дали печатачот е приклучен или не за компјутерот, седмиот бит дава информација за зафатеноста на печатачот и петтот бит дали има или нема хартија за печатење. Податочниот регистар го чува карактерот кој треба да

биде испечатен, а преку командниот регистар микропроцесорот ја задава операцијата која треба да ја изврши печатачот.



Слика 4.8. Блок дијаграм на влезно излезен контролер

Сите регистри во влезно/излезните контролери имаат своја адреса таканаречена интерфејс адреса. Интерфејс адресите се испраќаат преку адресните пинови на микропроцесорот, исто како и мемориските адреси. Се поставува прашањето, кога адресните битови ќе претставуваат мемориска адреса, а кога интерфејс адреса.

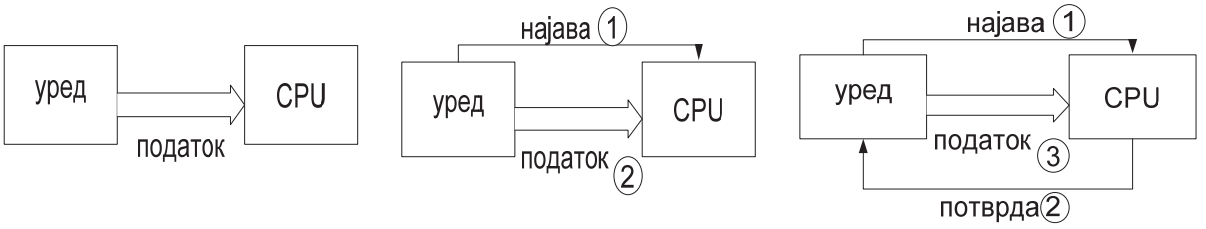
Адресирањето на периферните уреди може да се реализира на два начина:

- изолиран влез/излез
- мемориски пресликан влез/излез

Кај изолираниот влез/излез се користи сигналот IO/$\overline{M}$ (input output/$\overline{memory}$) за одвојување на меморијата од влезно-излезните уреди.

Кај мемориски пресликаниот влез/излез сигналот IO/$\overline{M}$ не се користи. Кај овој пристап за одвојување на меморијата од периферните уреди се користи адресен бит, обично најзначајниот. Ако имаме 16 адресни линии, тогаш кога битот A_{15}=0 адресата ќе биде мемориска. Ако A_{15}=1 тогаш адресата од микропроцесорот ќе биде интерфејс адреса за периферен уред. Предности на мемориски пресликаниот влез/излез е што за пренос до или од периферните уреди се користат истите инструкции за пренос како и за меморијата. Ова претставува значително проширување на инструкциското множество за разлика од изолираниот влез/излез, каде што за комуникација меѓу микропроцесорот и периферниот уред може да се користат само специјални инструкции за пренос. Но, пресликаниот влез/излез има и свои недостатоци. Тука е, пред сè, трошењето адресни битови за влезно-излезните уреди. Автоматски мемориският простор ќе се намали. Друг недостаток е што применетите инструкции траат подолго време поради подолгите инструкциски кодови.

Пред да започне преносот на податоци меѓу микропроцесорот и некој периферен уред, тие си испраќаат контролни сигнали, со цел да се обезбеди сигурен пренос. Во зависност од видот на контролните сигнали, разликуваме **четири режими на паралелен пренос**: едноставен, стробиран, режим на поздрав и двоен поздрав. На сликата 4.9. дадени се контролните сигнали за првите три режими и нивните блок шеми. Со кругчиња е обележан редоследот на испраќање на сигналите.



А) едноставен режим Б) стробиран режим В) режим на поздрав

Слика 4.9. Блок дијаграми на режимите на паралелен пренос

Уредот кој испраќа податоци може да испрати сигнал за најава (англиски-strobe), а уредот кој прима податоци може да испрати потврда дека ја примил најавата (англиски-acknowledge). Кај режимот на едноставен пренос микропроцесорот и влезно излезните уреди не разменуваат контролни сигнали пред да започне преносот на корисничка информација. Овој вид на пренос е најнесигурен бидејќи може да дојде до лесно губење на корисничката информација. Едноставниот режим е најнесигурен, но е најекономичен бидејќи се штеди во хардвер. Кај стробираниот режим уредот кој испраќа податоци го најавува преносот преку активирање на сигналот за најава (strobe). Стробираниот режим се користи во комуникацијата меѓу микропроцесорот и тастатурата. Кај режимот на поздрав (handshaking) уредите си разменуваат контролни сигнали, сигнал за најава (strobe) и сигнал за потврда (acknowledge). Режимот на двоен поздрав е најсигурен, но е најскап, бидејќи четирите бакарни вода ќе се употребат за испраќање контролни сигнали. Исто така, колку повеќе сигнали постојат, толку подолго ќе трае преносот.

4.6. Практични влезно излезни порти

Најчесто користени компјутерски порти се: паралелната порта, сериската порта и USB портите. За да можеме да поврзуваме периферни уреди важно е да го познаваме распоредот на пиновите на портите и значењето на нивните сигнали.

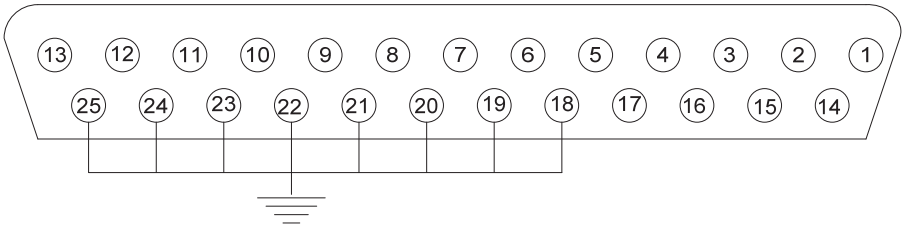
Паралелната порта се користи за поврзување на печатачи, а подоцна ќе видиме дека се користи и при програмирање на микроконтролери. Паралелната порта има најголем број на пинови. Компјутерот може да има три паралелни порти: LPT1, LPT2 и LPT3, но најчести има една порта LPT1.

Паралелната порта се контролира преку три регистри: податочен, команден и статусен. Овие регистри користат три последователни интерфејс адреси. Стандардните хексадецимални адреси на овие регистри се прикажани во табелата 4.2 .

Порта	Податочна порта	Статусна порта	Командна порта
LPT1	378H	379H	37AH
LPT2	278H	279H	27AH
LPT3	3BCH	3BDH	3BEH

Табела 4.2. Стандардни интерфејс адреси за регистрите на паралелната порта

Кабелот за поврзување на компјутерот со печатачот има два конектори DB25 и CENT39. Конекторот DB25 претставува излез на компјутерот, а конекторот CENT39 влез за печатачот. Распоредот на нивните пинови е даден на слика 4.10..



Слика 4.10. Паралелна порта DB25

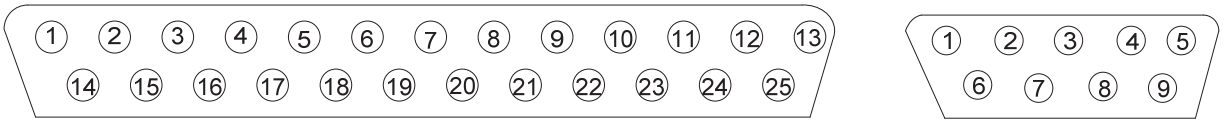
Во табелата 4.3. се прикажани сигналите за секој од пиновите.

број на пин		сигнал	функција
DB25	CENT39		
1	1	$\overline{Data\ strobe}$	Микропроцесорот испраќа сигнал за најава
2	2	Data 0	Прв бит од податокот
3	3	Data1	Втор бит од податокот
4	4	Data2	Трет бит од податокот
5	5	Data3	Четврти бит од податокот
6	6	Data4	Петти бит од податокот
7	7	Data5	Шести бит од податокот
8	8	Data6	Седми бит од податокот
9	9	Data7	Осми бит од податокот
10	10	$\overline{Ack}$	Печатачот испраќа сигнал за потврда дака го примил податокот
11	11	Busy	Печатачот е зафатен
12	12	Paper empty	Нама хартија
13	13	Select	Печатачот е приклучен
14	14	Afd	Поместување на печатачот за еден ред подоле

15	32	$\overline{Error}$	Грешка при печатењето
16	-	Int	Печатачот се иницира да го испечати карактерот , а потоа се ресетира
17	31	Select in	Избран е печатач
18-25	19-30	Заземјување	/
-	17	Заземјување	/
-	16	Заземјување	/
-	33	заземјување	/

Табела 4.3. Функциски опис на пиновите на паралелната порта DB25

Серискатa порта на компјутерот (COM) се користи за поврзување на надворешни модеми, глувче, плотер и др. Исто така преку неа може да се изгради мала локална мрежа која користи оперативен систем Windows. Серискатa порта користи два вида на конектори: D-SUB (25 пински) и D-SUB (9 пински). Обликот на конекторот и распоредот на пиновите се прикажани на слика 4.11.



DSUB25 DSUB9

Слика 4.11. Сериски порти

Во табелата 4.4. објаснето е значењето на поважните сигнали од овие два конектори.

D-SUB 25	D-SUB 9	Сигнал	Функција
Пин 2	Пин 2	TD-Transmit Data	Сериски влез на пдатоци
Пин3	Пин 3	RD-Receive Data	Сериски излез на податоци
Пин 4	Пин 7	RTS-Request To Send	Иницира спремност на модемот за размена на податоци
Пин 5	Пин 8	CTS-Clear To Send	Се активира после приемот на сигналот CD
Пин 6	Пин 6	DSR-Data Set Ready	Иницира спремност за работа
Пин 7	Пин 5	SG-Signal Ground	Маса
Пин 8	Пин 1	CD-Carrier Detect	Врската е воспоставена
Пин 20	Пин 4	DTR-Data Terminal Ready	После приемот на сигналот DSR иницира спремност на DTR
Пин 22	Пин 9	RI-Ring Indicator	Детектиран сигнал на телефонската линија

Табела 4.4. Функциски опис на пиновите на серискитe порти

Протоколот за испраќање на податоци ќе го објасниме преку пример за размена на податоци меѓу два компјутери А и В. Протоколот за пренос можеме да го поделиме во три фази: воспоставување на врска, пренос на податоци и раскинување на врска.

Фазата на воспоставување на врска можеме да ја поделиме во три чекори:

1. Компјутерот А го активира сигналот DTR со што му иницира на модемот А дека сака да пренесува податоци. Потоа го пренесува телефонскиот број на приемникот до модемот В преку пинот TD.
2. Кога модемот А ќе воспостави врска со модемот В, модемот В му го најавува на својот компјутер дојдовниот повик преку активирање на пинот RI. Ако компјутерот В е спремен за прием на податоци тогаш тој ќе го активира сигналот DTR. После добивањето на зелено светло модемот В го активира сигналот DSR дека е спремен да прими податоци од компјутерот А.
3. После приемот на повратната информација модемот А го информира својот компјутер преку активирање на пинот CD. Исто така го активира пинот DSR со што звршува фазата на воспоставување на врска.

За фазата на пренос на податоци се користи режимот на поздравување при што контролните сигнали се разменуваат преку пиновите RTS и CTS. Самите податоци се испраќаат преку пинот TD.

Раскинувањето на врската се врши преку едноставно деактивирање на сигналите RTS (Request To Send) на двете страни, што од друга страна повлекува деактивирање на сигналите CTS.

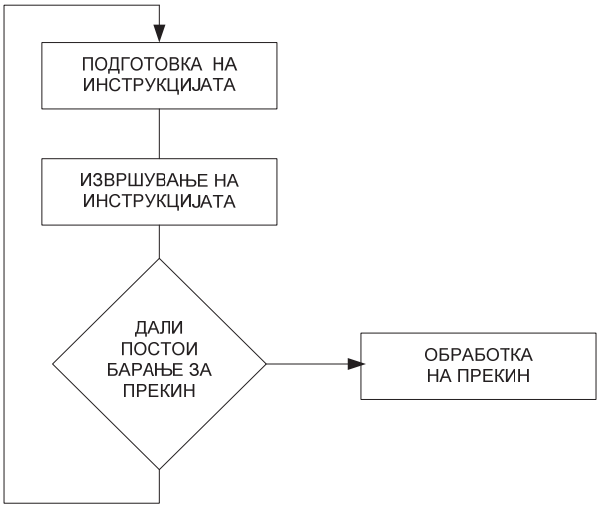
4.7. Системи со прекини (interrupt)

Честопати се случува некој од периферните уреди да побара од микропроцесорот да го прекине извршувањето на тековната програма и да премине кон извршување на потпрограмата за сервисирање на тој периферен уред.Прекините не мора да бидат предизвикани само од периферните уреди. До прекин доаѓа и кога ќе настапи недозволена ситуација во внатрешноста на самиот микропроцесор. Овие прекини се нарекуваат внатрешни прекини. Нив микропроцесорот ги третира како и надворешните прекини. Периферните уреди испраќаат до микропроцесорот посебни барања за прекин. За таа цел се активира специјален пин INTR (interrupt). По извршувањето на секоја инструкција од тековната програма, микропроцесорот проверува дали постои барање за прекин. Ако такво барање постои, тогаш микропроцесорот може да ја прекине тековната работа и да премине кон обработка на прекинот, при што

се активира пинот INTA (Interrupt acknowledge). На ваков начин микропроцесорот го известува периферниот уред дека неговото барање за прекин е прифатено. Ова е прикажано на слика 4.12. Ако прекинот не е итен, тогаш може да биде отфрлен или да биде ставен во состојба на чекање.

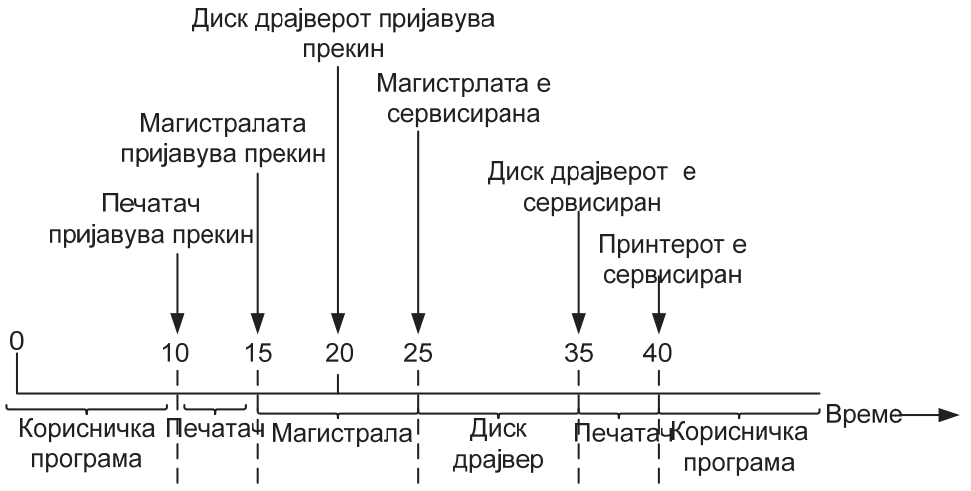
Доколку микропроцесорот го прифати барањето за прекин, тогаш ќе бидат преземени следниве активности:

1. Периферниот уред испраќа таканаречен прекинувачки вектор (код составен од 8 бита) до микропроцесорот заради идентификација.
2. Прекинувачкиот вектор му помага на микропроцесорот полесно да ја пронајде програмата за сервисирање на прекинот (interrupt service routine).
3. Откако прекинот ќе биде обработен, микропроцесорот продолжува да ја извршува тековната програма онаму каде што застанал пред појавата на прекинот.



Слика 4.12. Блок дијаграм за сервисирање на прекин

Се поставува прашањето што ќе се случи доколку истовремено до микропроцесорот пристигнат повеќе барања за прекин од повеќе периферни уреди. За да не дојде до хаос во обработката на прекини, на некои периферни уреди им се дава поголем приоритет во однос на останатите. На сликата 4.13. е прикажан еден таков пример. До микропроцесорот пристигнуваат барања за прекин од 3 уреди. Највисок приоритет има магистралата, потоа доаѓа диск-драјверот и најмал приоритет има печатачот. Микропроцесорот извршува некоја тековна програма кога одеднаш во моментот t=10 се јавува прекин, иницран од страна на печатачот. Микропроцесорот ја прекинува тековната програма и започнува програма за обработка на овој прекин. Оваа програма трае 10 временски единици. Но, во моментот t=15 магистралата доставува барање за прекин. Тогаш микропроцесорот ќе ја прекине потпрограмата на печатачот за да премине на обработка на прекинот од магистралата (таа има приоритет во однос на печатачот).



Слика 4.13. Обработка на прекини со различен приоритет

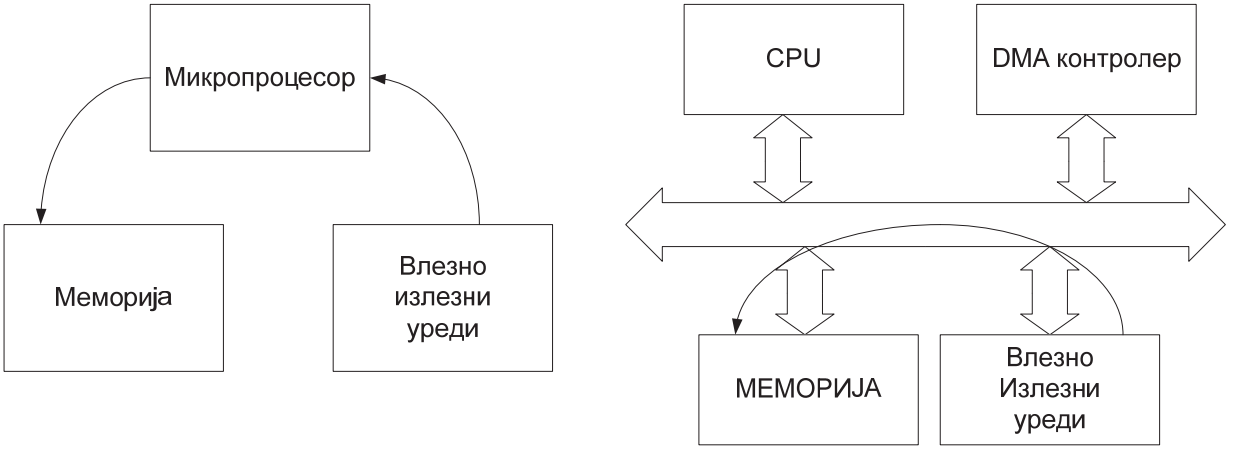
Додека се сервисира магистралата, пристигнува ново трето барање за прекин од страна на диск-драјверот. Микропроцесорот прво ќе ја заврши програмата за обработка на прекинот на магистралата и тоа трае до t=25. Во истиот момент микропроцесорот започнува програма за обработка на прекинот од диск драјверот. Дури откако ќе заврши во t=35 микропроцесорот се враќа да ја доврши програмата за обработка на прекинот од печатачот. Значи, иако печатачот прв достави барање за прекин, тој ќе биде последен обработен, бидејќи печатачот има најмал приоритет. Откако микропроцесорот ќе ги обработи сите примени барања за прекин, тој ќе се врати на тековната програма.

Кај најголем број микропроцесори разликуваме два вида прекини – маскирани и немаскирани прекини. Маскирањето значи забрана за микропроцесорот да ја прекине тековната програма поради обработка на добиениот прекин. Кај маскираните прекини микропроцесорот најпрво испитува дали е ставена маска и во случај да е поставена, микропроцесорот преминува на извршување на следната инструкција без да посвети внимание на добиеното барање за прекин. За поставување маски на прекините постои специјална инструкција SIM (Set Interrupt Mask). За читање на маската за прекини се користи инструкцијата RIM (Read Interrupt Mask).

4.8. DMA пренос

Понекогаш микропроцесорот троши 20% од своето работење на пренос на податоци. Кратенката DMA (Direct Memory Access) всушност значи директен пристап до меморија, без посредство на микропроцесорот. DMA преносот претставува посебен концепт со кој микропроцесорот се ослободува од преносот на податоци меѓу меморијата и влезно излезните уреди. Со овој

концепт не само што не се оптоварува микропроцесорот туку се обезбедува и посигурен пренос односно се намалува губењето на податоци. Ова посебно се однесува на преносот од некој многу брз периферен уред. Концептот DMA е имплементиран преку употреба на специјален DMA контролер. Ова е прикажано на слика 4.14.

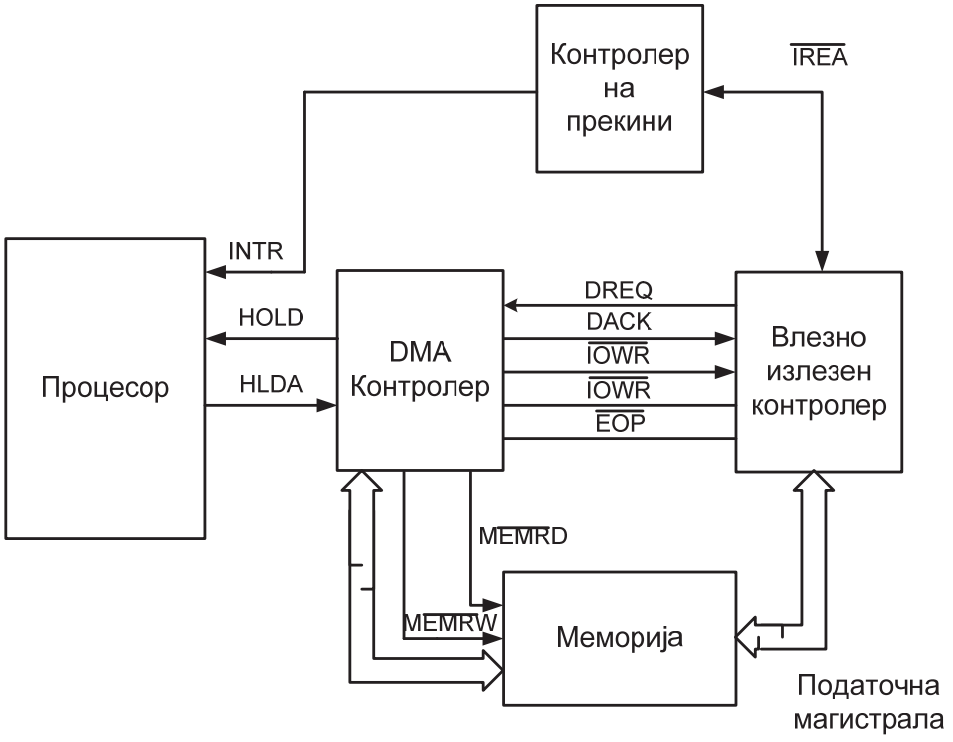


Слика 4.14. Споредба меѓу систем со и без DMA концеот

DMA контролерот претставува роб (slave) за микропроцесорот. Кога треба да се пренесе блок од податоци од некој влезно излезен уред до меморијата и обратно, микропроцесорот ги испраќа: интерфејс адресата на уредот, мемориската адреса, бројот на бајти кои треба да се пренесат и контролен сигнал за видот на пренос (влезен или излезен). Откако ќе ја прими инструкцијата за пренос, DMA контролерот станува господар (master) на магистралите и почнува да генерира контролни сигнали за да го реализира преносот. DMA контролерот врши распределба (арбитража) на магистралите, превземање на мемориската адреса, и генерирање на RD и WR контролни сигнали. После извршениот пренос ги ослободува магистралите, врши обновување на мемориската адреса и бројот на бајти и доколку има уште бајти за пренос ја повторува целата постапка. На крајот од преносот микропроцесорот добива нов прекин после што може да го провери статусот на преносот (успешен или не). Доколку се работи за брз пренос на поголемо количество податоци тогаш DMA контролерот не мора да бара користење на магистралите после секој завршен пренос на еден бајт или збор.

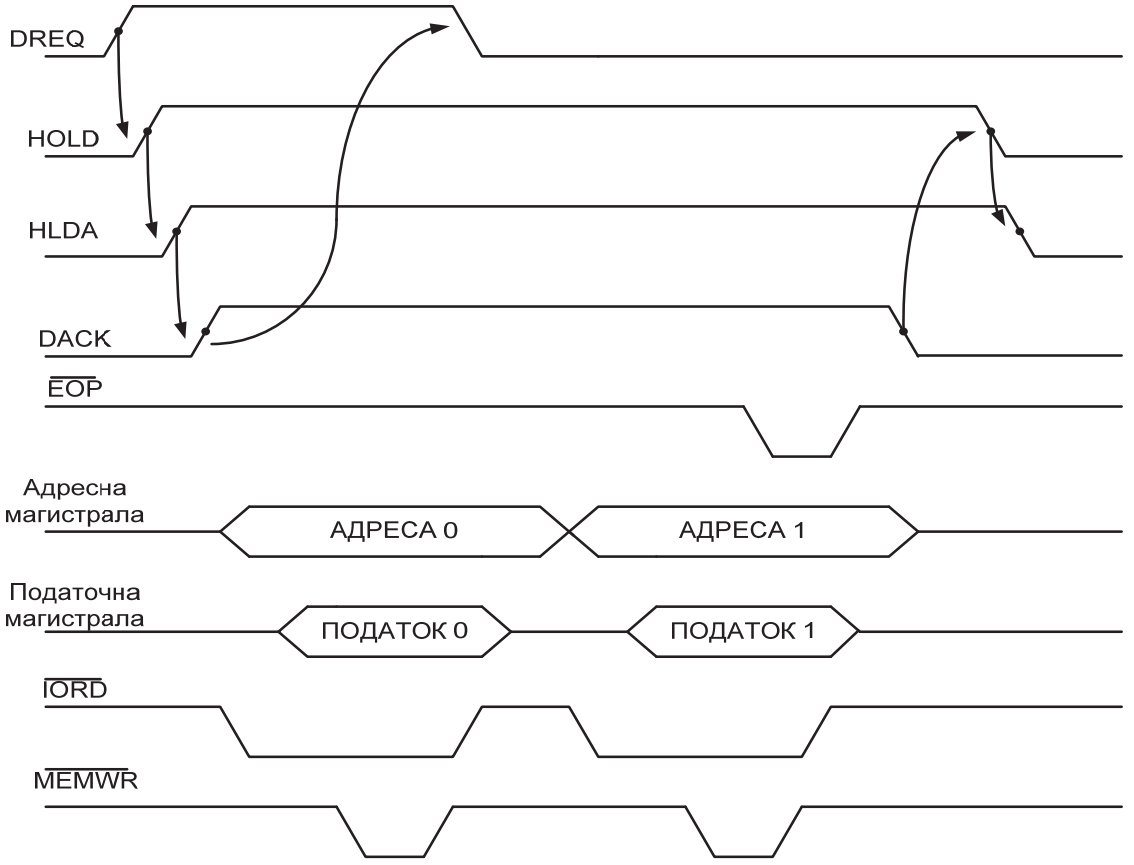
За илустрација на DMA преносот ќе објасниме еден едноставен пример прикажан на слика 4.15. На слика 4.15. е прикажано поврзувањето меѓу сите инволвирани уреди во DMA преносот: микропроцесорот, DMA контролерот, меморијата и влезно/излезниот контролер. До адресната и податочната магистрала имаат пристап микропроцесорот и DMA контролерот, но поради поголема едноставност не е прикажано поврзувањето на микропроцесорот со магистралите. Еден DMA канал се користи за пренос на податоци до само еден влезно/излезен уред. Не може да се случи DMA каналот да го делат повеќе уреди и пренесуваниот податок да пристигне до повеќе уреди.

Претпоставуваме дека се работи за операција читање (влезен уред) и дека треба да се пренесе информација од два збора (два 16-битни податоци).



Слика 4.15. Начин на поврзување на DMA контролерот

На слика 4.16. прикажан е временскиот дијаграм за операцијатата DMA пренос. Кога влезниот уред е спремен за пренос тој испраќа барање до DMA контролерот преку DREQ (DMA request) линијата. После приемот на овој сигнал DMA контролерот го подига на високо ниво HOLD сигналот со што му најавува на микропроцесорот дека сака да изврши пренос на податоци. Микропроцесорот дава дозвола преку активирање на сигналот HLDA (Hold Acknowledge) со што DMA контролерот станува господар на магистралите. Потоа DMA контролерот го испраќа сигналот DACK (DMA Acknowledge) за да го информира влезниот уред дека DMA преносот може да почне. Сигналот DACKпреставува одговор, потврда за испратеното барање DREQ. DMA контролерот ги генерира контролните сигнали за читање од влезниот уред $\overline{IORD}$ и пишување во меморијата $\overline{MEMWR}$. Влезниот уред реагира на контролните сигнали и ги поставува податоците на податочната магистрала. Кога ќе се пренесе податокот вредноста на мемориската адреса се зголемува, бројот на бајтите се намалува и ако тој број не е нула може да започне нов пренос. После завршувањето на преносот на сите бајти DMA контролерот го испраќа сигналот EOP (End Of Process). Исто така доаѓа до гаснење на сигналите DACK и HOLD, после што микропроцесорот повторно станува господар на магистралите, а DMA контролерот роб.



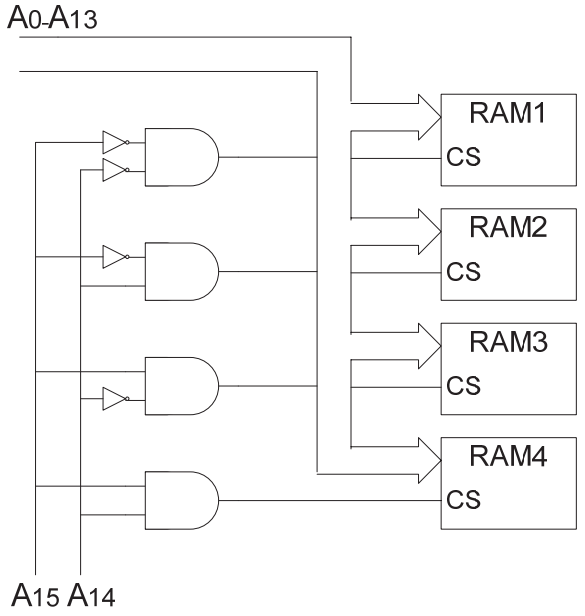
Слика 4.16. Временски дијаграм на контролните сигнали на DMA контролерот

Доколку се работи за бавен влезен уред тогаш нема потреба од целосно одземање на магистралите од микропроцесорот. Наместо DMA барањето да важи за сите бајти кои треба да се пренесат, тоа ќе важи само за еден бајт и после испраќањето на секој податок ќе се активира сигналот DREQ.

4.9. Адресно декодирање

Во еден компјутер има голем број мемориски и периферни уреди. Микропроцесорот не може истовремено да комуницира со два или повеќе уреда, туку само со еден од нив. Адресното декодирање е постапка за селекција на само еден уред од повеќето. При читање на еден податок од RAM меморијата, прво се селектира меморискиот чип, а потоа се избира една мемориска локација од веќе избраниот чип. Да се потсетиме, селекцијата на меморискиот чип се врши преку активирање на пиновите за селекција $\overline{CS}$, $\overline{CE}$ или $\bar{S}$. Дел од адресните линии на микропроцесорот се користат за добивање на сигналот за селекција на еден мемориски чип, а останатите линии служат за пристап до бараната мемориска локација.

Постапката адресно декодирање ќе ја објасниме преку примерот прикажан на сликата 4.17.



Слика 4.17. Адресен декодер со два адресни влеза

Овој пример е премногу едноставен и нереален, но ќе ни помогне да ја разбереме постапката на адресно декодирање. Вкупниот број адресни линии изнесува 16, од A_0 до A_{15}. Тоа значи дека располагаме со адресен простор од 2^{16}=64КВ. Адресниот простор е поделен меѓу четири RAM чипови. Капацитетот на секој чип изнесува 8КВ=64КВ:4. За избор на една локација од кој било чип се користат адресните линии од A_0 до A_{13}. Двете најзначајни адресни линии A_{14} и A_{15} служат за избор на еден од четирите чипа. Изборот на чип се врши според табела 4.5.

	A_{15} A_{14}
RAM1	00
RAM2	01
RAM3	10
RAM4	11

Табела 4.5. Табела на вистинитост за адресниот декодер со два адресни влеза

Сите адреси на првиот чип ќе почнуваат на 00, на вториот 01, на третиот 10 и на четвртиот 11. Останатите адресни битови се менуваат. Битовите од A_{13} до A_0 се нули за почетните адреси и единици за крајните адреси. На слика 4.18. се дадени почетните и крајните адреси на сите четири чипа, односно мемориската мапа за сите четири чипа.Во зависност од вредноста на битовите A_{15} и A_{14}, можеме да одредиме на кој мемориски чип припаѓа одредена адреса.

		A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
RAM1	почеток	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	крај	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM2	почеток	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	крај	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM3	почеток	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	крај	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM4	почеток	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	крај	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Слика 4.18. Мемориска мапа на RAM меморијата, составена од четири чипа од по 16KB

Пример 4.1. Микропроцесорот користи 16-битна адреса A_{15}-A_0. За селекција на мемориски модул, адресните линии A_{15} и A_{14} се поставени на следниве логички нивоа:

$A_{15} = 1$, $A_{14} = 0$

Која од наведените хексадецимални адреси припаѓаат на наведениот мемориски модул?

А) 6100H Б) 9100H В) C100H Г) F100H

Решение: Прво во редица ги пишуваме сите адресни битови од A_{15} до A_0. Второ, ги претвораме хексадецималните адреси во бинарен и тоа под редицата од адресни битови. Трето, ги заокружуваме адресните битови A_{15} и A_{14} и гледаме која адреса го исполнува горниот услов.

	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
6100H	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0
9100H	**1**	**0**	0	1	0	0	0	1	0	0	0	0	0	0	0	0
C100H	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
F100H	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0

Гледаме дека тоа е адресата 9100H.

Бројот на адресни линии и бројот на чипови се променливи величини.

Пример 4.2: Меморијата има вкупен капацитет од 2MB и е составена од 8 идентични чипа. Одреди колку адреснии линии се користат за избор на чип и кои адресни линии се користат за избор на локација од веќе избраниот чип.

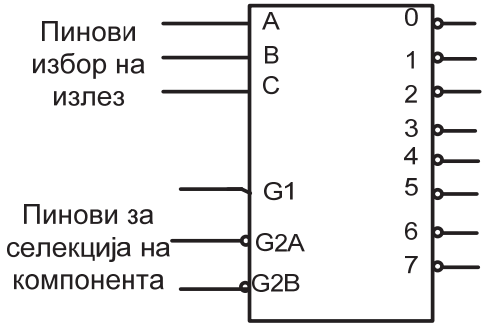
Решение: Вкупниот мемориски капацитет ни овозможува да го одредиме вкупниот број адресни линии. 2MB= $2^1 \cdot 2^{20} = 2^{21} \cdot 21$ претставува вкупен број на адресни линии кои се бележат од A_{20} до A_0.

	$A_{20}A_{19}A_{18}$
RAM1	000
RAM2	001
RAM3	010
RAM4	011
RAM5	100
RAM6	101
RAM7	110
RAM8	111

Табела 4.6. Адресни битови за адресно декодирање на 8 чипови со капацитет од по 256KB

Бидејќи се 8 чипови, ќе ни требаат три адресни битови за да направиме 8 различни комбинации. Секогаш се избираат најзначајните адреснии линии, и тоа во овој случај ќе бидат линиите A_{20}, A_{19} и A_{18}. Останатите адресни линии од A_{17} до A_0 ќе се користат за избор на еден бајт од вкупно 256KB колку што изнесува капацитетот на еден мемориски чип. Во табелата 4.6. се дадени вредностите на трите адресни битови за избор на чип.

Во микропроцесорската техника многу често се користат интегрирани кола - декодери. На сликата 4.19. е прикажан адресен декодер 74LS138. Тој има три адресни влеза што значи ќе има осум излези. Влезовите G1, G2A, G2B служат за селекција на компонентата. Пинот G1 треба да биде на високо ниво, а пиновите G2A, G2B на ниско ниво. Пиновите A, B и C служат за селекција на еден од излезните пинови. Во табелата 4.7. е дадена табелата на вистинитост.



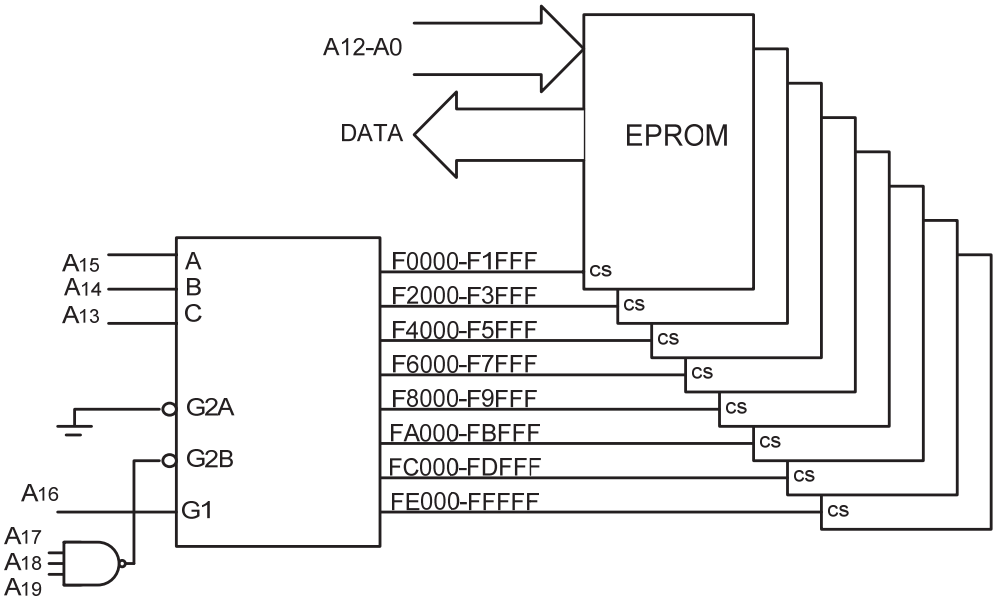
Слика 4.19. Пин дијаграм на декодер 74LS 138

ABC	Излез
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Табела 4.7. Табела на вистинитост на декодер 74LS138

Декодерот 74LS138 може да се користи за декодирање мемориски адреси и за декодирање интерфејс-адреси. На сликата 4.20. е прикажано

едноставно коло за декодирање составено од декодер 74LS138 и 8 мемориски EPROM чипа.



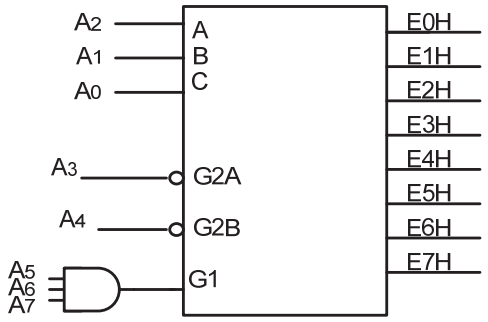
Слика 4.20. Поврзување на декодер 74LS38 со мемориски чипови

Секој од чиповите има капацитет од 8КВ што значи вкупниот капацитет ќе изнесува 64КВ. На сликата 4.20. се дадени и адресните опсези за секој од чиповите. Адресните битови од A_{13}, A_{14} и A_{15} служат за избор на еден излез од декодерот, односно еден мемориски чип од вкупно осум. Нивните вредности се менуваат според табелата 4.7. со тоа што $A_{15}A_{14}A_{13}$=ABC. A_{16} секогаш е единица, бидејќи пинот G1 е активен на логичка единица. Битовите A_{17}, A_{18} и A_{19} треба да се сите единици за да на излез од НИ колото имаме логичка нула,потребна за активирање на пинот G2B. Ако кој било бит од A_{17}, A_{18}, A_{19} биде еднаков на нула, тогаш на излаз од НИ колото ќе добиеме логичка единица, пинот G2B нема да се активира, а со тоа и декодерот 74LS138 ќе остане неактивен.

Декодерот 74LS138 има 8 излези и затоа со еден декодер 74LS138 може да се декодираат 8 интерфејс-адреси. Да се потсетиме, интерфес-адресата е уникатна комбинација од 8 бита и служи за избор на периферен уред. Интерфејс-адресата се носи на влез од декодерот 74LS138, прикажано на сликата 4.21.

Адресните битови од A_7 до A_4 треба да ги исполнат условите под кои ќе дојде до активирање на пиновите за селекција на компонентата. Условот што треба да се исполни зависи од логичките кола поставени пред пиновите G2A, G2B и G1. Петте битови, од A_7 до A_3 се константни и важат за сите осум периферни уреди приклучени на излезите од декодерот 74LS138. Битовите A_2, A_1 и A_0 служат за избор на еден периферен уред. За да се активираат пиновите G2A и G2B, треба A_3=0 и A_4=0. За пинот G1 ни треба логичка единица. За да

на излез од И-колото добиеме логичка единица, сите негови влезови треба да се еден, односно $A_5=A_6=A_7=1$.



A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
1	1	1	0	0	0	0	0	=E0H
1	1	1	0	0	0	0	1	=E1H
1	1	1	0	0	0	1	0	=E2H
1	1	1	0	0	0	1	1	=E3H
1	1	1	0	0	1	0	0	=E4H
1	1	1	0	0	1	0	1	=E5H
1	1	1	0	0	1	1	0	=E6H
1	1	1	0	0	1	1	1	=E7H

Слика 4.21. Табела на вистинитост за декодер 74LS138 за адресен опсег од E0H до E7H

Коментар: Доколку на влез од пинот G1 имавме логичко коло НИЛИ, тогаш трите бита A_7, A_6 и A_5 ќе требаше да бидат нули за да на излез од НИЛИ-колото добиеме логичка единица. Адресите ќе беа од 00H до 07H.

Заклучоци:

Операцијата читање од меморија трае три дигитски такта. Во првиот такт микропроцесорот испраќа адреса до меморијата. Вториот дигитски такт е такт на чекање и тогаш меморијата се пребарува и ја пронаоѓа саканата мемориска локација. Во третиот такт меморијата го испраќа саканиот податок до микропроцесорот. Доколку времето на пристап на меморијата е поголемо тогаш може да се вметнат два или повеќе такта на чекање.

Според концептот на виртуелна меморија програмата се дели на повеќе блокови и секој блок се извршува во микропроцесорот независно од другите. Откако ќе се изврши првиот блок од програмата, тој се враќа од главната меморија во секударната, а од секундарната во главната меморија се префрла следниот блок, па по него третиот и така натака додека не се изврши целата програма.

Со воведување на поимот виртуелна меморија се врши проширување на главната (примарна) меморија преку надворешната (секундарна) меморија. Секундарната меморија е поделена на страни, а главната меморија на рамки. Бројот на рамки е многу помал од бројот на страни. По потреба се врши замена на страните сместени во рамките.

Според намената кеш мемориите можат да бидат инструкциски и податочни. Во инструкциската кеш меморија се чуваат инструкциите кои се

однапред донесени од главната меморија и чекаат да бидат повикани за извршување од страна на микропроцесорот. Податочната кеш меморија ги чува податоците кои микропроцесорот почесто ги повикува, а за истите нема место во општите регистри на микропроцесорот.

Во пософистицираните мемориски организации постојат неколку нивоа на кеш меморија. Постојат три нивоа на кеш меморија. Кеш меморијата од прво ниво се наоѓа во самиот микропроцесорски чип, кеш меморијата од второ ниво се наоѓа во самото пакување на микропроцесорот и кеш меморијата од трето ниво се наоѓа на матичната плоча.

Баферот не врши обработка на сигналите туку тој врши харверско поврзување на периферните уреди со податочната магистрала само додека трае преносот на податоци. Лечот е уред составен од тактирачки флип-флопови чија задача е да го продолжат времетраењето на побудниот сигнал.

Колата кои посредуваат во комуникацијата меѓу периферните уреди и системската магистрала се нарекуваат влезно/излезни контролери или интерфејс контролери.

Адресирањето на периферните уреди може да се реализира на два начина: изолиран влез/излез и мемориски пресликан влез/излез. Кај изолираниот влез/излез се користи сигналот IO/$\overline{M}$ (input output/$\overline{memory}$) за одвојување на меморијата од влезно-излезните уреди. Кај мемориски пресликаниот влез/излез за одвојување на меморијата од периферните уреди се користи адресен бит, обично најзначајниот. Ако имаме 16 адресни линии, тогаш кога битот A_{15}=0 адресата ќе биде мемориска. Ако A_{15}=1 тогаш адресата од микропроцесорот ќе биде интерфејс адреса за периферен уред.

Пред да започне преносот на податоци влезниот уред испраќа сигнал за најава (Strobe), а ако микропроцесорот го одобри преносот тогаш тој испраќа сигнал за потврда (Acknowledge).

Паралелната порта се користи за поврзување на печатачи, а подоцна ќе видиме дека се користи и при програмирање на микроконтролери. Сериската порта на компјутерот (COM) се користи за поврзување на надворешни модеми, глувче, плотер и др. Исто така преку неа може да се изгради мала локална мрежа која користи оперативен систем Windows.

Периферните уреди испраќаат до микропроцесорот посебни барања за прекин. За таа цел се активира специјален пин INTR (interrupt). По извршувањето на секоја инструкција од тековната програма, микропроцесорот проверува дали постои барање за прекин. Ако такво барање постои, тогаш

микропроцесорот може да ја прекине тековната работа и да премине кон обработка на прекинот, при што се активира пинот INTA (Interrupt acknowledge).

DMA преносот претставува посебен концепт со кој микропроцесорот се ослободува од преносот на податоци меѓу меморијата и влезно излезните уреди. Концептот DMA е имплементиран преку употреба на специјален DMA контролер.

Пинот HOLD е влезен пин за микропроцесорот и преку него DMA контролерот испраќа барање за DMA пренос. Доколку микропроцесорот го одобри ова барање тој го активира излезниот пин HLDA (Hold Acknowledge). DMA контролерот прима барање за DMA пренос од периферниот уред преку пинот DREQ, а дава потврда преку активирање на сигналот DACK. DMA преносот завршува со активирање на сигналот EOP.

Адресното декодирање е постапка за селекција на само еден мемориски или периферен уред со кој микропроцесорот треба да оствари комуникација. Селекцијата на меморискиот чип се врши преку активирање на пиновите за селекција $\overline{CS}$, $\overline{CE}$ или $\overline{S}$. Дел од адресните линии на микропроцесорот се користат за добивање на сигналот за селекција на еден мемориски чип, а останатите линии служат за пристап до бараната мемориска локација.

Адресниот декодер 74LS138 има три адресни влеза и осум излези. Со него може да се декодираат 8 интерфејс-адреси или мемориски адреси. Влезовите G1, G2A и G2B служат за селекција на компонентата.

Прашања и задачи

1. Објасни го значењето на термините: синхронизација, сигнализација, TTL компатибилност и баферирање.

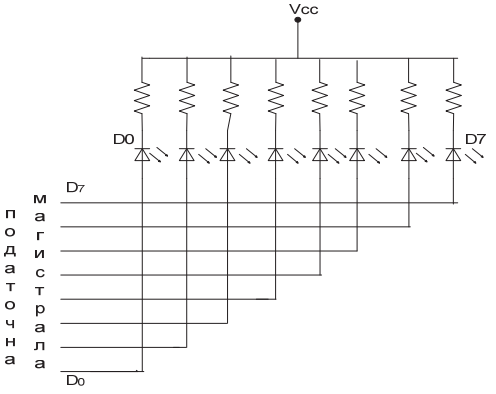
2. Лечот е коло што го продолжува времетраењето на побудниот сигнал. Објасни како?

3. Кои пинови се користат за поврзување на мемориите со општиот микропроцесор?

4. Објасни ја функцијата на сигналот wait? Кој уред го испраќа и прима овој сигнал?

5. Кои контролни сигнали се активираат за време на секој од трите дигитски такта во временскиот дијаграм за операцијата читање од меморија?

6. Кои лед диоди ќе светнат ако микропроцесорот испрати информација 54H до осумте лед диоди преку податочната магистрала? Објасни зошто!



7. Објасни го начинот на зголемување на меморискиот простор кај концептот на виртуелна меморија?

8. Микропроцесорот користи 32-битна виртуелна адреса. Големината на една страна изнесува 8KB. Колку страни може да собере виртуелниот адресен простор?

9. Виртуелната меморија содржи 16 страни, но само четири рамки за страни. На почетокот главната меморија била празна. Програмата ги доставува барања за следните страни: 0, 7, 2, 7, 5, 8, 2, 4. Кои барања ќе предизвикаат замена на страна ако се применат концептите LRU и FIFO?

10. Виртуелната меморија користи осум страни со големина од 1024 зборови и четири физички виртуени рамки. Подоле е прикажана состојбата на виртуелната меморија.

Виртуена страна	Рамка на страна
0	3
1	1
2	Не е присутна
3	Не е присутна
4	2
5	Не е присутна
6	0
7	Не е присутна

Кои виртуелни адреси ќе иницират замена на старата мемориска страна со нова?

11. Кои податоци се чуваат во кеш меморијата?

12. Спореди ги поимите кеш линија и кеш влез?

13. Објасни го значењето на битовите во кеш влезовите?

14. Колку пинови имаат портите: DB25, DSUB25 и DSUB9?

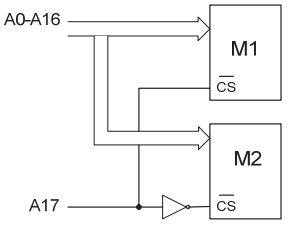
15. Каде се користи паралелната порта на сметачот?

16. Кои сигнали се генерираат со постапката на адресно декодирање?

17. Микропроцесорот користи 16-битна адреса A_{15}-A_0. За селекција на мемориски модул, адресните линии A_{15} и A_{14} се поставени на следниве логички нивоа A_{15}=0 A_{14}=1. Која од наведените хексадецимални адреси му припаѓа на наведениот мемориски модул?

А) 6500H Б) 5400H В) A700H Г) 0100H

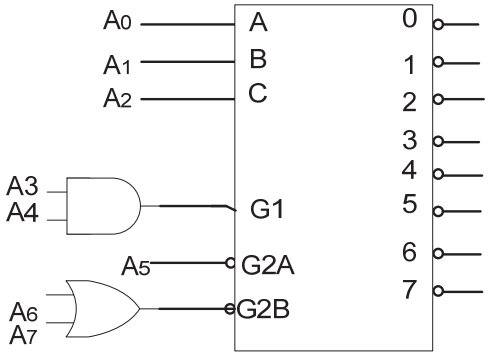
18. На сликата е прикажано коло за адресно декодирање на два мемориски модула. Кои адреси ги зафаќа меморискиот модул M2?



19. Како се бележат пиновите за селекција на компонентата и пиновите за избор на излез кај декодер 74LS138?

20. Микропроцесорски систем со 16-битна адресна магистрала ги користи следниве мемориски модули:
RAM1 со адресен опсег 0000h-3FFFH
RAM2 со адресен опсег 4000H-7FFFH
RAM3 со адресен опсег 8000H-BFFFH
EPROM со адресен опсег C000H-FFFFH
Кој од овие модули е адресиран ако најзначајните адресни битови се поставени на следниве логички нивоа A15=0 , A14=1?

21. На сликата е прикажан адресен декодер 74LS138 за декодирање на осум интерфејс-адреси. Која е адресата на шестиот излезен пин?



22. Набroj ги четирите режими на паралелен пренос меѓу микропроцесорот и периферните уреди!

23. Објасни за што служи сигналот STROBE кај стробираниот режим и режимот на двојно поздравување?

24. Кој сигнал ја одделува меморијата од периферните уреди кај адресното декодирање со мемориски пресликан влез/излез?

25. Кои се предностите и недостатоците на системот со мемориски пресликан влез/излез во однос на системот со мемориски изолиран влез/излез?

26. Објасни кои уреди ги активираат пиновите $\overline{\text{INTR}}$ и INTA и кога? На кое логичко ниво се активираат тие?

27. По одобрувањето на прекинот, периферниот уред ја испраќа интерфејс-адресата до микропроцесорот по податочната магистрала. Објасни за што ќе ја употреби микропроцесорот интерфејс- адресата?

28. Објасни ја постапката за маскирање прекини!

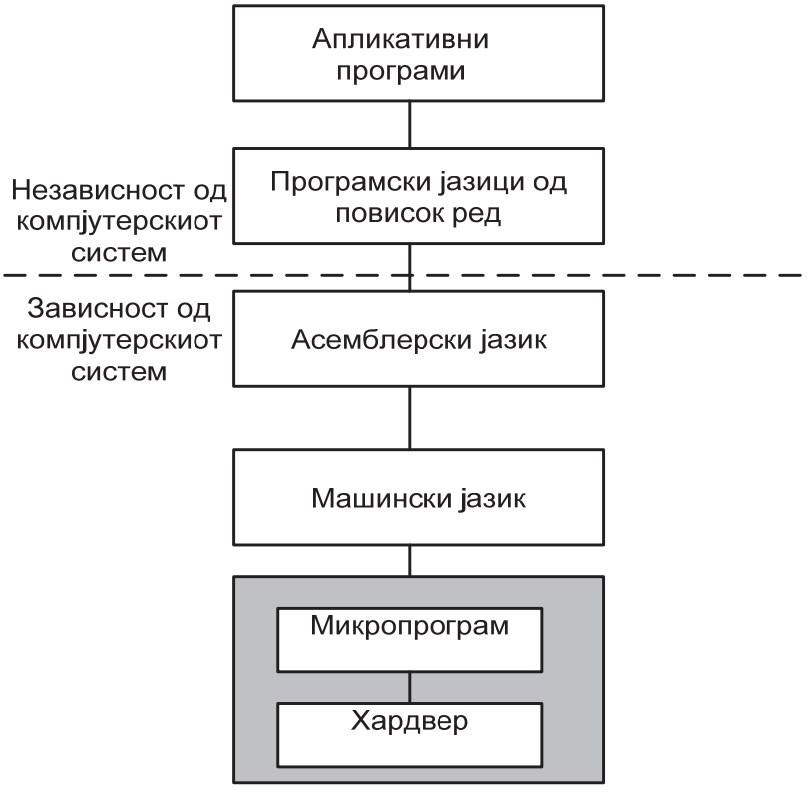
29. Кога се користи концептот на директен пренос на податоци?

30. Објасни ја функцијата на пиновите HOLD-HLDA и DREQ-DACK на DMA контролерот?

5. Програмирање на микропроцесор

5.1. Поделба на програмските јазици

Програмските јазици овозможуваат комуникација меѓу човекот и сметачите. Сите програмски јазици се поделени во три групи: машински јазици, асемблерски јазик и програмски јазици од повисок ред. На слика 5.1. прикажана е хиерархијата на различните видови програмски јазици и нивната имплементираност во компјутерските системи.



Слика 5.1. Хиерархија на програмски јазици

Асемблерот е програмски јазик кој посредува меѓу машинскиот јазик и програмските јазици од повисок ред. Тој спаѓа во групата на програмски јазици од понизок ред. Асемблерскиот јазик е погоден за пишување програмски рутини за влезно-излезните уреди, софтвер за разни видови мобилни уреди и развојни програми за пишување на нов софтвер. Вишите програмски јазици, како што се BASIC, FORTRAN, C++, PASCAL и други, се користат за изработка на апликативни програми.

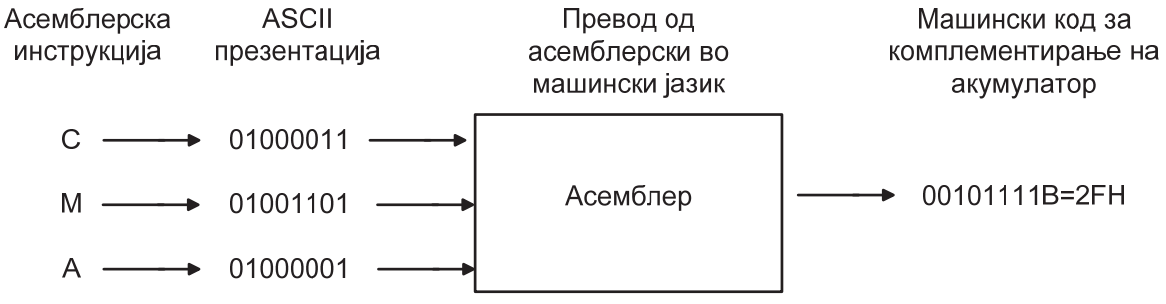
Програмерот пристапува кон микропроцесорот преку давање инструкции. Кај машинските јазици инструкциите се претставени во бинарен облик преку низи од нули и единици. Само во овој облик тие се прифатливи за микропроцесорот. Во рамките на секоја инструкција се разликуваат два главни дела: операциски код и адресен дел. Операцискиот код го одредува видот на операцијата што треба да ја изврши микропроцесорот, а адресниот дел ги определува адресите на локациите од кои треба да се земат или внесат потребните податоци. Подолу е прикажана една инструкција во машински код. Првите осум бита претставуваат операциски код, а сите останати адресен дел.

01110111000011100001

Машинските јазици се попогодни кога се сака да се постигне поголема искористеност на хардверските ресурси и да се зголеми брзината на работата. Бинарниот облик на инструкциите е непогоден за човекот. Човекот не може да ги запамети сите комбинации од битови со кои се претставени инструкциите. Во асемблер операциските кодови се претставуваат со кратенки од англиски зборови со кои се опишува значењето на инструкцијата, а адресите се заменуваат со симболични имиња. Според тоа, претходно дадената машинска инструкција во симболичен облик може да се запише вака:

Add address

Меѓу асемблерските и машинските инструкции постои еднозначна коресподенција. Преводот од асемблерски јазик во машински код го извршува самиот сметач со посредство на посебна програма-преведувач наречена асемблер. Слика 5.2. го прикажува преведувањето на инструкцијата за комплементирање на содржината на акумулаторот од асемблер во машински јазик.



Слика 5.2. Превод од асемблерски во машински јазик

Предности на програмските јазици од повисок ред во однос на асемблерскиот јазик се:

- Инструкциите се многу блиски до говорот и начинот на размислување на човекот. Програмските јазици од повисок ред дозволуваат успешно програмирање без доволно познавање на хардверот. Тоа не е случај со асемблерот.
- Програмирањето бара помалку време, програмите се релативно кратки и направените грешки се откриваат полесно. Едноставното програмирање е условено и од богатата програмска библиотека, која е составена од едноставни, но и сложени инструкции така да програмерот не мора да се мачи истите софтверски да ги креира.
- Најголема предност на програмските јазици од повисок ред е можноста за нивна примена во различни компјутерски системи со незначителни измени. За споредба асемблерот се разликува за различни микропроцесори и не може да се користи во различни системи.

Но, и покрај набројаните недостатоци на асемблерот во однос на програмските јазици од повисок ред, тој се уште се користи. Причина за ова се двете негови најважни карактеристики:

- Заштеда на мемориски простор. Програмата напишана во асемблер содржи многу поголем број на инструкции отколку истата програма напишана во повисок програмски јазик. Но, кога асемблерската програма ќе се преведе во машински јазик ќе се заклучи дека за меморирање на асемблерската програма ни треба многу помалку програмска меморија. Ова било посебно важно порано кога компјутерските системи имале трајни мемории со многу мал капацитет.
- Втора добра причина за употреба на асемблерот е многу големата брзина на извршување на програмите. Преведувањето од асемблер во машински јазик е многу лесно и брзо. Можеби ќе се изгуби повеќе време за составување на програмата, но за корисникот од голема важност ќе биде брзината со која се извршува програмата.
- Можеби најголема предност на асемблерот е неговиот директен пристап до хардверот. Со асемблерските инструкции ние пристапуваме до регистрите во внатрешноста на микропроцесорот или до некоја мемориска локација од RAM меморијата. Со секое извршување на инструкциите се побудува некој од пиновите на микропроцесорот или мемориските чипови. Учејќи го асемблерот ние всушност уште подобро го запознаваме хардверот на компјутерот.

5.2. Составни делови на асемблерска инструкција

Асемблерската инструкција се состои од 4 полиња:

- Поле за лабела
- Поле за мнемоник
- Поле за операнди
- Поле за коментар

Распоредот на овие полиња е следен:

[ЛАБЕЛА] МНЕМОНИК [ОПЕРАНДИ] [КОМЕНТАР]

Средните загради значат дека овие полиња не се задолжителни. Некои инструкции ги содржат, а некои не.

Лабелата се користи за именување на некоја инструкција или константа во програмата. Ова поле не е задолжително. Кај осумбитните микропроцесори лабелата се состои од 1 до 5 алфанумерички знаци. Како лабели не смеат да се користат имиња на регистри и операциски кодови. Кога се користи за именување на инструкција тогаш лабелата е одвоена од мнемоникот со знакот две точки (:) и празно место. Полето за лабела наоѓа примена кај инструкциите за гранење, инструкциите за потпрограми и циклуси. Имено сите овие инструкции во својот состав содржат адреса на мемориска локација. Лабелата всушност претставува замена за адресата на мемориската локација на која треба да се скокне или започне подпрограмата.

Мнемоникот претставува скратеница од англиските зборови со кои симболично се опишува функцијата на инструкција. Ова поле е задолжителен дел од било која асемблерска инструкција и понекогаш е единствено поле во инструкцијата. Во примерот прикажан на слика 5.2. мнемоник претставува CMA, што е всушност кратенка од зборовите Complement Accumulator и во превод значи пресметка на прв комплемент, негација на содржината на акумулаторот.

Операнди се податоци врз кои се извршува назначената инструкција. На пример, во собирање тоа се прв и втор собирок, во степенување имаме основа и експонент итн. Полето со операндите може да биде празно, да содржи еден или два операнда во зависност од видот на инструкцијата. Ако полето содржи два операнда, тогаш тие се издвоени со запирка. Во овој случај, едниот операнд претставува цел, а другиот извор на податок. Податоците што се појавуваат во полето на операндот можат да бидат прикажани во различни бројни системи:

- Со H се означува хексадецималниот број
- Со D декадниот број или е без ознака
- Со Q и O се означуваат окталните податоци
- Со B се означуваат бинарните податоци

Операндите не се секогаш константи. Операндот може да биде и променлива величина. Во тој случај операндот не се содржи во инструкцијата туку тој може да претставува содржина на некој општ регистар или мемориска локација. Начините за пронаоѓање на операндите се викаат начини на адресирање и со нив ќе се запознаеме подоцна.

Коментарот се пишува на крајот од асемблерската инструкција или како независни искази низ програмата. Асемблерските коментари секогаш се одвоени со знакот ; (точка запирка). Кога микропроцесорот ќе дојде до знакот точка запирка тој не го анализира понатамошниот текст. Но, за корисникот и програмерот коментарите се многу важни полиња бидејќи тие многу помагаат во толкувањето на програмата.

Подоле е прикажан пример за една конкретна инструкција.

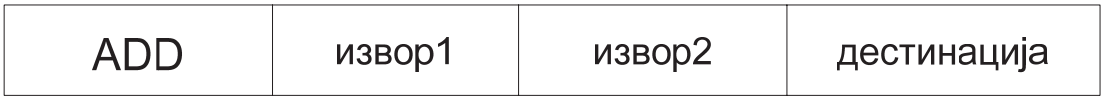
POVTORI: INC REZULTAT ; зголеми ја вредноста на резултатот за еден

POVTORI претставува лабела со која се именува инструкцијата INC REZULTAT. За програмерот е многу полесно да користи лабела отколку да работи со адресата на локацијата во која се наоѓа оваа инструкција. Најверојатно во понатамошниот текст на програмата програмерот ќе употреби инструкција за скок со која ќе треба да скокнеме токму до оваа инструкција INC REZULTAT. INC е мнемоник и претставува кратенка од англискиот збор Increase што во превод значи зголеми. REZULTAT е операнд. REZULTAT претставува симболично име за некој општ регистар или мемориска локација. Доделувањето на симболични имиња се врши преку специјални асемблерски наредби, наречени директиви. Со нив ќе се запознаеме подоцна кога ќе ја објасниме постапката за пишување на асемблерска програма. Се разбира, за нас оваа инструкција ќе беше многу потешка за толкување доколку го немаше коментарот на крајот од истата.

5.3. Начини на адресирање

Ако извршиме анализа на асемблерските инструкции преведени во машински јазик ќе заклучиме дека мал број битови од добиената низа се користат за прикажување на операцискиот код на инструкцијата. Поголем број од битовите се користат за дефинирање на операндите врз кои треба да се

изврши инструкцијата, отколку за дефинирање на самата инструкција. Ќе извршиме анализа на инструкцијата за собирање ADD, која бара дефинирање на три операнди: два собироци и еден збир. За собироците велиме дека се извори на податоци (source), а за збирот велиме дека е дестинација на добиениот резултат (destination). Ова е прикажано на слика 5.3.



Слика 5.3. Три-адресна асемблерска инструкција

На пример, ако мемориската адреса е 32-битна тогаш после операцискиот код инструкцијата ќе содржи три 32-битни адреси, односно 96 адресни битови. Адресите ќе бараат многу повеќе битови од самиот операциски код.

Генерално постојат две методи за намалување на битовите со кои се дефинираат операндите. Ако операндот се користи почесто тогаш тој може да се смести во некој општ регистар, а не во некоја мемориска локација. Добивката од употребата на регистрите за чување на операнди е двојна: побрз пристап до операндите и помалку битови потребни за дефинирање на операндите. На пример, ако микропроцесорот содржи 32 општи регистри тогаш ќе бидат потребни пет битови за да можат да се искодираат сите регистри (2^5=32). Значи ако операндите за операцијата собирање се наоѓаат во регистрите тогаш за нивно дефинирање ќе бидат потребни 15 битови, а доколку операндите се во мемориски локации тогаш ќе бидат потребни 96 битови.

Се разбира употребата на регистрите за чување на операнди не е секогаш идеално решение. Состојбата дури може и да се влоши доколку е потребно операндите во регистрите претходно да се пренесат од меморијата. Во тој случај ќе биде потребно да се дефинираат регистрите и мемориските локации. Ова воопшто нема да вреди доколку операндот што се пренесува се употреби само еднаш. За среќа анализите покажале дека најголем број операнди се користат повеќекратно. Поради тоа новите микропроцесорски архитектури предвидуваат многу поголем број на општи регистри сè со цел да се намали бројот на пристапи до меморијата.

Втор метод за редуцирање на битовите, потребни за дефинирање на операндите е методот на еднозначно прикажување на повеќе операнди. На пример, во една иста инструкција еден ист операнд ќе биде извор на податок и дестинација на добиениот резултат. Во ваков случај наместо ADD инструкцијата да содржи три адреси таа ќе содржи две адреси. Постапката собирање во случај на три-адресна инструкција е прикажана преку релацијата:

ДЕСТИНАЦИЈА=ИЗВОР 1+ИЗВОР 2

Во случај на две-адресна инструкција за собирање ќе важи релацијата:

РЕГИСТАР 2=РЕГИСТАР 2+ИЗВОР 1

Гледаме дека во случај на две-адресна инструкција регистарот 2 истовремено претставува и дестинација и извор 2. Недостаток на две-адресната инструкција е што после извршувањето на инструкцијата се губи старата содржина на операндот регистар 2. Доколку сакаме да ја заштитиме старата содржината на операндот регистар 2, истата треба да ја сместиме во некој друг општ регистар пред извршувањето на инструкцијата.

Процесорите од постарите генерации користат само еден општ регистар таканаречен акумулатор. На пример, со инструкцијата ADD секогаш на содржината на акумулаторот се додава содржината на некоја мемориска локација. Во овој случај треба да се дефинира само еден операнд односно мемориска локација во која се наоѓа операндот. Намалувањето на бројот на операнди од три на еден изгледа како голем напредок, но за жал ова важи само за едноставните пресметки, каде што не се јавуваат голем број на меѓурезултати.

Од досега изнесеното можеме да заклучиме дека операндите можат да бидат зададени на неколку начини:

- во самата инструкција;
- во некој од општите регистри на микропроцесорот;
- во некоја мемориска локација;
- или да бидат внесени преку некој влезен уред;

Во зависност од тоа како се пронаоѓа саканиот операнд постојат повеќе начини на адресирање, од кои поважни се: адресирање на регистри, директно адресирање, непосредно адресирање и регистарско индиректно.

Кај адресирањето на регистри операндите се наоѓаат во општите регистри на микропроцесорот. На слика 5.4. е прикажана инструкција која користи адресирање на регистри

мнемоник	регистар1	регистар2

Слика 5.4. Формат на адресирање на регистри

Инструкцијата е дво-адресна. Операндот после операцискиот код (регистар1) може да биде извор на податок и дестинација на добиениот резултат, а последниот операнд во инструкцијата (регистар 2) е само извор на податок. Предности на овој адресен вид е намалениот број пристапи до

меморијата. Потребните операнди се веќе во микропроцесорот, па истите не мора да се бараат од меморијата. На ваков начин се штеди во време потребно да се пребара меморијата и да се пренесат податоците до микропроцесорот, а ова значи побрзо извршување на инструкциите. Исто така кодот на инструкцијата е пократок кога операндите се наоѓаат во општите регистри, а не во мемориските локации. Недостаток на адресирањето на регистри е потребата од поголем број на регистри доколку сакаме да ги сместиме сите операнди во нив.

Нешто подолги инструкции се оние кои користат **непосредно адресирање.** Кај овој начин на адресирање операндот се содржи во самата инструкција. Зборот непосредно значи податокот следува веднаш после кодот на инструкцијата. На слика 5.5. е прикажана општата инструкција која користи непосредно адресирање.

мнемоник	регистар1	податок

Слика 5.5. Формат на непосредно адресирање

Гледаме дека на крајот од инструкцијата се наоѓа податок и тој може да биде број во било кој броен систем: бинарен, декаден или хексадецимален. Овој начин на адресирање е посебно згоден за работа со константи. На пример, се применува кога некој општ регистар треба да се наполни со константна вредност или кога некој број треба да се додаде на содржината на некој општ регистар. Непосредното адресирање не дава можност за работа со варијабили.

За работа со варијабили се користи **директното адресирање**. Кај овој начин на адресирање инструкцијата ја содржи адресата на мемориската локација во која се наоѓа операндот. На слика 5.6. е прикажана општата инструкција која го користи овој вид на адресирање.

мнемоник	регистар1	Адреса на мемориска локација

Слика 5.6. Формат на директно адресирање

Директното адресирање многу често се користи бидејќи нуди пристап до сите локации од меморијата, за разлика од регистарското каде што имаме пристап до многу мал број на регистри. Можноста за работа со варијабили се состои во тоа што адресата на мемориската локација останува иста, а нејзината содржина може да се менува секогаш кога ќе се пристапи до локацијата. Недостаток на директното адресирање е должината на иструкциските формати бидејќи инструкциите во себе содржат мемориски адреси кои се подолги кодови од кодовите на регистрите и самите податоци.

Втор недостаток на директното адресирање е долгото времетраење на инструкциите. На пример, ако инструкцијата со адресирање на регистри е составена од еден бајт тогаш инструкцијата со директно адресирање ќе биде составена од најмалку три бајта. Тоа значи дека самиот пренос на инструкцијата со директно адресирање ќе трае три пати подолго. Кога ќе се пренесат сите бајти од инструкцијата повторно треба да се оди до меморијата со адресата од инструкцијата и да се пронајде и пренесе саканиот операнд.

Регистарско индиректното адресирање користи адреса на мемориска локација за пронаоѓање на саканиот операнд. Но, кај овој начин на адресирање адресата на локацијата не се содржи во самата инструкција, туку во некој регистар во микропроцесорот. Регистрите кои се користат за чување на адреси се наречени покажувачи (pointer). На ваков начин ќе се задржи пристапот до сите локации од меморијата, а ќе се скрати должината и времетраењето на инструкциите.

5.4. Инструкциско множество на општ микропроцесор

Инструкциите за програмирање на микропроцесорите се слични меѓу себе, без разлика за кој микропроцесор станува збор. Се разбира постојат и извесни разлики. На пример, постапката за собирање на два броја е иста кај сите видови микропроцесори, со таа разлика шно некои користат две-адресни инструкции, а други три-адресни инструкции за собирање. Сите инструкции можат да се поделат во неколку групи: инструкции за пренос на податоци, аритметички инструкции, логички инструкции, инструкции за гранење, инструкции за реализација на потпрограми и инструкции за работа со стек меморија.

5.4.1. Инструкции за пренос на податоци

Кај инструкциите за пренос на податоци саканиот податок го земаме од изворот и го копираме во дестинацијата. Извор или дестинација може да бидат регистрите во микропроцесорот или локациите во меморијата. Според тоа можат да постојат четири различни видови на пренос на податоци: од регистар во регистар, од регистар во мемориска локација и од мемориска локација во регистар. Инструкции за четвртиот вид на пренос, од една мемориска локација во друга мемориска локација, не постојат. Да не заборавиме дека податокот може да биде даден и непосредно.

Најчесто користен мнемоник за пренос е MOV што претставува кратенка од англискиот збор Move. Овој мнемоник во превод значи помести, иако подобар збор за објаснување на преносот на податоци би бил зборот копирај (copy). Кога ќе кажеме помести го податокот помислуваме дека изворот од кој го земаме податокот ќе се испразни, а дестинацијата ќе се наполни со тој податок. Но, не е така. После инструкцијата MOV податокот од изворот ќе се ископира во дестинацијата, така да изворот и дестинацијата ќе имаат исти содржини. После инструкцијата MOV во дестинацијата ќе имаме дупликат на податокот од изворот. Незгодно е што дестинацијата го губи податокот што го имала пред инструкцијата MOV. Кај различни микропроцесори изворот и дестинацијата имаат различни положби во асемблерската инструкција. На пример, во инструкциите за пренос кај Интеловите микропроцесори после мнемоникот се запишуваа дестинацијата, а потоа изворот на податок. Ова е прикажано во следнава инструкција.

MOV дестинација, извор.

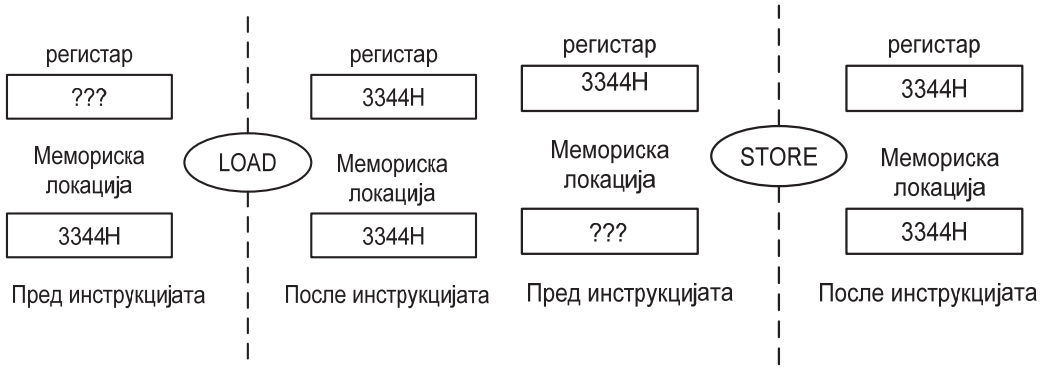
Дестинацијата и изворот меѓу себе се разделени со запирка. Подоле се дадени две асемблерски инструкции за пренос и нивните коментари.

```
MOV регистар, адреса        ;Податокот од мемориската локација со
                            ;дадената адреса го копираме во регистарот.

MOV адреса , регистар       ;Податокот од регистарот го копираме во
                            ;мемориската локација со зададената адреса.

MOV регистар, 65H           ;Податокот 65H се внесува во регистарот.
```

Кај поголем број на микропроцесори постојат специјални инструкции за пренос на податоци од или кон меморија:

- LOAD- Овој англиски збор во превод значи наполни и служи за пренос на податок од мемориската локација со зададената адреса во акумулаторот.
- STORE- Овој англиски збор во првод значи складирај и служи за пренос на податок од акумулаторот во мемориска локација со наведената адреса.

Ова графички е прикажано на слика 5.7. Гледаме дека после извршувањето на двете инструкции, LOAD или STORE, содржината на регистарот и мемориската локација се идентични, но различни е изворот на податокот 3344H. Кај инструкцијата LOAD извор на податок е мемориската локација, а кај инструкцијата STORE извор е регистарот. Исто така, кај инструкцијата LOAD се губи старата содржина на регистарот, а кај инструкцијата STORE се губи содржината на мемориската локација.



Слика 5.7. Извршување на инструкциите LOAD и STORE

Во групата на инструкции за пренос на податоци спаѓаат и инструкциите за работа со влезно излезни уреди. Поголем дел од микропроцесорите имаат една инструкција за пренос од влезен уред кон регистар во микропроцесорот и една инструкција за пренос од регистар кон излезен уред. Мнемоник за влезната инструкција е IN, а за излезната OUT. Подоле се дадени две инструкции за пренос на податоци од влезно излезни уреди.

```
IN регистар, интерфејс адреса        ;Пренос од влезен уред во регистар.

OUT интерфејс адреса, регистар       ;Пренос од регистар во излезен уред.
```

Интерфејс адресата претставува бинарен код за препознавање на периферниот уред. Секој влезен или излезен уред си има своја интерфејс адреса и преку неа микропроцесорот дознава со кој уред ќе комуницира. Како што локациите имаат мемориски адреси така влезно излезните уреди имаат интерфејс адреса. Исто така од дадените инструкции IN и OUT можеме да заклучиме дека дестинацијата на податокот се пишува после мнемоникот на инструкцијата, а изворот на податокот после запирката, исто како што беше и кај MOV инструкцијата.

На крајот да споменеме дека кај инструкциите за пренос треба да водиме сметка за големината на пренесуваниот податок. На пример, не можеме да пренесуваме 16 битен податок во 8 битен регистар бидејќи едноставно за податокот ќе нема доволно место. Исто така не се препорачува пренос на 8-битен податок во 16-битен регистар затоа што нема да биде искористен целиот капацитет на регистарот.

5.4.2. Аритметички инструкции

Најчесто користени аритметички инструкции се инструкциите за собирање и одземање. Инструкција за собирање е ADD, а за одземање SUB. SUB е всушност кратенка од англискиот збор Subtract што во превод значи одземање. Постојат неколку варијанти на овие инструкции. Кај некои

микропроцесори инструкцијата за собирање содржи два операнди, а кај други микропроцесори три операнди. Ова е илустрирано преку следниве две инструкции:

```
ADD дестинација, извор                ;Изворот се додава на дестинацијата.

ADD извор1, извор2, дестинација       ;Дестинација=извор1+извор2
```

Инструкциите за собирање се разликуваат и по тоа дали се зема или не во предвид знаменцето за пренос (Carry). Подоле се дадени две такви инструкции:

```
ADD извор, дестинација      ; На дестинацијата се додава само изворот.

ADC извор дестинација       ;(Add Carry) На дестинацијата се додава
                            ;изворот и  вредноста на знаменцето за
                            ;пренос.
```

Истото што важи за инструкцијата за собирање важи и за инструкцијата за одземање.

После собирањето и одземањето доаѓаат инструкциите за множење и делење. Инструкција за множење е MUL што е кратенка од англискиот збор за множење, multiply. Инструкција за делење е DIV што е кратенка од англискиот збор за делење-divide. Да напоменеме дека при операцијата множење производот ќе биде двапати подолг од множителите, а кај операцијата делење покрај место за количникот треба да се одреди и место за остатокот. На пример, ако множиме два 8 битни броеви тогаш производот може да биде 16 битен. Подоле се дадени две инструкции за множење.

```
MUL извор1, извор2, дестинација   ;дестинација=извор1·извор2

MUL  извор                        ;Содржината на работниот регистар се
                                  ;множи со изворот и производот се сместува
                                  ;повторно во работниот регистар.
```

Кај инструкцијата со три операнди, делителот, деленикот и количникот може да бидат во било кој регистар или мемориска локација. Кај инструкцијата со еден операнд деленикот и количникот секогаш се наоѓаат во работниот регистар, а делителот може да биде во било кој регистар или мемориска локација. Се разбира дека делителот може да биде зададен и непосредно, како константа. Да споменеме дека постојат специјални инструкции за множење и делење на броеви со предзнак и нивните мнемоници се IMUL и IDIV.

Многу практична примена наоѓаат инструкциите за намалување или зголемување за еден. Овие инструкции се користат кога е потребно да се

изврши броење на некоја величина. Инструкција за зголемување со еден е INC (Increase) , а за намалување со еден DEC (Decrease). Овие две инструкции имаат еден операнд, регистар или мемориска локација.

5.4.3. Логички инструкции

Најважни логички инструкции се инструкциите: AND (логичко множење), OR (логичко собирање) и NOT (негација или прв комплемент). Некои микропроцесори ги користат и инструкциите XOR (ексклузивно или), NOR (НИЛИ коло) и NAND (НИ коло). Со значењето на логичките функции се запознавме во првата тема, Основни логички кола. Слично како аритметичките инструкции, и логичките инструкции имаат по два извори на операнди и дестинација за добиениот резултат. Кај некои микропроцесори еден од изворите се користи и како дестинација. Подоле се прикажани неколку логички инструкции и нивните коментари.

```
AND дестинација,извор          ;Логичко множење на изворот и дестинацијата
                               ;и резултатот се запишува во дестинацијата.

NOT дестинација                ;Замена на дестинацијата со неговиот прв
                               ;комплемент.

OR извор1, извор2, дестинација ;Логичко собирање на двата извори и
                               ;запишување на резултатот во дестинацијата.
```

Да се потсетиме дека изворите и дестинацијата можат да бидат општи регистри или мемориски локации, а еден од изворите може да биде зададен и непосредно како број во бинарен, декаден или хексадецимален броен систем.

Многу често еден од операндите на логичките инструкции претставува маска. Маската претставува однапред дефинирана комбинација од нули и единици. Каков ќе биде распоредот на битовите зависи од неменатa на маската. На пример, логичката инструкција AND се користи кога треба да се издвои еден бајт од 16, 32 или 64-битен податок. Подоле е прикажан еден ваков пример, кога од 32-битниот податок треба да се издвои вториот бајт од десно. За таа цел ќе се употреби маска која ќе биде составена од нули на позицијата од оние бајти кои треба да се изгубат, а на местото од вториот бајт од десно ќе се постават само единици.

Пример 5.1:

```
          00110010 00011100 10111001 11100010--- 32-битен податок
AND       00000000 00000000 11111111 00000000---маска
-------------------------------------------------
          00000000 00000000 10111001 00000000---издвоен бајт
```

Логичката инструција OR се користи кога треба да се тестира некој бит од податокот. Во маската на позицијата од тој бит ќе биде поставена единица, а на сите останати позиции логичка нула. Ако после извршеното логичко собирање не добиеме резултат единици на сите позиции, тоа значи дека испитуваниот бит бил логичка нула. Ова е прикажана во примерот подоле каде што се тестира првиот бит од десно.

Пример 5.2:

```
            xxxxxxxx  xxxxxxxx---испитуван податок
OR          11111111 11111110---маска
-------------------------------------------------
            11111111 1111111x---Резултат
```

5.4.4. Инструкции за ротација и поместување

Инструкциите за ротација и поместување се многу корисни и ги има во повеќе варијанти. За разлика од досега изучените инструкции, овие имаат само еден операнд односно податокот се зема од изворот, се обработува и се враќа на истото место. Ова значи дека изворот и дестинацијата се едно исто.

Потапката на ротација ќе ја објасниме преку пример. 16-битниот податок 10000000 ќе го ротираме за едно место во лево, а потоа за едно место во лево. При ротација во лево сите битови од податокот се поместуваат за едно место во лево, нултиот бит доаѓа на местото од првиот, првиот на местото од вториот,..., шестиот на местото од седмиот. Седмиот бит неможе да оди во лево, тој ќе излезе од податокот, ќе кружи и ќе дојде на местото од нултиот бит. При ротација во десно сите битови ќе се поместат за едно место во десно, седмиот на местото од шестиот, шестиот на местото од петтиот,...., првиот на местото од нултиот бит. Нултиот бит ќе излезе од податокот, ќе кружи од десно кон лево и ќе дојде на местото од седмиот бит.

```
10000000----податок
-------------------------------------------------
00000001---- ротација во лево
01000000---- ротација во десно
```

Ротацијата на битовите може да се изврши неколку пати, а ова се нагласува преку бројот запишан на крајот од инструкцијата. Подоле се прикажани неколку инструкции за ротација.

```
ROL дестинација, 4      ;Битовите во дестинацијата се ротираат за три
                        ;места во лево.

ROR дестинација, 3      ;Битовите во дестинацијата се ротираат за три
                        ;места во десно.
```

```
ROL извор, дестинација, 4     ;Битовите од изворот се ротираат четири пати во
                              ;лево и резултатот се сместува во дестинацијата.
```

Инструкциите за поместување (shift) се многу слични со инструкциите за ротација. Кај инструкциите за поместување битовите од податокот се поместуваат за неколку места во лево или десно, но оние битови кои ќе го напуштат податокот се губат, не доаѓаат од другата страна на податокот. Испразнетите места се надополнуваат со нули. Ова е прикажано во примерот подоле при што битовите се поместени за три места во лево или десно.

1110001010011101------податок

0001110001010011------поместување во десно за три места
0001010011101000------поместување во лево за три места

Мнемоници за овие инструкции се SHL (Shift Left) за поместување во лево и SHR (Shift Right) за поместување во десно. Подоле се дадени неколку примери.

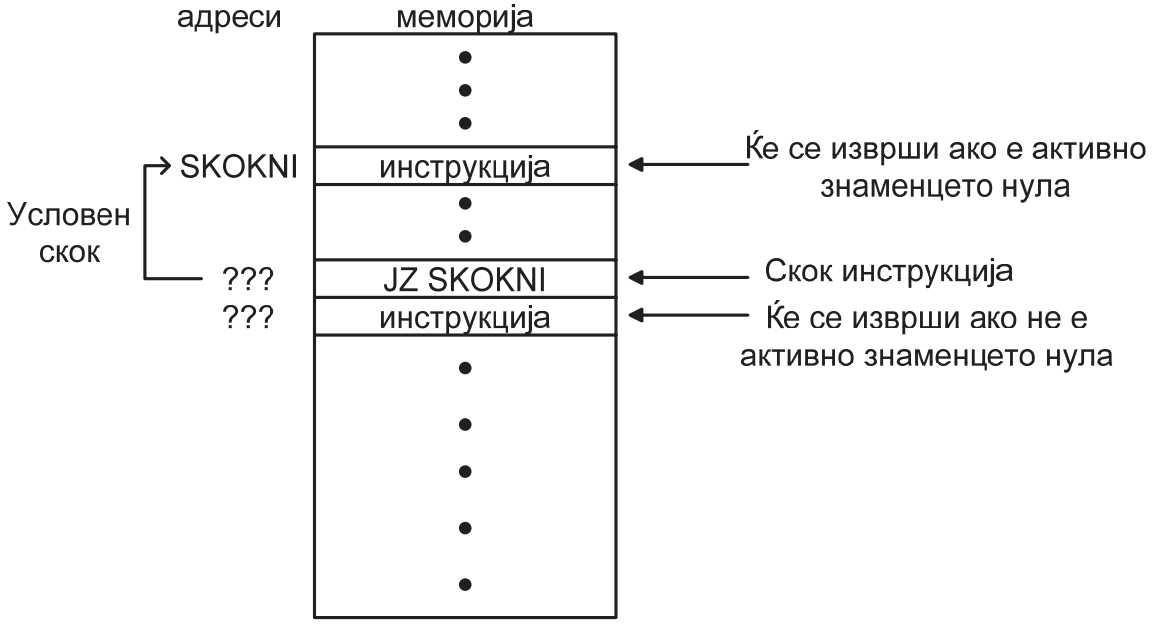
```
SHL дестинација, 5            ;Поместување на битовите во дестинацијата за пет
                              ;места во лево.

SHR дестинација,4             ;Поместување на битовите во дестинацијата за
                              ;четири места во десно.

SHR извор, дестинација, 2     ;Битовите од изворот се поместуваат во десно два
                              ;пати и резултатот се сместува во дестинацијата.
```

5.4.5. Инструкции за скок и гранење

При извршувањето на програмата многу често се јавува потреба да се провери, тестира добиениот резултат. Од вредноста на добиениот разултат зависи дали ќе се изврши следната инструкција или некоја друга. Да се потсетиме, резултатите од извршените ариметичко логички инструкции дирекно влијаат врз состојбата на знаменцата во статус регистарот. Со значењето на знаменцата се запознаваме во третата тема, единица Аритметичко логичка единица. Бидејќи извршувањето на инструкцијата е условено од состојбата на знаменцата овие инструкции се познати под името условни инструкции. Бидејќи од овие инструкции зависи текот на програмата тие се познати и под името контролни инструкции.

Многу често се користат инструкциите за скок. Мнемоник за овие инструкции е JMP. Тоа е кратенка од зборот Jump што во превод значи скок. Да

нагласиме дека освен условни скокови постојат и безусловни скокови чие извршување не зависи од состојбата на знаменцата. Овие инструкции во себе не содржат ниту еден операнд, туку ја содржат адресата на мемориската локација на која треба да се скокне. Понекогаш адресата не е зададена како број туку како лабела. На слика 5.8. е прикажано извршувањето на инструкцијата за условен скок JZ (Jump Zero). За адресата е употребена лабела, односно локацијата на која треба да се скокне има адреса со симболично име SKOKNI. Скокот ќе биде извршен доколку се акивира знаменцето нула, односно ако резултатот од претходно извршената инструкција е еднаков на нула. Доколку не е исполнет овој услов тогаш нема да се скокне на адресата наведена во скок инструкцијата, туку програмата ќе продолжи со својот тек односно ќе се изврши следната инструкцијата, која се наоѓа под скок инструкцијата.



Слика 5.8. Графички приказ за извршување на скок инструкција

Подоле се прикажани повеќе инстукции за скок со нивните коментари.

JMP адреса ;Безусловен скок

JZ адреса ;Скокот ќе се изврши ако е активно знаменцето нула (резултат ;еднаков на нула).

JS адреса ;Скокот ќе се изврши ако е активно знаменцето за знак ;(негативен резултат).

JC адреса ;Скокот ќе се изврши ако е активно знаменцето за пренос.

JO адреса ;Скокот ќе се изврши ако е активно знаменцето за ;пречекорување.

Инструкциите за гранење (branch) се слични со инструкциите за скок. Кај инструкциите за гранење се врши споредба меѓу два операнди и во зависност од добиениот резултат ќе се изврши или не скокот во наведената инструкција. Споредуваните операнди можат да бидат еднакви, првиот да биде поголем или помал од вториот. Во понатамошниот текст се прикажани неколку инструкции за гранење

```
BEQ извор1,извор1, адреса    ;(Branch Equal) Ако извор1 и извор2 се еднакви се
                             ;скока на наведената адреса.

BEQZ извор, адреса           ;(Branch Equal Zero) Ако изворот е еднаков на нула
                             ;се скока на наведената адреса.

BGT извор1, извор2, адреса   ;(Branch Greater Than) Ако извор1 е поголем од
                             ;извор2 се скока на наведената адреса.
```

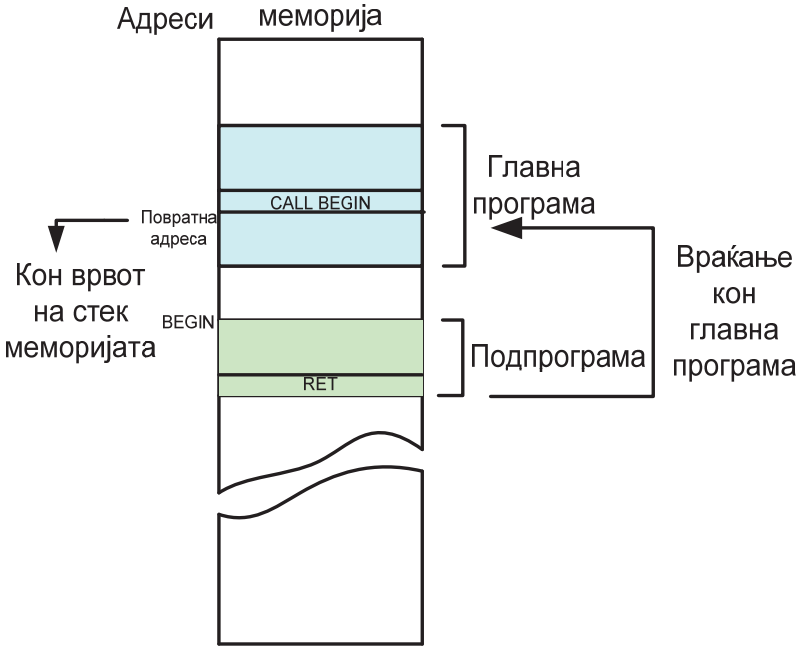
5.4.6. Инструкции за работа со потпрограми

Потпрограмата се извршува независно од главната програма и може да биде сместена било каде во меморијата. Потпрограмата може да биде програма за пресметка на sin, квадрат или логаритам од некоја променлива величина или програма за внесување или за изнесување податоци од одреден периферен уред. Кога главната програма содржи потпрограма, доаѓа до прекин на главната програма сè додека не биде завршена потпрограмата. Во овој случај, во програмскиот бројач ќе се смести адресата на првата инструкција од потпрограмата и неговата содржина ќе се зголемува за еден сè додека не биде извршена последната инструкција од потпрограмата. Сега се поставува прашањето: Како да се вратиме во главната програма? За оваа цел, пред да се скокне на потпрограмата, треба да се запамти адресата на следната инструкција од главната програма. Програмскиот бројач не може да се искористи, бидејќи е зафатен со потпрограмата. Адресата на следниот бајт од главната програма се запишува на врвот од стек меморијата (пласт). Кога микропроцесорот ќе заврши со извршување на потпрограмата, ја префрла адресата од врвот на стекот во програмскиот бројач што значи враќање назад во главната програма.

Често се случува самата потпрограма да содржи друга потпрограма. Во тој случај враќањето од првата потпрограма во главната потпрограма се остварува со внесување на адресата на следниот бајт од главната програма во втората локација од стекот (гледано одгоре надолу), а за враќање од втората во првата потпрограма ќе биде искористена најгорната мемориска локација од

стекот. Колку потпрограми имаме, толку мемориски локации од стекот ќе бидат зафатени.

На слика 5.9 е прикажана постапката за заштита на повратната адреса во случај на повикување на подпрограма и враќањето од подпрограмата кон главната програма.



Слика 5.9. Постапка за извршување на потпрограма

Постојат две инструкции за работа со потпрограми. Едната е за повикување на потпрограма и има мнемоник Call што во превод значи повикај. Оваа инструкција во себе ја содржи само адресата на мемориската локација од која почнува потпрограмата (Call адреса). Втора инструкција е инструкцијата за враќање од потпрограмата во главната програма и таа има мнемоник RET што е кратенка од англискиот збор Return што во превод значи враќање. Да напоменеме дека вредноста на програмскиот бројач автоматски се носи на врвот од стекот кога ќе се појави Call, а се враќа назад кога ќе се појави RET инструкцијата. Тоа не е работа на програмерот.

5.5. Пишување на програми

Програмата на асемблерски јазик се вика изворна програма. Најнапред е потребно изворната програма да се внесе во меморијата (преку некој влезен уред, најчесто тастатура) и да се запамети во некоја датотека под одредено име. Внесувањето на програмата се извршува преку специјална програма наречена едитор.

По запишувањето на изворната програма во некоја од надворешните мемории, го повикуваме асемблерот и му наредуваме да ја преведе програмата. Преводот се врши од асемблерски јазик на машински јазик. Асемблерот пристапува кон преведување и доколку нема грешки, креира таканаречена објект-програма, која потоа повторно се запишува во посебна датотека.

Објект-програмата е програма на машински јазик, но ни оваа програма не е спремна за извршување. За да таа стане употреблива, треба да се смести на посебно место во меморијата и евентуално да се поврзе со други програми. Оваа работа ја врши специјалната програма наречена поврзувач (линкер). Доколку нема проблеми при поврзувањето (на пример, не постои потребниот модул), линкерот формира таканаречена извршна програма. За отстранувањето грешки во асемблерската програма се користи специјална програма наречена дебагер.

Ако програмата е напишана во некој виши програмски јазик, тогаш за нејзино преведување на машински јазик се користи специјална програма-преведувач наречена компајлер. Едиторот, асемблерот, линкерот, дебагерот и компајлерот претставуваат алатка програми. За секоја од овие помошни програми програмерот треба да знае како да ги користи, т.е. потребно е да се познаваат наредбите преку кои се користат алатка програмите.

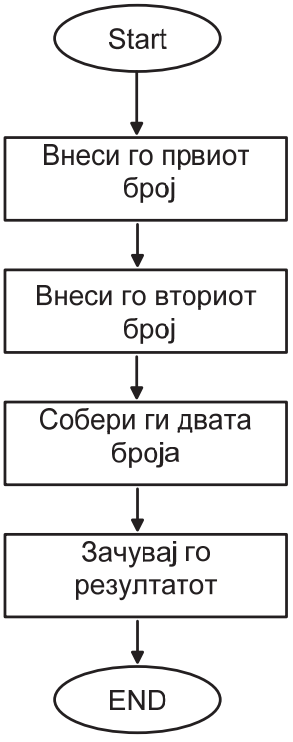
Според структурата програмите може да се поделат во неколку групи: линиски програми, програми со гранење, програми со циклуси, програми со потпрограми и сложени програми. Во понатамошниот текст ќе се запознаеме со секоја од овие структури.

5.5.1. Линиски програми

Овие програми се составени од инструкции за пренос, аритметички, логички инструкции и инструкции за ротација и поместување. Инструкциите се наредени една врз друга и така се извршуваат. Не може да се случи некоја инструкција да биде прескокната или да има прекин во програмата поради извршување на подпрограма. Исто така не може да се случи враќање назад во програмата. Значи откако ќе започне линиската програма цело време се спуштаме надолу по програмата се додека не дојдеме до нејзиниот крај. Вредноста на програмскиот бројач линеарно се зголемува и следната инструкција е секогаш под тековната.

Како пример за линеарна програма ќе ја објасниме програмата која врши собирање на два броја. Двата собироци се наоѓаат во меморијата и потребно е истите да се пренесат во микропроцесорот, поточно во два негови регистри. Исто така резултатот се заштитува преку негово пренесување во меморијата.

Заради подобро разбирање на програмата даден е нејзиниот блок дијаграм, во кој се објаснети сите чекори при креирањето на програмата.



Слика 5.10. Блок дијаграм за линиска програма

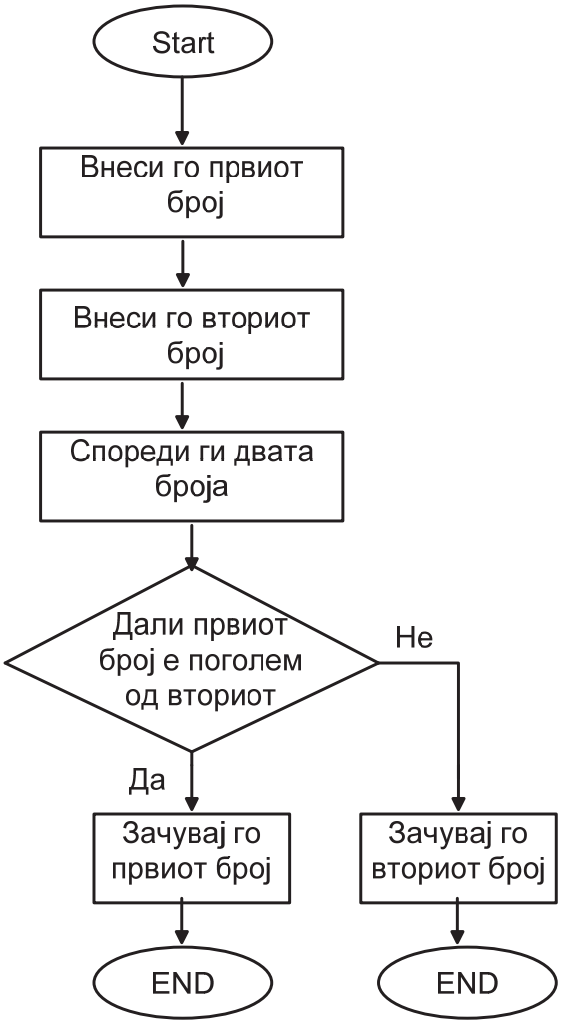
```
MOV R1, адреса1      ;Вредноста на локацијата со наведената адреса се копира
                     ;во регистарот R1.

MOV R2, адреса2      ;Вредноста на локацијата со наведената адреса се копира
                     ;во регистарот R2.

ADD R1,R2            ;Вредноста на регистарот R2 се додава на вредноста на
                     ;регистарот R1.

MOV адреса3, R1      ;Резултатот од собирањето се запишува на мемориска
                     ;локација со адреса3.
```

5.5.2. Програми со гранење

Програмите со гранење во својот состав содржат инструкции за скок (Jump) и инструкции за гранење (Branch). Да се потсетиме овие инструкции се нарекуваат контролни инструкции бидејќи тие можат да го сменат текот на програмата, односно следната инструкција не се наоѓа секогаш под тековната инструкција. Во која насока ќе се одвива програмата зависи од тоа дали е

исполнет условот наведен во инструкциите или не. Условот уште се нарекува алтернатива и во блок дијаграмите таа се бележи со ромб.

Како пример ќе ја објасниме програмата за споредување на два позитивни броја и потоа ќе се измеморира поголемиот број.



Слика 5.11. Блок дијаграм за програма со гранење

За споредба на двата броја се користи операцијата одземање. Ако при одземањето се активира знаменцето за пренос (carry) тогаш вториот број е поголем од првиот, а во случај да не се активира ова знаменце тогаш првиот ќе биде поголем од вториот. Како и кај линиската програма ѝ тука двата броја се пренесуваат од меморијата и се сместуваат во два општи регистри.

```
MOV R1, адреса1     ;Вредноста на локацијата со наведената
                    ;адреса се копира во регистарот R1.

MOV R2, адреса2     ;Вредноста на локацијата со наведената
                    ;адреса се копира во регистарот R2.
```

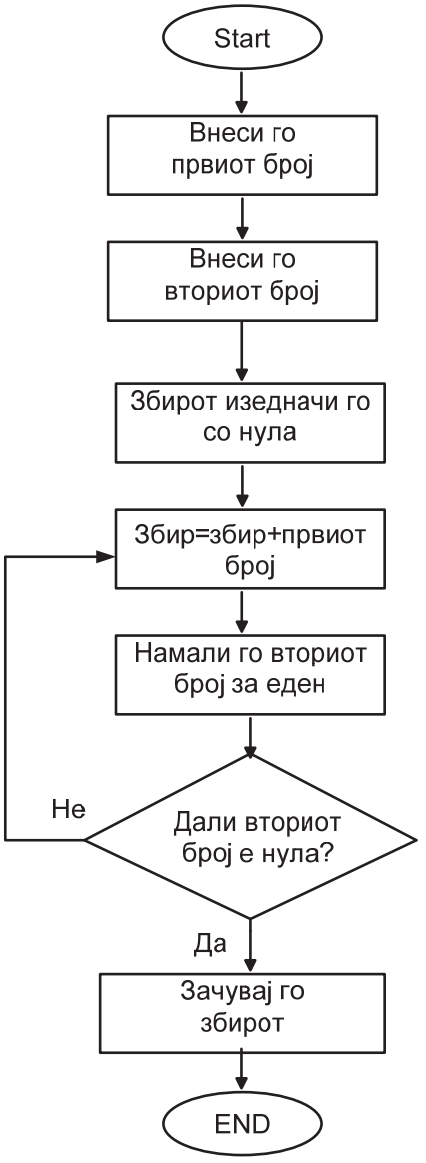
```
        SUB R1, R2          ;Од вредноста на регистарот R1 се одзема
                            ;вредноста на регистарот R2.

        JC VTORIOT          ;Ако е активно знаменцето за пренос (Carry) се
                            ;скока на локацијата со адреса VTORIOT.

        MOV адреса3, R1     ;Ако знаменцето за пренос не е активно во
                            ;мемориската локација се пренесува
                            ;содржината на регистарот R1.

        JMP KRAJ            ;Се скока следната инструкција за да
                            ;регистарот R2 не се запише во мемориската
                            ;локација.

VTORIOT: MOV адреса3, R2    ;Регистарот R2 се запишува во мемориската
                            ;локација.

KRAJ:           /
```

Гледаме дека употребата на лабелите VTORIOT и KRAJ многу ја олеснува работата на програмерот бидејќи на тој начин не мора де се памтат адресите на соодветните локации на кои треба де се изврши скокот.

5.5.3. Програми со циклуси

При креирањето на програмите многу често се јавува потреба од повеќекратно извршување на група од инструкции. Некои видови на микропроцесори дури и имаат посебни инструкции за повторување на одредени програмски секвенци. Мнемоник за овие инструкции е LOOP што преведено на македонски значи јамка, циклус. Циклусите начесто значат враќање назад во програмата и тоа неколку пати. Циклусите бараат употреба на бројач кој всушност покажува колку пати ќе се повтори циклусот. Бројачите се најчесто регистри и тие можат да бројат нанапред или наназад.

Како пример ќе ја објасниме програмата за множење на два броја. Множењето се реализира на тој начин што првиот број го собираме онолку пати колку што изнесува вториот број. Всушност вториот број ќе претставува бројач. После секое додавање на првиот број вредноста на вториот ќе се намалува за еден. Циклусот ќе се повторува се додека вториот број не стане нула. За програмата е многу важно пред да започне додавањето на првиот број резултатот да се изедначи со нула. Ова е прикажано во блок дијаграмот прикажан на слика 5.12.



Слика 5.12. Блок дијаграм за програма со циклус

Подолу е прикажана програмата со циклус. Како и во претходните две програми и овде броевите се пренесуваат од меморијата во регистрите.

```
        MOV R1, адреса1     ;Вредноста на локацијата со наведената адреса
                            ;се копира во регистарот R1.

        MOV R2, адреса2     ;Вредноста на локацијата со наведената адреса
                            ;се копира во регистарот R2.

        SUB R3, R3          ;Одземајќи го сам од себе го ресетираме
                            ;регистарот R3.

BACK:   ADD R3, R1          ;На регистарот R3 го додаваме регистарот R1.
```

```
DEC R2          ;Регистарот R2  го намалуваме за еден.
JNZ BACK        ;Ако резултатот од претходната инструкција не
                ;е нула се враќаме назад, на уште едно
                ;додавање. Ако резултатот е нула одиме на
                ;долната инструкција.
MOV адреса3, R3 ;Вредноста на регистарот R3 го внесуваме во
                ;мемориска локација со адреса3.
```

Програмите кои претставуваат комбинација од линиските програми, програмите за гранење и циклуси се нарекуваат сложени програми.

Заклучоци:

Предности на асемблерот како програмски јазик се: заштеда на мемориски простор, голема брзина на извршување на програмите, директен пристап до хардверот. Недостатоци се: малата програмска библиотека, зависност од видот на компјутерски систем, програмите се долги и грешките се откриваат потешко.

Асемблерската инструкција се состои од 4 полиња и истите се зададени по следниов редослед: поле за лабела, поле за мнемоник, поле за операнди и поле за коментар. Лабелата се користи за именување на некоја инструкција или константа во програмата. Мнемоникот претставува скратеница од англиските зборови со кои симболично се опишува функцијата на инструкцијата и ова поле е задолжителен дел од било која асемблерска инструкција. Операнди се податоци врз кои се извршува назначената инструкција. Коментарот се пишува на крајот од асемблерската инструкција или како независни искази низ програмата. Асемблерските коментари секогаш се одвоени со знакот ; (точка запирка).

Постојат два методи за намалување на битовите со кои се дефинираат операндите. Прв метод е операндот кој се користи почесто да се смести во некој општ регистар, а не во мемориска локација. Втор метод е методот на еднозначно прикажување на повеќе операнди. На пример, во една иста инструкција еден ист операнд може да биде извор на податок и дестинација на добиениот резултат.

Начините на адресирање го објаснуваат начинот на пронаоѓање на саканиот операнд. Кај адресирањето на регистри операндите се наоѓаат во општите регистри на микропроцесорот. Кај непосредното адресирање операндот се содржи во самата инструкција. Зборот непосредно значи

податокот следува веднаш после кодот на инструкцијата. Кај директното адресирање инструкцијата ја содржи адресата на мемориската локација во која се наоѓа операндот.

Најчесто користена инструкција за пренос е MOV дестинација, извор. Дестинацијата и изворот можат да бидат: мемориски локации, регистри или непосредно зададени операнди.

Постојат специјални инструкции за пренос од или кон меморија. Инструкцијата LOAD адреса служи за пренос на податок од мемориската локација со зададената адреса во акумулаторот. Инструкцијата STORE адреса служи за пренос на податок од акумулаторот во мемориска локација со наведената адреса.

Поголем дел од микропроцесорите имаат една инструкција за пренос на податок од влезен уред кон регистар во микропроцесорот (IN) и една инструкција за пренос од регистар кон излезен уред (OUT).

Најчесто користени аритметички инструкции се инструкциите за собирање и одземање. Инструкција за собирање е ADD, а за одземање SUB. Инструкција за множење е MUL што е кратенка од англискиот збор за множење, multiply. Инструкција за делење е DIV што е кратенка од англискиот збор за делење-divide.

Најважни логички инструкции се инструкците: AND (логичко множење), OR (логичко собирање) и NOT (негација или прв комплемент). Некои микропроцесори ги користат и инструкциите XOR (ексклузивно или), NOR (НИЛИ коло) и NAND (НИ коло).

Логичката инструција OR се користи кога треба да се тестира некој бит од податокот. Во маската на позицијата од тој бит ќе биде поставена единица, а на сите останати позиции логичка нула. За издвојување на еден бајт од 16, 32 или 64 битен податок се користи инструкцијата AND и маска составена од нули на позицијата на оние битови кои треба да се изгубат.

ROL е инструкција за ротација налево, а ROR е инструкција за ротација надесно. Ротацијата на битовите може да се изврши неколку пати, а ова се нагласува преку бројот запишан на крајот од инструкцијата. Кај инструкциите за поместување битовите од податокот се поместуваат за неколку места во лево или десно, но оние битови кои ќе го напуштат податокот се губат, не доаѓаат од другата страна на податокот.

Мнемоник за инструкциите за скок е JMP. Скоковите се делат на условни и безусловни. Овие инструкции во себе не содржат ниту еден операнд, туку ја

содржат адресата на мемориската локација на која треба да се скокне. Кај инструкциите за гранење се врши споредба меѓу два операнди и во зависност од добиениот резултат ќе се изврши или не скокот во наведената инструкција.

Кога главната програма содржи потпрограма, доаѓа до прекин на главната програма сè додека не биде завршена потпрограмата. Во овој случај, во програмскиот бројач ќе се смести адресата на првата инструкција од потпрограмата и неговата содржина ќе се зголемува за еден сè додека не биде извршена последната инструкција од потпрограмата. Постојата две инструкции за работа со потпрограми. Едната е за повикување на потпрограма (Call), а другата инструкцијата за враќање од потпрограмата во главната програма (RET).

Линиските програми се составени од инструкции за пренос, аритметички, логички инструкции и инструкции за ротација и поместување. Инструкциите се наредени една врз друга и така се извршуваат.

Програмите со гранење во својот состав содржат инструкции за скок (Jump) и инструкции за гранење (Branch). Овие инструкции се нарекуваат контролни инструкции бидејќи тие можат да го сменат текот на програмата, односно следната инструкција не се наоѓа секогаш под тековната инструкција.

Циклусите значат враќање назад во програмата и тоа неколку пати. Циклусите бараат употреба на бројач кој всушност покажува колку пати ќе се повтори циклусот. Бројачите се најчесто регистри и тие можат да бројат нанапред или наназад.

Прашања и задачи:

1. Асемблерот е програмски јазик најблизок до машинскиот. Објасни!
2. Кои се предности и недостатоци на асемблерот како програмски јазик?
3. Наброј ги составните делови на една асемблерска инструкција и објасни ја нивната намена!
4. Како се бележат операндите прикажани во различни бројни системи?
5. Во кои асемблерски инструкции се користат лабелите?
6. Спореди ги три-адресните и две-адресните асемблерски инструкции!

7. Наброј ги четирите начини за адресирање во асемблерската инструкција!

8. Објасни кои се предности и недостатоци на адресирањето на регистри!

9. Кој начин на адресирање се користи за работа со варијабили?

10. Како се викаат регистрите кои ги чуваат адресите на мемориските локации?

11. Наброј ги групите инструкции на кои е поделено инструкциското множество на општиот микропроцесор!

12. Која е позицијата на изворот и дестинацијата на податокот во Инteловите инструкции за пренос?

13. Кои се специјални инструкции за пренос на податоци од меморија во микропроцесор и обратно?

14. Коментирај ја инструкцијата IN регистар, интерфејс адреса!

15. Каде се наоѓаат операндите во инструкцијата MUL извор?

16. Објасни ја употребата на маски кај логичките инструкции!

17. Наброј неколку инструкции за ротација и поместување!

18. Податокот 1000110011010011
а) ротирај го за три места во лево
б) помести го за четири места во десно

19. Која е разликата меѓу инструкциите за гранење и скок инструкциите?

20. Искоментирај ја инструкцијата BEQZ извор, адреса!

21. Објасни ја примената на стек меморијата кај инструкциите за работа со потпрограми!

22. Кои програми ги викаме изворни, објект програми и извршни програми?

23. Објасни ја структурата на линиските програми, на програмите со гранење и програмите со циклус!

24. Напиши програмска секвенца за одземање на содржината на една мемориска локација од друга!

25. Каква функција има следната програмска секвенца?

```
        MOV R1,80H
POVTORI: ROL R1,1
        JMP POVTORI
```

6. Микроконтролери

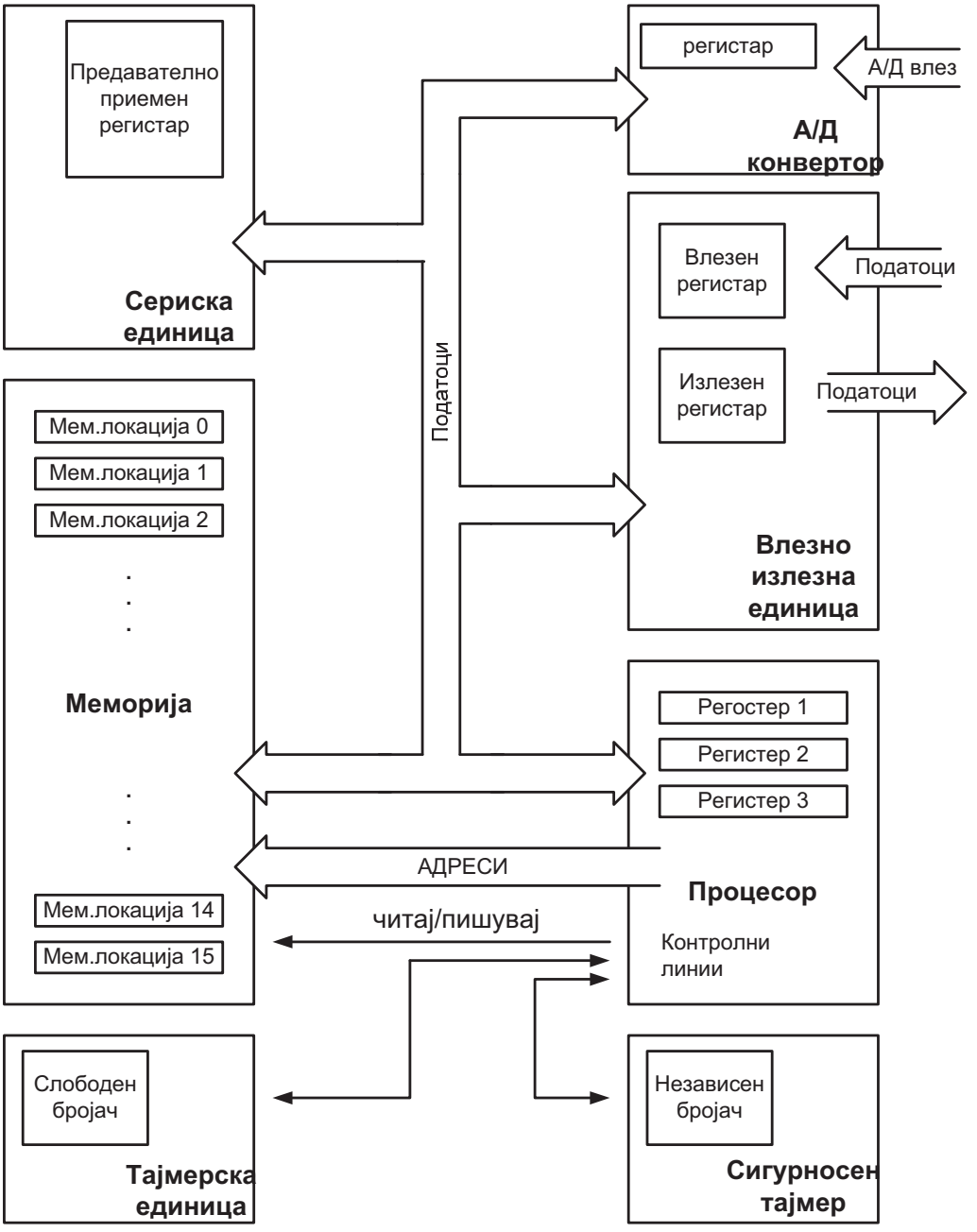
6.1. Вовед во микроконтролери

Постојат голем број на разлики меѓу микроконтролерот и микропроцесорот. Прва и најважна разлика е неговата функционалност. За да микропроцесорот функционира потребно е да му се додадат други компоненти како што се мемориите и влезно излезните уреди. Ова всушност значи дека микропроцесорот е само ,,срцето" на компјутерот. Од друга страна **микроконтролерот е** дизајниран да биде **се во едно**. Со тоа се штеди простор и време потребни за дизајнирање на уредите. Микроконтролерот е **микрокомпјутер во облик на интегрирано коло**. Тој во својот состав содржи сè што има еден сметач: микропроцесор, мемории и бакарни водови за пренос на податоци. Се разбира, хардверот на микроконтролерот е неспоредливо мал во однос на оној што го имаат микропроцесорите. На пример, микроконтролерот PIC16f84 има само 18 пина, капацитетот на RAM изнесува само 68B, работната фреквенција му е околу 4 MHz. Но, големината воопшто не ја ограничува неговата функционалност. Можноста за нивно повеќекратно програмирање, ниската цена, заштедата на простор придонесуваат микроконтролерите да се користат во многу електронски склопови. Примената на микроконтролерите ја среќаваме насекаде околу нас: апаратите во домаќинството, индустријата, автомобилите, играчките итн.

Денес процесното и автоматско управување не може да се замисли без употреба на микроконтролери. Микроконтролерите примаат информации за средината што ги опкружува преку сензорите, приклучени на нивните влезни пинови. Потоа овие податоци се процесираат во микроконтролерот и по извршувањето на програмата се активираат извршните единици приклучени на излезните пинови од микроконтролерот (електромотори, пумпи, дисплеи итн). Кога би го погледнале историскиот развој на електрониката, би можеле да кажеме дека по аналогната електроника следува дигиталната електроника, дигиталната електроника води кон развој на микропроцесорската техника, од неа произлегуваат микроконтролерите, а тие, пак, го условуваат напредокот на роботиката.

На слика 6.1. е прикажана блок шемата на микроконтролер со неговите составни делови и нивната поврзаност.

Исто како кај компјутерите, за **мемориската единица** на микроконтролерите најважни поими се адресирањето и мемориската локација. Мемориските локации се наредени една врз друга и секоја мемориска локација си има свој реден број, кој се нарекува мемориска адреса.



Слика 6.1. Блок шема на општ микроконтролер

Адресирањето не е ништо друго туку избор на една мемориска локација. Контролниот сигнал R/$\overline{W}$ одлучува која операција ќе се извршува, читање од

или пишување во меморијата. Податоците и инструкциите кои се чуваат во меморијата ги обработува централно процесорската единица. Таа во својот состав содржи регистри, аритметичко логичка единица и контролна единица.

За поврзување на меморијата со централно процесорската единица се користат снопови од паралелни бакарни проводници наречени **магистрали**. Бројот на проводници изнесува 8, 16 или повеќе. Податочната магистрала е широка колку и обработуваните податоци, а широчината на адресната магистрала зависи од бројот на мемориски локации.

Микропроцесорот, мемориите и магистралите формираат една функционална целина, но таа нема никава врска со надворешниот свет. Поради тоа на оваа функционална целина се додава уште еден блок составен од неколку мемориски локации, кои од една страна се поврзани со податочната магистрала, а од друга страна со пиновите на микроконтрлерот кои се видливи со голо око. Овие додадени локации се викаат **порти**. По својата природа портите можат да бидат влезни, излезни и двонасочни. Пред да започне работата со портата треба да се избере истата, а потоа да се прочита или запише податокот. Имено, портите се однесуваат како обични мемориски локации со таа разлика што тие директно влијаат врз логичката состојба на пиновите на микроконтролерот.

Сепак и овој начин на комуникација си има свои недостатоци, посебно кога се работи за пренос на податоци на големи растојанија. За да се пренесат осум бита ќе бидат потреби осум линии за пренос со што се губи економичноста. Поради тоа е потребна **единица за сериска комуникација** која овозможува пренос на податоци преку три линии: приемна, предавателна и контролна линија. За да ваквиот концепт функционира мора да се утврдат правила за размена на податоци. Овие правила се викаат протокол. На пример, на предавателната линија е поставена логичка единица се додека не започне преносот. Кога ќе започне преносот, предавателната линија се спушта на логичка нула со што приемната страна ќе знае дека и се испраќа податок. Предавателната страна ги испраќа битовите од бинарниот податок при што претпоставуваме дека за секој бит ќе биде потребно време од еден дигитски такт. После испраќањето на осмиот бит, односно после осмиот дигитски такт предавателната линија повторно се подига на високо ниво, со што се означува крај на преносот. Бидејќи постојат две линии, предавателна и приемна, микроконтролерот ќе може истовремено да прима и да испраќа податоци.

Тајмерскиот блок е уште еден мошне важен дел од микроконтролерот. Тајмерот ни дава информација за изминатото време. Тајмерот всушност претставува **слободен бројач** чија вредност се зголемува во еднакви временски интервали, така да ако се познаваат неговите вредноси во моментите T1 и T2, врз основа на нивната разлика може да се добие информација за поминатото време.

Едно од најважните својства на микроконтролерот е неговата прецизност. Да претпоставиме дека во управувачкиот процес се појавила некоја пречка и нашиот контролер прекинал со извршувањето на програмата или уште полошо, продолжил да работи неточно. За да се надмине овој проблем микроконтролерот треба да се ресетира. Човекот не е во можност постојано да ја надгледува работата на микроконтролерот. За таа цел се воведува еден дополнителен **сигурносен бројач** таканаречен watchdog што во превод значи куче чувар. После секое правилно извршување на програмата таа запишува нула во сигурносниот бројач. Доколку дојде до ,,заглавување" на некое место во програмата, нема да се запише нула, вредноста на бројачот ќе почне да се зголемува и кога ќе ја достигне максималната вредност сам ќе го ресетира микроконтролерот. Со ова сигурносниот бројач ќе предизвика повторно извршување на програмата, но овој пат правилно.

Бидејќи надворешните сигнали битно се разликуваат од оние кои ги користи микроконтролерот (нули и единици) тие треба да се претворат во бинарен облик. Оваа задача ја извршува единицата за аналогно дигитално претварање.

За програмирање на микроконтролерите се користат повеќе програмски јазици од кои би ги издвоиле асемблерот, C и бејзикот. Асемблерот е програмски јазик од понизок ред и програмирањето е многу бавно, но зафаќа помалку место во меморијата и дава најдобри решенија во поглед на брзината на извршување на програмата.

6.2. Микроконтролер PIC 16f84

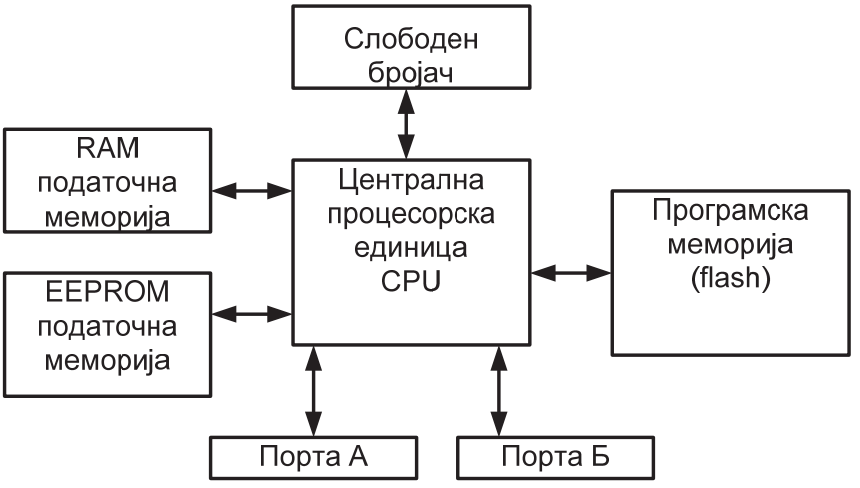
За да бидеме практични, анализата на микроконтролерите ќе ја сведеме на анализа на еден реален и многу често користен контролер PIC16f84. Кратенката PIC има две значења. PIC може да биде кратенка **од Programmable Intelligent Computer**. Но, во некои литератури PIC се толкува и како кратенка од зборовите **Programmable Interrupt Controller** што преведено значи програмибилен контролер на прекини. Сервисирањето на прекините кај микроконтролерот е многу слично со сервисирањето на прекините кај осумбитните микропроцесори. За видовите прекини и за нивната контрола ќе биде објаснето подоцна.

PIC 16f84 работи во **три** различни **мода**: програмски мод, работен мод и мод на мирување. Во програмски мод PIC 16f84 се програмира, односно се внесува корисничката програма во програмската меморија. Постои можност PIC 16f84 да се програмира дури и кога е вграден во некој електричен склоп без потреба истиот да се извади од неговото подножје. Оваа постапка е

позната под името ISCP (In-Circuit Serial Programming). Во работен мод контролерот извршува задачи за кои е и наменет, обработува податоци и управува со разни процеси. Микроконтролерот се поставува во мод на мирување (sleep), со цел да се заштеди во потрошувачката на енергија доколку нема потреба од негово активно работење. По потреба, PIC 16f84 може да се разбуди и да продолжи со извршување на започната програма или комлетно да се ресетира.

На слика 6.2. е прикажана упростена блок-шема на микроконтролерот PIC 16f84 Накратко ќе ги објасниме поважните составни делови.

Централната процесорска единица е ,,срцето" на контролерот. Таа има иста функција како и процесорот во компјутерот и служи за пресметка на адресите на мемориските локации и за декодирање и извршување на инструкциите зададени со програмата. PIC16f84 користи два вида адресирања, директно и индиректно адресирање, но со нив подобро ќе се запознаеме кога ќе ја изучуваме мемориската организација на контролерот PIC16f84. **W е работен регистар** и тој служи како извор на податоци за обработка, но и како дестинација на добиениот резултат после извршената аритметичко логичка операција. Буквата W е прва буква од англискиот збор Working што во превод значи работен. Сите операции, како што се собирањето, одземањето, поместувањето и други, се извршуваат во аритметичко логичката единица.



Слика 6.2. Блок шема на контролер PIC16f84

Корисничката програма што ја извршува микроконтролерот се наоѓа во програмската меморија. Инструкциите една по една се носат во инструкцискиот регистар, потоа во управувачката единица, каде што се декодираат. Декодерот и управувачката единица всушност генерираат контролни сигнали за аритметичко логичката единица и за оспособување на соодветните регистри. Извршувањето на некои инструкции може да влијае врз состојбата на знаменцата во статус регистарот. Дали некое знаменце ќе се активира или не

зависи од добиениот резултат после извршувањето на инструкцијата. Со значењето на битовите од статус регистарот ќе се запознаеме подоцна кога детално ќе ги објасниме сите регистри во микроконтролерот PIC16f84.

EEPROM **меморијата** ги чува податоците и после исклучувањето на напојувањето. Во неа се чуваат податоци добиени од извршувањето на програмата, но и константи битни за самото процесирање. На пример, таков податок би била вредноста на температурата кај температурните регулатори. RAM е привремена меморија и таа ги чува меѓурезултатите и другите податоци кои не се со голема важност и после исклучувањето на напојувањето.

Микроконтролерот PIC16f84 располага со **две влезно излезни порти А и В,** преку кои тој комуницира со надворешниот свет. Подоцна ќе видиме како некои пинови на портите А или В се користат за генерирање на прекин или како влез за слободниот бројач на микроконтролерот.

Со **генераторот на такт и тајмерите** на микроконтролерот ќе се запознаеме подоцна кога ќе го проучиме временскиот систем на микроконтролерот PIC16f84. Да споменеме дека овој микроконтролер користи надворешен извор на дигитски такт (кристален осцилатор или RC осцилатор). Слободниот бројач е 8-битен, го брои секој четврти такт од кристалниот осцилатор и може да достигне максимална вредност од 255. Исто така во состав на тајмерската единица се наоѓа и сигурносниот бројач, watchdog.

Микроконтролерот PIC16f84 може да **се ресетира на четири начини**. Тој се ресетира секогаш после вклучувањето на напојувањето, со цел сите регистри да се доведат во почетна состојба. Микроконтролерот може да се ресетира со доведување на логичка нула на пинот MCLR (microcontroller clear). Ресетирање може да предизвика пречекорувањето на сигурносниот бројач или преминот во мод на мирување. Еден од најважните ефекти на ресетирањето е поставувањето на програмскиот бројач на вредност нула (0000H) со што програмата почнува да се извршува од првата напишана инструкција.

6.3. Мемории на PIC16f84

Капацитетот на мемориите е многу мал: RAM=68B, EEPROM=64B и програмска меморија=1KB. На слика 6.3. е прикажана мемориската организација на микроконтролерот PIC16f84 и мемориските мапи на различните видови мемории.

Сега подетално ќе се запознаеме со функцијата на сите видови мемории и начинот на пристапување до нив.

RAM меморија

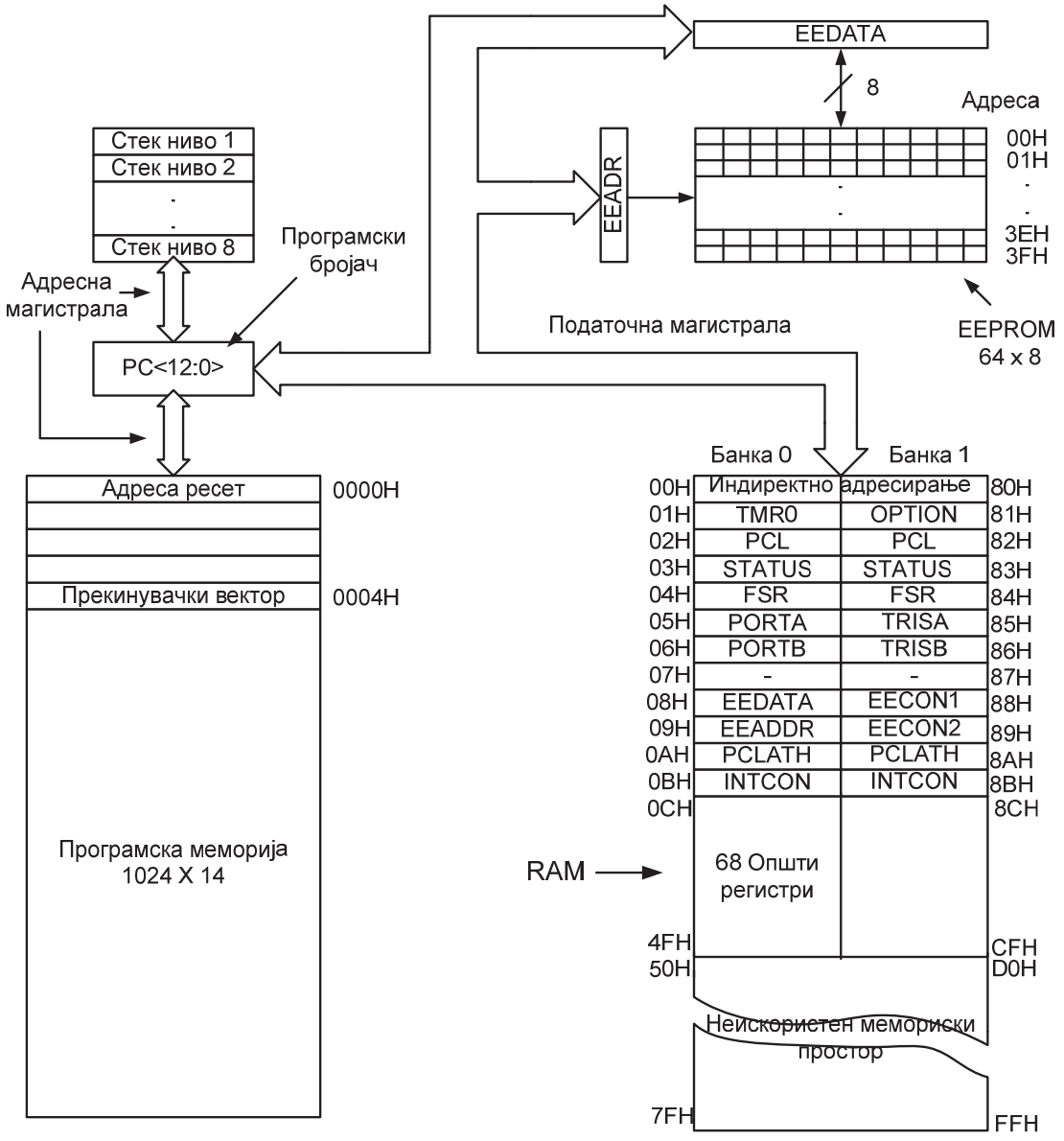
RAM меморијата во микроконтролерите ја има истата функција како и RAM меморијата во сите останати сметачи. Таа е привремена меморија што ги губи податоците по исклучувањето на напојувањето. Таа е **работна меморија**, односно во неа процесорот ги чува сите потребни податоци за извршување на зададената програма. До мемориските локации процесорот пристапува преку испраќање соодветни адреси. Но, во својата организација RAM меморијата на PIC16f84 значително се разликува од RAM меморијата на општиот микропроцесор.

RAM меморијата е поделена во две банки означени со банка 0 и банка 1. На банката 0 и припаѓаат локациите чија адреса почнува на нула, (од 00H до 4F), а на банка 1 локациите со адреси кои почнуваат на еден (од 80H до CFH). Локациите со адресите од 0CH до 7FH и од D0H и FFH, сега за сега, не се користат и со нивно читање секогаш се добива резултат нула. За избор на мемориска банка се користи статус регистарот за што ќе стане збор во понатамошниот текст. Додека е избрана нултата банка не може да се пристапува до локациите од првата мемориска банка.

Кај компјутерите **регистрите со општа и специјална намена** се наоѓаат во микропроцесорот, а кај PIC 16f84 тие **се сместени во RAM меморијата**. Првите 12 мемориски локации од секоја банка на RAM меморијата претставуваат регистри со специјална намена. Регистрите со специјална намена скратено се запишуваат со кратенката SFR (Special Function Registers). На пример слободниот бројач на микроконтролерот (TMR0) претставува мемориска локација од нултата банка со адреса 01H. Статус регистарот може да се најде во двете мемориски банки односно ако се наоѓаме во нултата банка статус регистарот ја претставува локацијата со адреса O3H, а ако се наоѓаме во првата банка тогаш тоа е локацијата со адреса 83H.

Програмскот бројач е 13 битен регистар. Неговите осум понезначајни битови PCL (Program Counter Low) се наоѓаат во локацијата со адреса 04H во нултата банка или во локацијата со адреса 84H во првата банка. Останатите пет позначајни битови се наоѓаат во локацијата со адреса 0AH во нултата банка или локација со адреса 8AH во првата банка. Да се потсетиме дека програмскиот бројач ја содржи адресата на следната инструкцијата која треба да се изврши. Со зголемување на адресата на инструкцијата која моментално се извршува за еден, се добива адресата на следната инструкција. Кога се работи за скок инструкции тогаш програмскиот бројач станува еднаков на адресата наведена во скок инструкцијата. Кога регистрите се присутни во двете мемориски банки, како што се статус регистарот или програмскиот бројач, тогаш е сеедно во која банка се наоѓаме во дадениот момент. Понатаму ќе се

запознаеме со функцијата на сите регистри со специјална намена и значењето на нивните контролни битови.



Слика 6.3. Мемориска организација на PIC16f84

Под регистрите со специјална намена се наоѓаат регистрите со општа намена (GPR-General Purpose Registers). Има вкупно **68 регистри со општа намена**. За пристап до општите регистри сеедно е во која банка се наоѓаме. Ако се наоѓаме во нултата банка тогаш за адресирање на општите регистри се користат адресите од 0CH до 4FH, а ако се наоѓаме во првата банка тогаш тоа се адресите од 8CH до CFH. Понекогаш за пристап до општите регистри се користат симболични имиња. Имено постојат специјални наредби, таканаречени директиви, со кои на секој општ регистар му се задава симболично име. Навистина многу е полесно да се памти име отколку вредност на адреса во хексадецимален броен систем. Со директивите ќе се запознаеме

подоцна кога ќе ја објаснуваме постапката за пишување на програма за контролер PIC16f84.

Програмска меморија

Програмската меморија е трајна флеш меморија и во неа се чува **корисничката програма.** Програмата во програмската меморија се внесува сериски, бит по бит, преку седмиот пин на портата B. За внесување на програмата во PIC16f84, потребен е компјутер, програматор, специјален софтвер (MPLAP, FPP...), кабел за поврзување на програматорот со паралелната порта на компјутерот и корисничка програма.

EEPROM меморија

Втора податочна и трајна меморија е **EEPROM меморијата**. Во неа се чуваат **податоци добиени како резултат од извршувањето на корисничката програма.** На пример, тоа може да бидат податоци добиени од некоја оддалечена метеролошка станица (температура, надворешен притисок, количество врнежи) или податок за бројот на паркирани возила на еден паркинг за едно денонoќие итн. За разлика од програмската меморија, податоците од EEPROM се читаат и се пишуваат паралелно, 8 бита одеднаш. До EEPROM меморијата се пристапува индиректно и се користи посебна процедура со цел да се заштити EEPROM меморијата од случајно, несакани запишувања во меморијата. За читање и за пишување во EEPROM меморијата се користат три специјални регистри. EEADR ја чува адресата на локацијата во која се пишува или чита. EEDATA го чува податокот што треба да се впише или податокот што е прочитан. EECON е 5-битен регистар и содржи контролни битови поврзани со пишувањето и читањето од EEPROM меморијата. Значењето на овие битови ќе биде објаснето подоцна.

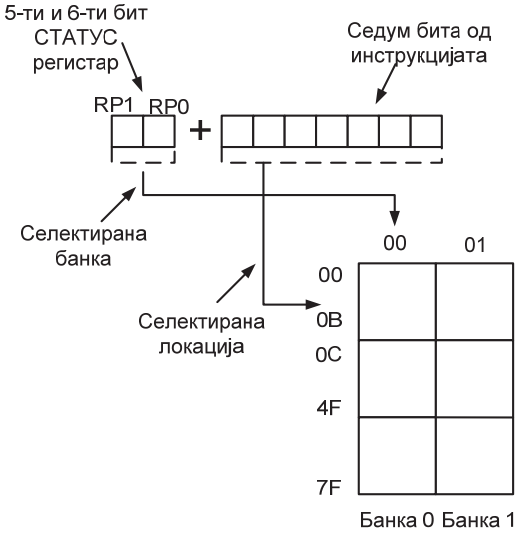
Стек меморија

PIC16f84 има 13-битна стек меморија со 8 нивоа. Негова основна функција е да ја сочува вредноста на програмскиот бројач кога се преминува од главна програма во потпрограма, за да микроконтролерот знае да се врати на местото на кое била повикана потпрограмата. При враќање од потпрограмата во главната програма вредноста од стек меморијата се враќа во програмскиот бројач.

6.4. Начини на адресирање кај PIC16f84

За пристап до локациите од RAM меморијата се користат два начини на адресирање: директно и индиректно адресирање.

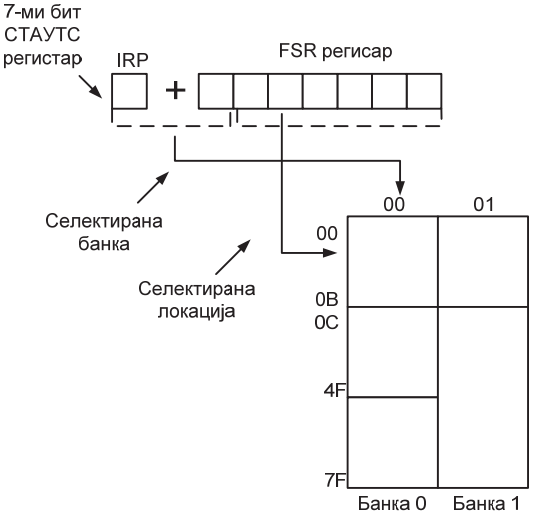
Кај **директното адресирање** адресата за пристап до RAM локацијата се содржи во самата инструкција, исто како кај општиот микропроцесор. На слика 6.4. е прикажана постапката на директно адресирање.



Слика 6.4. Пресметка на адреса кај директно адресирање

Со директното адресирање се добиваат 9-битни адреси. Првите два најзначајни битови се всушност битовите RP1 и RP0 од статус регистарот, кои служат за избор на една од двете мемориски банки. Останатите седум битови се од самата инструкција и служат за избор на една локација од веќе селектираната банка.

Постапката за формирање на мемориска адреса кај **индиректното адресирање** е прикажано на слика 6.5.



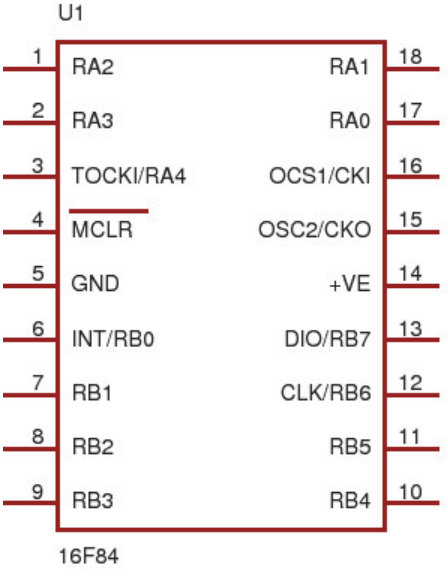
Слика 6.5. Пресметка на адреса кај индиректно адресирање

Кај индиректното адересирање адресата не се содржи во инструкцијата, туку во еден специјален регистар FSR. Овој регистар заедно со седмиот бит од

статус регистарот IRP формираат адреса за селекција на една локација од RAM меморијата. За да го прочитаме саканиот податок не мора да се оди до избраната локација, туку податокот од таа локација може да се прочита од регистарот INDF со адреса 00H во нултата банка или адреса 80H во првата банка. На пример нека општиот регистар со адреса 0FH го содржи податокот 20H. Ова значи дека со примена на индиректното адресирање регистарот FSR ќе ја содржи вредноста 0FH, а регистарот INDF вредноста 20H. Индиректното адресирање има примена при работа со низи.

6.5. Пин-дијаграм на PIC16f84

На слика 6.6. е прикажан пин дијаграм на PIC 16f84.



Слика 6.6. Пин-дијаграм на PIC16f84

Микроконтролерот нема податочни пинови. Но, за комуникација со надворешниот свет ги користи пиновите на портите А и В. Зборот порта претставува група од пинови од микроконтролерот до кои може истовремено да се пристапува и да се поставува саканата комбинација од нули и единици. Физички портата претставува регистар во внатрешноста на контролерот кој со проводници е поврзан со пиновите на контролерот. Портите претставуваат физичка врска меѓу централно процесорската единица и надворешниот свет. Логичката состојба на пиновите може да се прочита од регистрите со специјална намена PORTA и PORTB. На слика 6.6 овие пинови се означени со RA и RB. Буквата R е прва буква од англискиот збор rack што во превод значи оптоварување. **Портата В е осумпинска, а портата А е петпинска.** Тие можат да бидат или влезни или излезни, но не и двонасочни. Дали портите А и В ќе бидат влезни или излезни зависи од битовите во регистрите TRISA и TRISB. **TRISA и TRISB** се осумбитни регистри. На пример, ако TRISB =

00001111, тоа значи дека 4, 5, 6 и 7 пин од портата B се излезни, а 3, 2, 1 и нултиот пин се влезни. Единицата значи дека пинот е влезен, а нулата дека е излезен. Заради полесно памтење бројот нула не асоцира на првата буква од англискиот збор (output), а бројот еден на првата буква од зборот input.

Некои од портите A и B од PIC 16f84 имаат двојна функција.

- Пинот **RA4/TOCKI** е четврти пин на портата A, но може да се користи и како влез за такт на слободниот бројач. Дали овој пин ќе биде четврти пин од порта A или надворешен влез за слободниот бројач зависи од логичката состојба на петтиот бит од регистарот OPTION, TOSC.

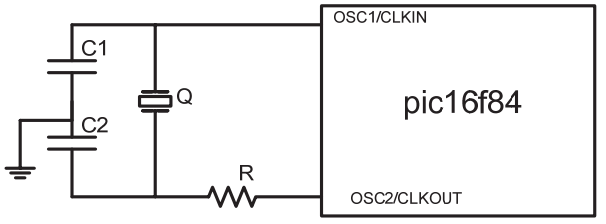
- **RBO/INT** е нулти пин на портата B, но може да се користи и како извор на надворешен прекин. Сите пинови за прекини се објаснети при опишување на функцијата на регистарот INTCON (Interrupt Control).

- Пиновите **RB4, RB5, RB6, RB7** имаат тројна функција. Тие се влезно-излезни пинови на портата B, но можат да се користат и како извори на прекин при промена на логичкото ниво. RB6 и RB7 се користат во програмски мод и тоа на RB6 се доведуваат такт импулси, а RB7 се користи за сериско внесување на програмата во микроконтролерот.

Останатите пинови на микроконтролерот PIC16f84 ја имаат следнава функција.

- $\overline{MCLR}$ се користи за ресетирање на внатрешните регистри кога ќе се постави на ниско ниво. Овој пин се користи во програмски мод на работа, при што се поставува на напон од +12 V до + 14 V.

- За приклучување на осцилатори како извори на такт се користат пиновите **OSC2/CLOCKOUT и OSC1/CLKIN**. На слика 6.7. е прикажано поврзувањето на PIC 16f84 со кристален осцилатор.



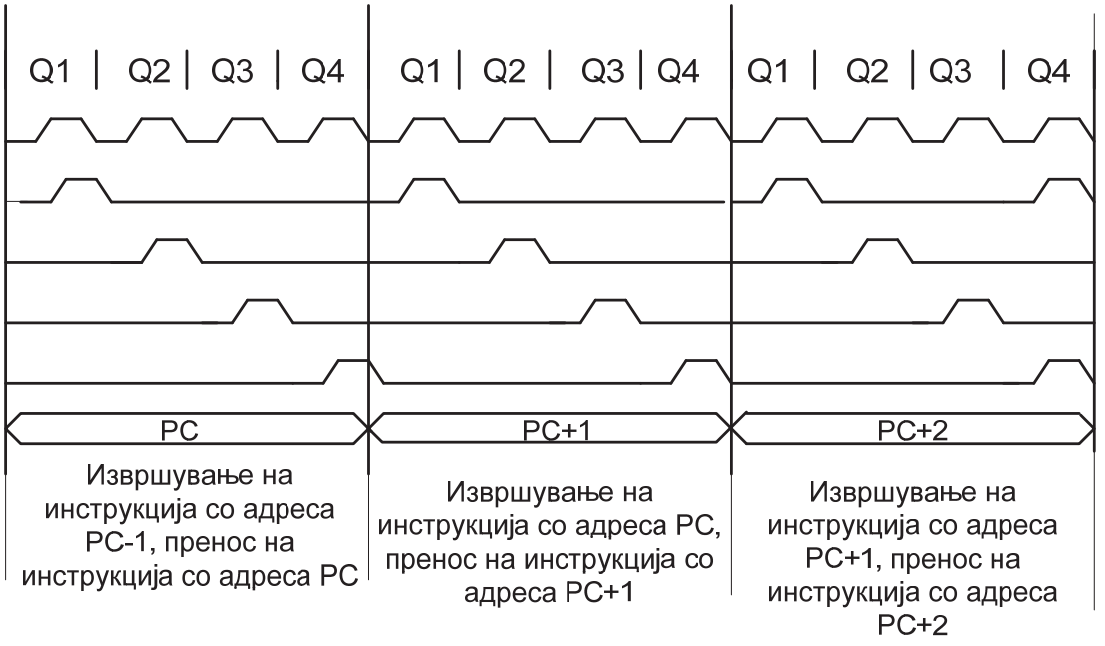
Слика 6.7. Поврзување на PIC16f84 со кристален осцилатор

Микроконтролерот може да користи повеќе видови на осцилатори: LP (low power), XT (crystal resonator), HS (high speed). Како извор на дигитски такт може да се користи и RC осцилатори.

- GND е маса, заземјување, а +Ve е напојување.

6.6. Временски систем на PIC16f84

Микроконтролерот работи во синхрон мод на работа. За да може да ги извршува иструкциите потребен е дигитски такт. Дигитскиот такт се добива од кристалниот осцилатор приклучен на пиновите од микроконтролерот. **За извршување на една инструкција потребни се четири дигитски такта Q1, Q2, Q3 и Q4.** Ова е прикажано на слика 6.8.

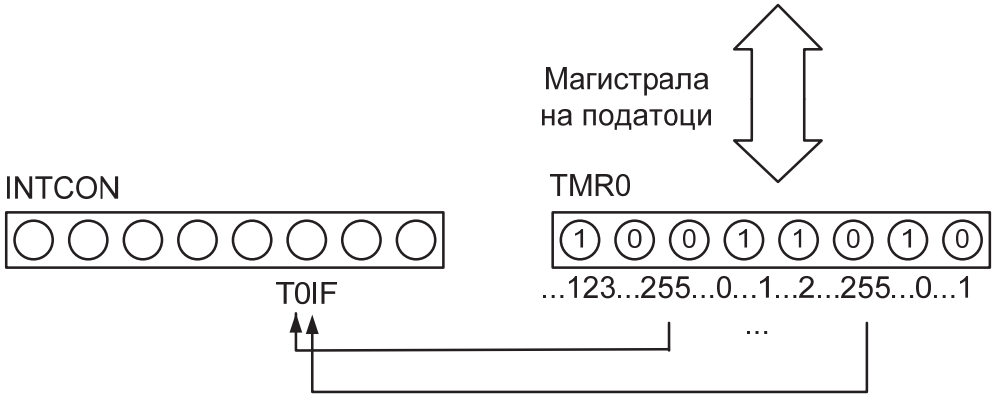


Слика 6.8. Временски дијаграм за извршување на инструкции

За време на првиот дигитски такт Q1 инструкцијата се пренесува од програмската меморија во инструкцискиот регистар. За време на дигитскиот такт Q2 инструкцијата се декодира, а во наредните два такта таа ефективно се извршува. На слика 6.8. се прикажани промените на содржината на регистарот програмски бројач кој ја покажува адресата на следниот бајт кој треба да се пренесе од меморијата во процесорот.

Микроконтролерот PIC-16f84 многу често се користи како тајмер или бројач. **Кога работи како тајмер, тогаш микроконтролерот се поврзува со кристален осцилатор**. Периодата е реципрочна вредност од фреквенцијата на дигитскиот такт (T=1/f). Бројот на дигитски такта што треба да ги изброи тајмерот се добива ако изминатото време се подели со периодата на кристалниот осцилатор. На пример ако фреквенцијата на кристалниот осцилаторот изнесува 1KHz тогаш периодата ќе изнесува 1ms, па за да добиеме време на доцнење од 1s потребно е да поминат 1000 периоди на кристалниот осцилатор, за време од 2s потребно е да се избројат 2000 периоди итн.

Тајмерот на микроконтролерот се бележи со кратенката **TMR0**, што е всушност, кратенка од timer0. Тој е осумбитен регистар и може да изброи од 00H до FFH. Кога ќе ја достигне максималната вредност, тајмерот повторно почнува да брои од нула. Овој премин во броењето се нарекува **пречекорување на слободниот бројач**. Подоцна ќе видиме дека пречекорувањето на бројачот TMR0 може да предизвика прекин (interrupt) во работата на микроконтролерот. Тогаш се активира битот TOIF (Timer0 Interrupt Flag) од регистарот за контрола на прекини INTCON. Со броење на пречекорувањата на слободниот бројач се зголемува максималниот број на избројани дигитски такта. На пример ако бројачот пречекорил 64 пати тогаш велиме дека микроконтролерот изброил 64·256=16384 дигитски такта. На сликата 6.9. е прикажана врската меѓу регистрите TMR0 и INTCON .

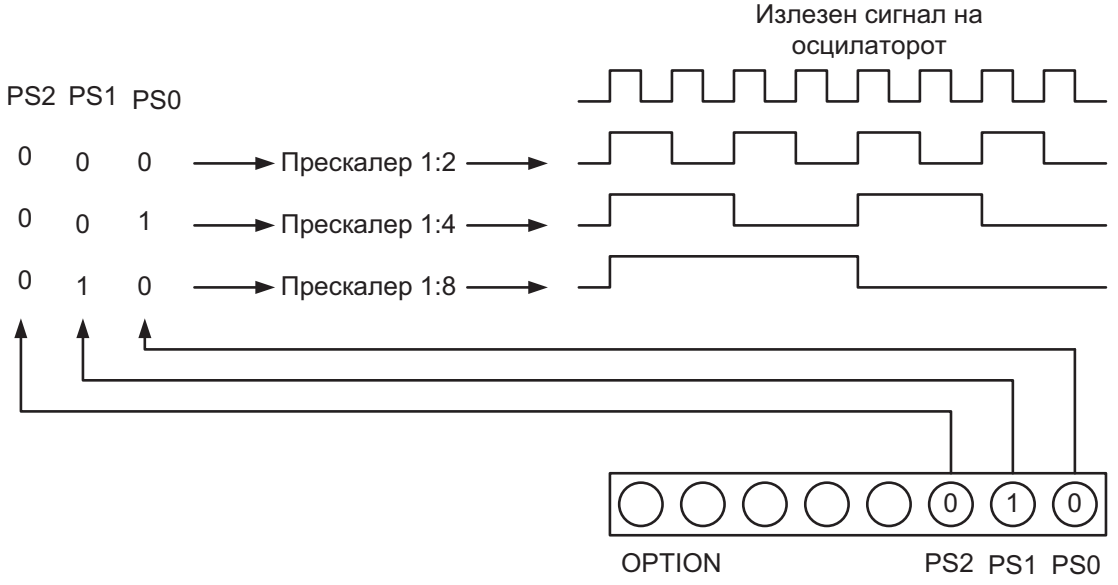


Слика 6.9. Прекин-пречекорување на бројач

Интересно е што слободниот бројач го брои секој четврти импулс од кристалниот осцилатор. Фреквенцијата на слободниот бројач е четири пати помала од фреквенцијата на кристалниот осцилатор. Фреквенцијата на слободниот бројач TMR0 може уште повеќе да се намалува со употреба на специјален делител на фреквенција таканаечен прескалер. Прескалерот е всушност 3-битен код во состав на регистарот за контрола на бројачите OPTION. Во зависност од вредноста на овој 3-битен код делителот може да прима вредности од 2 до 256. Влијанието на прескалерот врз фреквенцијата на слободниот бројач е прикажана на слика 6.10.

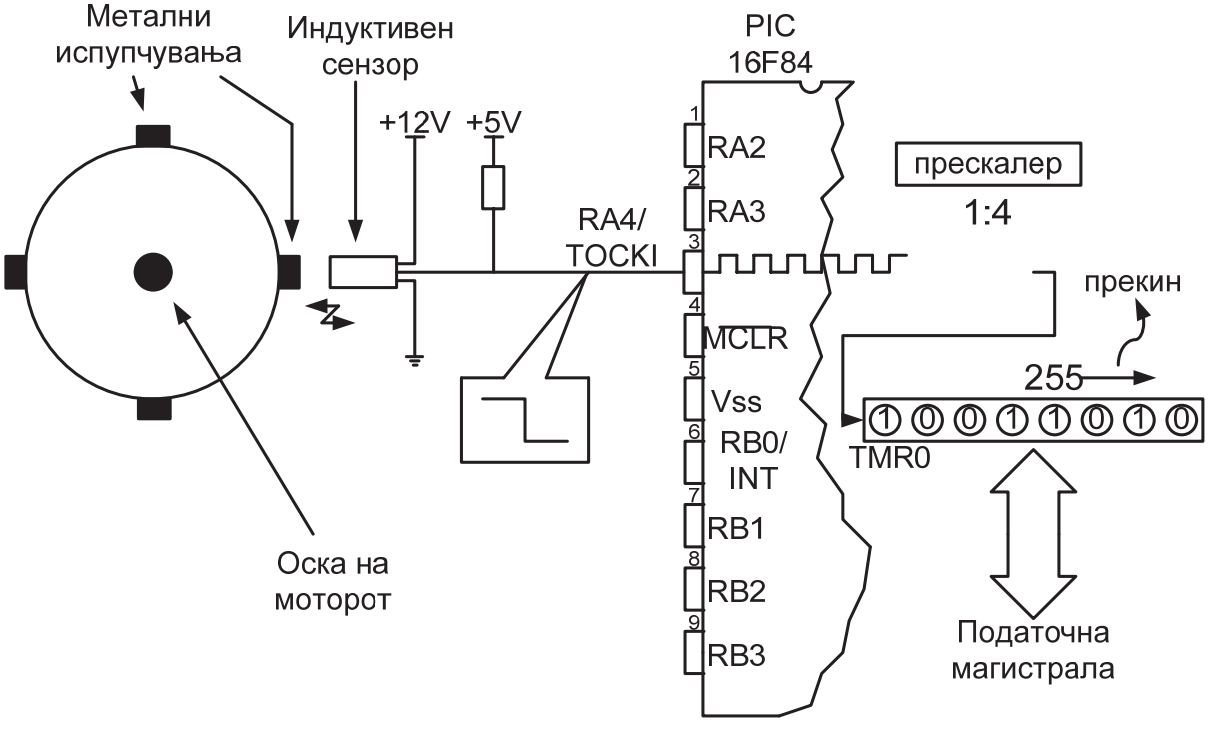
Покрај дигитски тактови од кристалниот осцилатор, регистарот TMRO може да брои и промени на состојбата на пинот **RA4/TOCKI** и тогаш велиме дека микроконтролерот е **бројач на надворешни случајни промени.** Под случајни промени подразбираме промени чие појавување не може да се предвиди. На пример во една минута можат да се појават илјада промени, но во следна минута да не се случи ниту една промена на влезниот напон на пинот RA4/TOCKI. На пример, на пинот RA4/TOCKI може да поврземе оптички сензор и секогаш кога овој сензор ќе се активира, вредноста на регистарот TMRO се зголемува за еден. На ваков начин можат да се избројат број на

влегувања и излегувања на возила на паркинг, број на притискања на тастер итн.



Слика 6.10. Влијание на прескалерот врз бројачот TMR0

На слика 6.11. е прикажан пример за употреба на микроконтролерот PIC16f84 како бројач на вртежи. На пинот RA4/TOCKI е врзан индуктивен сензор. Овој сензор е поставен на неколку милиметри од осовината која се врти.



Слика 6.11. Приклучување на индуктивен сензор на влез од PIC16f84

На оската од моторот се поставени четири метални испупчувања. Кога некое од овие испупчувања ќе се доближи до сензорот, ќе се промени неговата индуктивност, што ќе предизвика пад на напонот на влезниот пин RA4/TOCKI. Од слика 6.11. гледаме дека активатор за овој пин ќе биде опаѓачки раб односно пад на влезниот напон од еден на нула. Секој опаѓачки раб ќе предизвика зголемување на вредноста за слободниот бројач за еден. Кога слободниот бројач ќе ја достигне својата максимална вредност односно ќе изврши 256 броења ќе се појави прекин-пречекорување на бројачот. За да се добие бројот на вртежи во една секунда вредноста на слободниот бројач треба да се подели со изминатото време и бројот четири (поради четирите метални испупчувања).

Покрај слободниот бројач во состав на тајмерската единица се наоѓа уште еден бројач таканаречениот **сигурносен бројач**. Watchdog тајмерот всушност претставува RC осцилатор во склоп на самиот микроконтролерски чип. Ова значи дека сигурносниот бројач ќе работи дури и кога нема да има приклучено осцилатор на пиновите OSC2/CLOCKOUT и OSC1/CLKIN. Да се потсетиме **сигурносниот бројач се ресетира после извршувањето на секоја инструкција**. Ако некоја инструкција не може да се изврши вредноста на сигурносниот бројач ќе почне да се зголемува и кога ќе го достигне својот максимум микроконтролерот PIC16f84 се ресетира. Претпоспоставуваме дека при новото извршување на програмата нема да се појави истата пречка што предизвикало неизвршување на инструкцијата. Постои можност за вклучување или исклучување на сигурносниот бројач преку нултиот бит, WDTE (Watchdog Timer Enable) од таканаречениот конфигурационен збор. Периодата на сигурносниот бројач изнесува 18ms и таа е променлива во зависност од температурата и напојувањето на контролерот. Оваа периода може да се зголемува преку употреба на прескалерот од регистарот за контрола на бројачите OPTION. Изборот на бројач, watchdog или TMR0, кој ќе го користи прескалерот зависи од четвртиот бит во OPTION регистарот, PSA.

6.7. Препознавање на прекини кај PIC16f84

Во секојдневниот живот под прекин подразбираме појава која не спречува да ја завршиме започнатата работа. Прекините (Interrupt) се посебен начин на комуникација на централната процесорската единица со надворешниот свет.

Постојат **прекини чија состојба** процесорот **постојано ја проверува**. Пример за ваков прекин би била комуникацијата со тастатурата. Во текот на извршувањето на програмата, во еднакви временски интервали процесорот

проверува дали е притиснато некое копче од тастатурата. Доколку се утврди дека се случил прекин (притиснато копче) тогаш се повикува потпрограма со која се утврдува кое копче од тастатурата било притиснато.

Некои прекини се сосема случајни настани. Процесорот ја извршува својата тековна програма и доколку се појави некој од предвидените прекини тогаш ја прекинува тековната програма и го обработува прекинот. Обработката на прекинот се врши преку извршување на посебна потпрограма за сервисирање на прекинот. После обработката на прекинот процесорот ја довршува започната потпрограма. Важно е што пред да се случи прекинот, процесорот не превзема никакви дејства за да ја провери состојбата на прекините. Пример за ваков прекин е работата на машината за перење. Таа нема да почне да работи додека не се затвори вратата. Но откако ќе почне со работа состојбата на вратата воопшто не се зема во предвид. Но што ќе се случи доколку некој случајно ја отвори вратата? Се разбира во тој случај машината мора да прекине со својата работа. Но откако ќе се елиминира пречката, откако вратата ќе биде повторно затворена, машината треба да продолжи со својата работа од местото на кое била прекината. Ова значи дека пред да биде исклучена машината за перење мора де се запамети до каде стасала со извршувањето на својата програма.

Микроконтролерот PIC16f84 има **четири вида на прекини:**

- Крај на запишување на податок во EEPROM меморијата.
- TMR0 прекин предизвикан од пречекорување на слободниот бројач.
- Прекин при промена на состојбата на седмиот, шестиот, петтиот и четвртиот пин од портата B.
- Надворешен прекин на пинот RB0/INT.

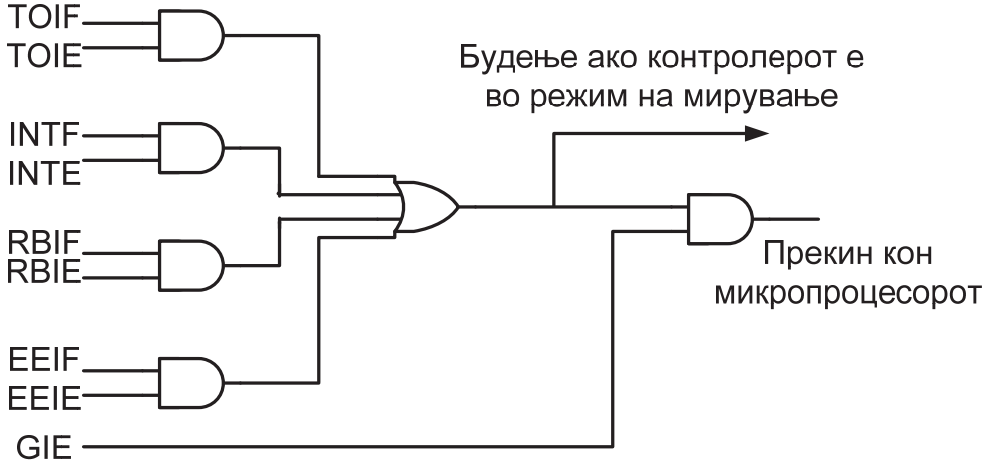
За секој вид на прекин постојат два вида на битови.

-**Interrupt Enable bit** (IE) - софтверски се поставуваат на високо ниво за да се овозможат 4 вида прекини. Софтверско сетирање значи поставување на битот на високо ниво и на ваков начин програмерот избира за работа еден од четирите можни прекини. Доколку некој од овие битови биде поставен на ниско ниво, тогаш и да се случи таков вид прекин, тој нема да биде забележан, извршен.

-**Interrupt Flag bit** (IF) се сетираат хардверски. Микроконтролерот ги поставува на високо ниво за да го извести корисникот дека настанал прекин. После извршувањето на потпрограмамта за сервисирање на прекинот програмерот треба да го ресетира знаменцето за прекин.

Со знаменцата за прекини и битовите за нивно оспособување ќе се запознаеме кога ќе го објасниме регистарот за контрола на прекините INTCON. За сите видови на прекини постои еден заеднички бит GIE (General Interrupt Enable) со кој се врши забрана или оспособување на сите видови прекини одеднаш. Овој бит е многу корисен бидејќи овозможува привремена забрана на сите прекини како не би дошло до прекинување на некоја многу важна програма. Кога ќе се ресетира битот GIE било кој прекин кој останал нерешен се игнорира. Прекините кои останале нерешени, се обработуваат после повторно сетирање на битот GIE. Исто така кога ќе се одговори на некој прекин, GIE битот повторно се ресетира како би се оневозможило појава на некој друг прекин. Микроконтролерот PIC16f84 не може истовремено да обработува повеќе видови на прекини, ниту пак да повикува потпрограми за сервисирање на прекини една во друга. Прекинувачката логика на микроконтролерот PIC16f84 е прикажана на слика 6.12.

За време на прекинот, вредноста на програмскиот бројач се чува на врвот од стекот. Да се потсетиме дека програмскиот бројач ја чува адресата на инструкцијата која требало да се изврши, но не се извршила бидејќи се случил прекинот. Но, чувањето на програмскиот бројач не е доволно бидејќи во прекинувачката програма можат да се користат некои регистри кои се користат и во главната програма. Доколку содржината на овие регистри не се заштити, после завршувањето на прекинувачката потпрограма, главната програма ќе добие регистри со сосема поинакви вредности, што секако ќе предизвика грешка во резултатот. Поради тоа, покрај вредноста на програмскиот бројач, треба да се сочуваат вредноста на работниот регистар W и на статус регистарот.



Слика 6.12. Прекинувачка логика за PIC16f84

За жал не постојат инструкции за внесување и изнесување на податоци во стек меморијата. Наместо да се работи со стек меморија се креира посебна потпрограма за заштита во која се дефинираат нови привремени регистри за чување на старите вредности на регистрите.

6.8. Регистри во PIC16f84

OPTION регистар

За контрола на бројачите и на тајмерот се користи регистарот OPTION. На сликата 6.13. е прикажан OPTION регистарот со кратенките од неговите битови.

7 бит	6 бит	5 бит	4 бит	3 бит	2 бит	1 бит	0 бит
RBPU	INTEDG	TOCS	TOSE	PSA	Прескалер		

Слика 6.13. Распоред на битови во OPTION

Во табелата 6.1. дадени се кратенките на сите битови од OPTION регистарот и нивните функции. Најпрво би почнале со изборот на извор на такт за бројачот TMRO. За ова служи битот **TOCS.** Кога овој бит е единица, се бројат промените на нивоата на пинот RA4/TOCKI. Кога битот TOCS е на нула, се бројат импулсите од осцилаторот. Доколку како извор на такт се избере надворешниот извор приклучен на пинот RA4/TOCKI, треба да се избере кои промени ќе се бројат. Ако битот **TOSE** е еднаков на еден, ќе се бројат опаѓачки рабови, а ако е еднаков на нула, тогаш ќе се бројат растечки рабови. Потоа следува изборот на прескалерот.

бит	Кратенка	Функција
седми	RBPU (RB Pull Up)	Pull up отпорници за портата B 1-исклучени отпорници, 0-вклучени
шести	INTEDG (Interrupt Edge)	Избор на активатор за пинот RA4/TOCKI 1-опаѓачки раб, 0-растечки раб
петти	TOCS (Timer0 Clock Select)	Избор на влезен сигнал за TMR0 1-пинот RA4/TOCKI 0-кристален осцилатор
четврти	TOSE (Timer0 Select Edge)	Избор на активатор за TMR0 1-опаѓачки раб, 0-растечи раб
трети	PSA (Prescaler Select Assignment)	Избор на бројач за прескалерот 0-TMR0, 1-Watchdog
2, 1, 0	3-битен прескалер	Множител на периода или делител на фреквенција

Табела 6.1. Функциски опис на битовите во OPTION регистарот

Трите најнезначајни битови од OPTION регистарот се викаат **прескалер.** Прескалерот го содржи делителот на работната фреквенција. Имено, ако кристалниот осцилатор има работна фреквенција од ред на величина на KHz,

тоа значи дека периодата е од ред на величина на ms. Ова време е нереално за човекот и тоа треба да се зголеми. За таа цел, се користи прескалерот. Вредноста на делителот на фреквенцијата т.е множителот на периодата се добива според табела 6.2.

Прескалер= PS2 PS1 PS0	Делител на фреквенција за слободен бројач	Делител на фреквенција за сигурносен бројач
000	2	1
001	4	2
010	8	4
011	16	8
100	32	16
101	64	32
110	128	64
111	256	128

Табела 6.2. Зависност на делителот на фреквенцијата од прескалерот

Битот **PSA** решава на кој бројач ќе биде доделен прескалерот. Доколку сакаме да работиме со слободниот бројач (TMRO), вредноста на битот PSA треба да биде нула. Два бита од регистарот OPTION не се користат за контрола на тајмерот. Тоа се 6 и 7 бит. Битот **INTEDG** служи за избор на работ на прекин на пинот RBO/INT. Кога овој бит е единица, прекин настанува при растечки раб на овој пин, а кога INTEDG=0, прекин настанува при опаѓачки раб. Пинот RBPU служи за исклучување/вклучување на Pull-Up отпорниците на портата B преку неговото сетирање/ресетирање.

INTCON регистар

За контрола на прекините се користи INTCON регистарот. Тој содржи два вида бита за контрола на прекини. На слика 6.14. е прикажан INTCON регистарот со кратенките на неговите битови.

7 бит	6 бит	5 бит	4 бит	3 бит	2 бит	1 бит	0 бит
GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

Слика 6.14. Распоред на битовите во INTCON регистарот

Во табелата 6.3. дадени се значењата на кратенките за сите битови од INTCON регистарот и нивните функции.

Битот **RBIF** се сетира ако се случи прекин на пиновите RB4, RB5, RB6, RB7 од портата B. За да овие пинови бидат осетливи на прекини истите мора да бидат дефинирани како влезни преку употреба на регистарот TRISB. Битот **INTF** се сетира ако се случи прекин на пинот RB0/INT. За да истиот прекин не

се појави и после враќањето во главната програма, треба да се изврши ресетирање на битот INTF уште додека се извршува потпрограмамта за сервисирање на прекините. Ова програмерот не смее да го заборави бидејќи микроконтролерот постојано ќе ја извршува потпрограмата за сервисирање на прекинот. Битот **TOIF** се сетира ако настане пречекорување на бројачот TMRO.

бит	кратенка	функција
седми	GIE (General Interrupt Enable)	Ги дозволуваме сите видови прекини
шести	EEIE (EEPROM Interrupt Enable)	Дозволуваме прекин за пишување во EEPROM
петти	TOIE (TMR0 Interrupt Enable)	Дозволуваме прекин за пречекорување на TMR0
четврти	INTE (Interrupt Enable)	Дозволуваме прекин на RB0/INT 1-дозволен 0-недозволен
трети	RBIE (Port B Interrupt Enable)	Дозволуваме прекин на 7, 6, 5, 4 пин од портата B 1-дозволен 0-недозволен
втори	TOIF (TMR0 Interrupt Flag)	Појава на прекин-пречекорување на слободен бројач 1-пречекорил TMR0 0-не пречекорил
први	INTF (Interrupt Flag)	Појава на прекин на нултиот пин од портата B (RBO/INT) 1-има прекин, 0-нема прекин
нулти	RBIF (Port B Interrupt Flag)	Прекин на пиновите 7, 6, 5, 4 од портата B 1-има прекин, 0-нема прекин

Табела 6.3. Функциски опис на битовите од INTCON регистарот

Битот **EEIE** се сетира кога треба да се впишат податоци во EEPROM меморијата. Впишувањето на податок во меморијата трае околу 10ms, што за микроконтролерот е прилично многу време. За да не дојде до губење на информација или грешка во тековната програма самото запишување во EEPROM се третира како еден вид прекин.

Битот **GIE** (General Interrupt Enable) служи за глобално овозможување на сите прекини. Доколку микроконтролерот одговори на прекин, т.е. извршува потпрограма за негово сервисирање, битот GIE треба да се ресетира со цел да се спречи појава на каков било нов прекин.

EECON регистар

Овој регистар служи за контрола на читањето и запишувањето во EEPROM меморијата. На слика 6.15. е прикажан овој регистар.

7 бит	6 бит	5 бит	4 бит	3 бит	2 бит	1 бит	0 бит
/	/	/	EEIF	WRERR	WREN	WR	RD

Слика 6.15. Распоред на битовите во EECON регистарот

На табелата е прикажано значењето на сите битови од EECON регистарот.

бит	кратенка	функција
седми	X	Нема значење
шести	X	Нема значење
петти	X	Нема значење
четврти	EEIF (EEPROM Interrupt Flag)	1-пишувањето е завршено успешно 0-уште чита
трети	WRERR (Write Error)	1-грешка при пишувањето, 0-нема грешка
втори	WREN (Write Enable)	1-пишувањето е овозможено 0-пишувањето е оневозможено
први	WR (Write)	1-иницира пишување, 0-не иницира пишување
нулти	RD (Read)	1-иницира читање, 0-не иницира читање

Табела 6.4. Функциски опис на битовите во EECON регистарот

Преку битовите **RD** и **WR** се избира една од двете операции, читање или запишување. Тие се сетираат софтверски, а се ресетираат хардверски по завршувањето на читањето, односно запишувањето. Битот **WREN** (Write Enable) е бит со кој се дозволува (WREN=1), односно се забранува (WREN=0) запишување во EEPROM меморијата. Битот **WRERR** (Write Error) се сетира кога ќе се појави грешка во пишувањето. Битот **EEIF** е бит-индикатор што означува крај на операцијата запишување во EEPROM.

Статус регистар

Статус регистарот служи за избор на една од двете мемориски банки, ја следи состојбата на аритметичко логичката единица и ресет состојбата. На слика 6.16. е прикажан распоредот на битовите во статус регистарот и нивните кратенки. Ресетирањето и сетирањето на битовите може да се врши и софтверски, со инструкции за кои подоцна ќе стане збор. Исклучок се битовите TO и PD чија состојба може да се прочита, но не и да се запише.

7 бит	6 бит	5 бит	4 бит	3 бит	2 бит	1 бит	0 бит
IRP	RP1	RP0	TO	PD	Z	DC	C

Слика 6.16. Распоред на битови во статус регистарот

Во табелата 6.5. е објаснета функцијата на секој бит од статус регистарот.

бит	кратенка	функција
седми	IRP	Се користи како осми бит при индиректното адресирање 1-првата банка 0-нулта банка
6,5	RP1:RP0	За избор на банка при директното адресирање 01-прва банка 00-нулта банка
четврти	TO (Time Out)	Индикатор за пречекорувањето на сигурносниот бројач 1-после вклучување на напојувањето, ресетирање на сигурносниот бројач и премин во мод на мирување 0-после пречекорувањето на сигурносниот бројач
трети	PD (Power Down)	1-после вклучувањето на напојувањето, секое ресетирање или после ресетирањето на сигурносниот бројач 0-после преминот во мод на мирување
втор	Z (Zero)	Се активира при аритметичко собирање или одземање 1-добиениот резултат е нула 0-добиениот резултат е различен од нула
прв	DC (Digit Carry)	1-пренос од трета на четврта позиција при собирање или позајмување при одземање 0-нема пренос
нулти	C (Carry)	1-пренос или позајмување на најзначајниот бит во резултатот при собирање или одземање 0-нема пренос

Табела 6.5. Функциски опис на битовите во статус регистарот

6.9. Инструкциско множество на PIC 16f84

Кога го проучувавме инструкциското множество на 8085 микропроцесорот, инструкциите ги поделивме според функцијата на: инструкции за пренос, логички, аритметички инструкции, ротација, скок-инструкции итн. Оваа поделба нема да ја примениме кај PIC 16f84, туку инструкциите ќе ги поделме на **четири групи:** инструкции за работа со

константи, WF (Working register-File) инструкции, бит-инструкции и контролни инструкции.

Инструкции за работа со константа

Најлесни се инструкциите што работат со константи. Кај нив секогаш дестинацијата е работниот регистар W. На пример, инструкцијата **MOVLW K** значи копирање (поместување) на константната K во работниот регистар W. Буквата L е почетна буква од англискиот збор Literal што значи константа. Во оваа група инструкции се: ADDLW K, SUBLW, ANDLW, IORLW, XORLW. Едниот операнд се наоѓа во регистарот W, другиот е самата константа K, а резултатот се запишува во регистарот W.

Пример 6.1:

```
SUBLW 80H          ;80H- W →W
IORLW 10010111B    ;10010111B логички се собира со содржината на W и
                   ;резултатот се запишува во W
```

WF инструкции (Working registar-File инструкции)

WF инструкциите можат да предизвикаат недоразбирање околу изборот на дестинацијата. Само **MOVWF f** е WF инструкција кај која секогаш дестинација е регистарот f.

Пример 6.2:

```
MOVWF FILE         ;Содржината на W се копира во регистарот познат под
                   ;името FILE.
```

Доколку сакаме да пренесеме податок во спротивна насока од регистарот FILE во работниот регистар W, тогаш се користи инструкцијата **MOVF FILE, W**. Кај сите останати WF инструкции (ADDWF, ANDWF, SUBWF, IORWF, XORWF) дестинацијата може да биде работниот регистар W или некој друг регистар FILE, што зависи од тоа дали по регистарот FILE има запирка и наведена дестинација или не.

Примери 6.3:

```
SUBWF FILE, W      ;FILE - W→W, дестинацијата е W.
SUBWF FILE         ;FILE-W→W.  Дестинацијата е регистарот FILE.
XORWF PRIKAZI      ;Се извршува инструкцијата ексклузивно ИЛИ врз
                   ;содржината на регистарот PRIKAZI и регистарот W и
                   ;резултатот се  запишува во регистарот  PRIKAZI
```

Други инструкции со променлива дестинација се прикажани на табела 6.6.

INCF f,d	(Increase File) Содржината на регистарот f се зголемува за еден и резултатот се сместува во регистарот d.
INCFSZ f,d	(Increase File Skip Zero) Следната инструкција се скока ако се добие резултат нула.
DECF f,d	(Decrease File) Содржината на регистарот f се намалува за еден и резултатот се сместува во регистарот d.
DECFSZ f,d ;	(Decrease File Skip Zero) Следната инструкција се скока ако се добие резултат нула.
COMF f,d	(Complement File) Содржината на регистарот f се комплементира и резултатот се запишува во регистарот d.
SWAP f,d	Позначајните 4 бита од регистарот f си ги заменуваат местата со 4 понезначајни бита од регистарот f и резултатот се запишува во регистарот d.
RLF f,d	(Rotate Left File) Битовите од регистарот f се ротираат за едно место во лево преку знаменцето carry и резултатот се сместува во регистарот d.
RRF f,d	(Rotate Right File) Битовите од регистарот f се ротираат за едно место во десно преку знаменцето carry и резултатот се сместува во регистарот d.

Табела 6.6. Инструкции со променлива дестинација

Уште еднаш да се потсетиме дека доколку не биде наведен регистарот d, тогаш резултатот се запишува во регистарот f. Можеме да кажеме дека резултатот се запишува во последниот регистар што е наведен во инструкцијата.

Бит-инструкции

Со бит-инструкциите се врши сетирање или ресетирање на еден бит од некој регистар. Поради тоа, во инструкцијата мора да се наведат регистарот и редниот број на битот во тој регистар.

Пример 6.4:
BSF FILE, 3 ;(Bit Test File) Се сетира третиот бит од регистарот FILE

Во оваа група инструкции спаѓаат:

BCF Bit Clear File (Ресетирање бит)

BSF Bit Set File (Сетирање бит)

BTFSC Bit Test File Skip if Clear (Ако битот е нула, скокни ја следната Инструкција.)

BTFSS Bit Test File Skip if Set (Ако битот е единица, скокни ја следната Инструција.)

Контролни инструкции

Контролните инструкции можат да го сменат текот на програмата, односно наместо да се изврши следната инструкција, може да биде повикана потпрограма, извршен условен или безусловен скок во зависност од логичката состојба на некој бит или знаменце.

инструкција	коментар	пример
CLRF f	исчисти (ресетирај) го регистарот	CLRF PORTA
CLRW	ресетирај го работниот регистар W	/
CALL address	повикај потпрограма (address е почеток на потпрограмата)	CALL DELAY
GOTO address	скокни на локација со почетна адреса address	GOTO START
RETFIE	враќање од потпрограма за сервисирање прекин	/
RETLW k	враќање од потпрограма со внесување константа во работниот регистар	RETLW 00110110B
RETURN	враќање од потпрограма	/
SLEEP	помини во мод на мирување	/

Табела 6.7. Контролни инструкции

6.10. Поврзување на микроконтролер PIC16f84

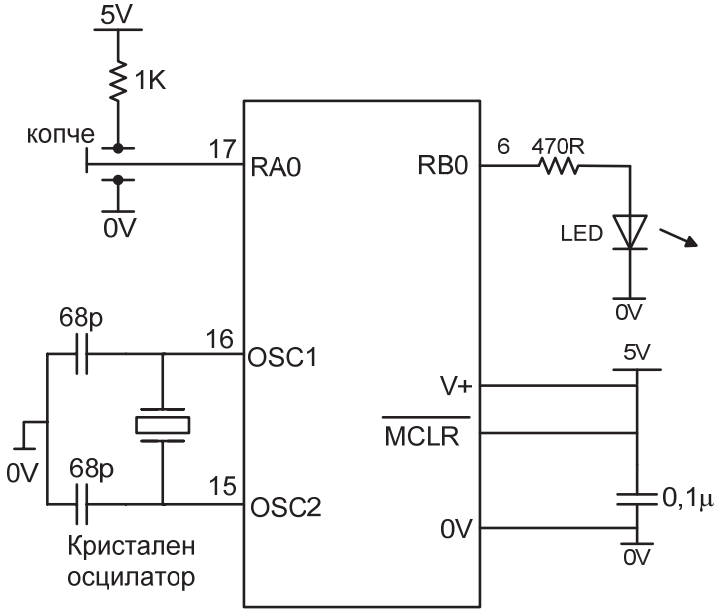
Веќе истакнавме дека микроконтролерот PIC16f84 наоѓа широка примена во процесното управување. Но, за почеток ќе објасниме еден многу едноставен пример. **На нултиот пин на портата А поставуваме копче (тастер), а на нултиот пин на портата В лед диода. Кога копчето ќе биде притиснато, LED диодата ќе свети 1 секунда, а потоа се гасне**. Прво ќе го објасниме начинот на поврзување на микроконтролерот, а потоа пишувањето на програма за овој пример.

На слика 6.17. е прикажан хардверот кој е потребен за овој пример. Кристалниот осцилатор и кондензаторите се потребни за да му дадат дигитски такт на микроконтролерот. Двата кондензатори од по 68pF треба да обезбедат стабилност на системот односно ги елиминираат еднонасочните компоненти. При проектирањето на електричниот уред осцилаторот се поставува што е можно поблиску до микроконтролерот за да не дојде до појава на шум на линиите преку кои микроконтролерот добива дигитски такт. Слична функција

има и кондензаторот поставен меѓу напојувањето на контролерот и масата. Гледаме дека пинот за ресетирање $\overline{MCLR}$ е поставен на високо ниво, со што е оневозможено ресетирање на микроконтролерот преку овој пин.

Излезната струја на пиновите од микроконтролерот изнесува 20mA-25mA што е доволно за побуда на лед диода или помало реле. За поголеми оптоварувања на излез треба да се изврши засилување на струјата преку употреба на транзистор за еднонасочни струи или тријак за наизменични.

За да лед диодата воопшто свети анодата треба да биде на повисок потенцијал од катодата. За да лед диодата максимално свети, без да дојде до оштетување, таа треба правилно да се поврзе. За таа цел сериски со диодата се поврзува отпорник, чија вредност се пресметува така да низ диодата тече максималната струја, која е препорачана од производителите.



Слика 6.17. Поврзување на PIC16f84 со копче и лед диода

Диодите се поврзуваат со микроконтролерите на различни начини. Во случајот прикажан на слика 6.17. катодата на лед диодата е врзана на маса, па за да диодата светне потребно е логичка единица на излезниот пин RB0. На тој начин анодата ќе биде на повисок потенцијал од катодата. Доколку диодата е свртена спротивно, со анодата на маса тогаш таа ќе свети на логичка нула. Покрај со маса лед диодата може да се поврзе и со напојувањето од 5V.

Најчесто на влез од микроконтролерот се поврзува прекинувач. Поврзувањето се врши преку **pull up отпорник** со кој се нагодува влезната струја на микроконтролерот. Кога прекинувачот е затворен до влезниот пин на микроконтролерот се испраќа логичка нула бидејќи тогаш пинот директно се врзува на маса. Кога прекинувачот е отворен тогаш влезниот напон на пинот е на високо ниво, односно пинот се врзува на напонот на напојување преку

заштитен отпорник (pull up отпорник). Прекинувачот има две стабилни состојби, вклучен или исклучен. За разлика од прекинувачот, копчињата од тастатурата имаат една стабилна состојба и една преодна состојба. Кога корисникот ќе притисне некое копче тогаш тој испраќа информација до микроконтролерот, односно микропроцесорот. Но, копчето останува во притисната состојба само една секунда или помалку. После ова копчето се отпушта. Но, времето од една секунда е предолго за микроконтролерот. Тој за време од една секунда може да направи многу проверки на копчето. Поради тоа меѓу две последователни проверки на копчето треба да се внесе **време на доцнење** и на таков начин ќе се избегне едно притискање на копчето микроконтролерот да го изброи многу пати.

6.11. Програмирање на PIC16f84

Откако се запознавме со хардверот на микроконтролерот PIC16f84 и неговото инструкциско множество ќе ја објасниме постапката на програмирање. При програмирањето на микроконтролерот постојат точно дефинирани правила до кои мора да се придржуваме. Заради поголема едноставност програмата ќе ја поделиме на неколку делови:

- Дефинирање на променливи и константи.
- Декларација на микроконтролерот и почетокот на програмата.
- Конфигурација на микроконтролерот .
- Потпрограми за доцнење.
- Иницијализација на портите.
- Пишување на главна програма.

Секој од овие делови ќе биде објаснет во понатамошниот текст.

Дефинирање на променливи и константи

За дефинирање на променливите и константите се **користи асемблерската директива EQU.** Променливите и константите се наоѓаат во регистрите со општа или специјална намена во RAM меморијата. Со директивата EQU на секој од овие регистри се доделува симболично име. Во директивата покрај името на регистарот е наведена неговата адреса во RAM меморијата. На пример, со директивата NIVO EQU 0DH на мемориската локација со адреса 0DH се доделува симболично име NIVO. На ваков начин до променливата од таа локација се пристапува преку името NIVO што е многу повеќе разбирливо за самиот програмер отколку да ја памти хексадецималната адреса на локацијата.

Подоле се дефинирани сите регистри, кои се употребени во програмата за вклучување и исклучување на лед диода преку копче.

```
TMR0      EQU   01H    ;Слободниот бројач е регистар со адреса 01H.
STATUS    EQU   03H    ;Статус регистарот е локација со адреса 03H.
PORTA     EQU   05H    ;Портата A е регистар со адреса 05H.
PORTB     EQU   06H    ;Портата B е регистар со адреса 06H.
TRISA     EQU   85H    ;Регистарот за избор на влезно излезни пинови
                       ;од портата A е регистар со адреса 85H.
TRISB     EQU   86H    ;Регистарот за избор на влезно излезни пинови
                       ;од портата B е регистар со адреса 86H
OPTION_R  EQU   81H    ;Регистарот за контрола на бројачите има адреса
                       81H
COUNT     EQU   0CH    ;COUNT е регистар со општа намена со адреса
                       ;0CH.
```

Од сите наведени регистри само COUNT е регистар со општа намена. Гледаме дека регистрите со специјална намена ги задржале своите имиња, а имињата на општите регистри ги избира програмерот во зависност од нивната намена. Исто така адресите на специјалните регистри се поклопуваат со нивните адреси прикажани на слика 6.3., што е разбирливо бидејќи за секој од овие регистри микроконтролерот си има посебна логика.

Декларација на микроконтролерот и почетокот на програмата

Постојат различни видови PIC микроконтролери. Ќе наброиме неколку: PIC1f83, PIC16f84, PICf872, PIC16C923, PIC17C42A, PIC18C242 и други. Тие се разликуваат по големината на мемориите и бројачите, бројот на пиновите и др. При пишувањето на програмата треба да се наведе видот на микроконтролерот. За ова се користи **директивата LIST.**

Со **директивата ORG** се одредува почетокот на главната програма во програмската меморија.

```
LIST    P=16f84      ;Избираме вид на микроконтролер.
ORG     0            ;Почетната адреса на главната програма е 0.
GOTO    START        ;Скокни на локација со симболично име START.
```

Конфигурација на микроконтролерот

После изборот на микроконтролер можеме да пристапиме кон негово конфигурирање. **Конфигурациониот збор** е 14-битен и со него се дефинира осцилаторот, примената на сигурносниот бројач и тајмерот за вклучување на напојувањето. Битовите од 13 до 4 се викаат битови за кодна заштита на

програмската меморија. Третиот бит се бележи со PWRTE што е кратенка од Power-Up Timer Enable и што во превод значи тајмер за вклучување на напојувањето. Овој тајмер внесува доцнење од 72ms односно за тоа време го држи контролерот во ресетирана состојба се додека не се стабилизира напојувањето. Ако овој бит е еднаков на еден тогаш тајмерот е онеспособен, а ако е еднаков на нула е оспособен. Вториот бит се бележи со WDTE што е кратенка од Watchdog Timer Enable што во превод значи оспособување на сигурносниот бројач. Кога овој бит е единица тогаш сигурносниот бројач работи, а ако е нула тогаш Watchdog тајмерот е онеспособен. Последните два бита, првиот и нултиот го дефинираат **видот на осцилатор** и ова е прикажано во табелата 6.8.

00	RC осцилатор
01	HS осцилатор
10	XT осцилатор
11	LP осцилатор

Табела 6.8.

Директива за конфигурирање на микроконтролерот е **_CONFIG**. Подоле е прикажано конфигурирањето за нашата програма за вклучување и исклучување на лед диода преку копче.

```
_CONFIG H'3FF0'      ;Избран е LP осцилатор, сигурносниот бројач е
                     ;онеспособен, тајмерот за вклучување на
                     ;напојувањето работи, нема кодна заштита.
```

Потпрограми за доцнење

Кога се користи кристален осцилатор како извор на дигитски такт, тогаш слободниот бројач TMRO го брои секој четврти импулс. Ако користиме осцилатор со фреквенција 32KHz=32768Hz, брзината на тајмерот ќе биде една четвртина од 32 KHz, односно 8KHz=8192Hz. За да добиеме време од 1 секунда, бројачот TMRO треба да изброи 8192 импулси. Тоа е прилично многу. Тука го искористуваме прескалерот од регистарот OPTICON. Ако прескалерот е еднаков на 111 тогаш дигитскиот такт со фреквенција 8KHz го делиме со бројот 256 и добиваме вредност 32. Значи за да добиеме доцнење од една секунда, тајмерот треба да изброи 32 пати. Во понатамошниот текст е даден пример за доцнење од 1 секунда.

```
DOCNI   CLR     TMRO            ;TMRO=000000000B
LOOP    MOVF    TMRO, W         ;TMRO се копира во W
```

```
        SUBLW     32.
        BTFSS     STATUS, ZEROBIT
        GOTO      LOOP
        RETLW     0
```

На почетокот од потпрограмата бројачот TMRO се ресетира со цел тој да почне да брои од нула. Додека трае оваа програмска секвенца, вредноста на бројачот постојано се зголемува. Броењето на бројачот TMRO трае сè додека тој не постигне вредност 32. Дали бројачот TMRO постигнал или не вредност 32 проверуваме така што од неговата содржина одземаме 32. Со инструкцијата BTFSS треба да го анализираме знаменцето ZERO. Ако од одземањето добиеме резултат 0, знаменцето ZERO ќе биде 1 (set). Тогаш се прескокнува инструкцијата GOTO и одиме на следната инструкција RETURN (затворање на потпрограмата). Ако резултатот не е еднаков на 0, тогаш знаменцето ZERO е еднакво на 0. Тогаш се извршува инструкцијата GOTO т.е. се повторува циклусот со име LOOP.

Како да направиме доцнење од 5 секунди? Ако за една секунда ни требаат 32 броења, за 5 секунди ни требаат 160 броења. Примерот ќе биде ист како и претходниот само во инструкцијата за одземање имаме SUBLW 160.

Иницијализација на регистри

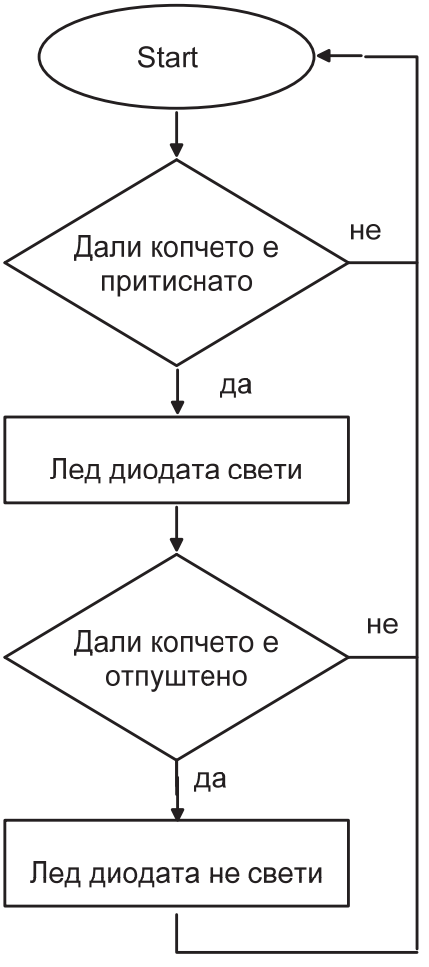
Иницијализацијата на регистрите значи нивно поставување во почетна состојба неопходна за успешно извршување на потпрограмите и главната програма. Прво треба да се изврши **дефинирање на насоката на движење на битовите низ пиновите на портите А и В.** Да се потсетиме, оваа функција ја извршуваат регистрите TRISA и TRISB. Регистрите PORTA и PORTB треба да се ресетираат, односно да се избришат, за да не се случи некоја нивна стара содржина да предизвика грешка во работата на микроконтролерот**. При програмирањето на времето на доцнење употребивме прескалер** со вредност 111. За таа цел треба да се повика OPTION регистарот и неговите три најнезначајни битови да се нагодат на вредност 111. Подолу е програмската секвенца со која е извршена оваа иницијализација на портите и регистрите.

```
START   BSF       STATUS,5        ;Се повикува првата мемориска банка.
        MOVLW     B'00011111'     ;Податокот преку работниот регистар
                                  ;W се внесува во регистарот TRISA.
        MOVWF     TRISA           ;Сите пинови на портата А се влезни
        MOVLW     B'000000000'    ;податокот преку работниот регистар
                                  ;W се внесува во регистарот TRISB.
```

```
MOVWF   TRISB       ;Сите пинови на портата B се излезни.
MOVLW   B'00000111' ;Податокот преку работниот регистар
                    ;W се внесува во регистарот
                    ;OPTION_R.
MOVWF   OPTION_R    ;Прескалер=111.
BCF     STATUS,5    ;Се повикува нултата мемориска
                    ;банка.
CLRF    PORTA       ;Се ресетира регистарот PORTA.
CLRF    PORTB       ;Се ресетира регистарот PORTB.
```

Главна програма

На слика 6.18. е прикажан алгоритамот за програма за вклучување на лед диода со копче.



Слика 6.18. Блок дијаграм за програмата вклучување на лед диода со копче

```
BEGIN   BTFSC  PORTA ,0   ;Проверуваме дали копчето е притиснато. Ако е
                          ;притиснато следната инструкција се
                          ;прескокнува.

        GOTO  BEGIN       ;Ако копчето не е притиснато се враќаме на
                          ;адреса BEGIN и го проверуваме повторно.

        BSF  PORTB, 0     ;Се вклучува лед диодата.

RELEASE BTFSS  PORTA, 0   ;Ако копчето се отпуштило се скока следната
                          ;инструкција.

        GOTO RELEASE      ;Се враќаме на адресата RELEASE, за да
                          ;повторно провериме дали копчето се отпуштило.

        CALL DOCNI        ;Внесуваме доцнење од една секунда.

        BCF PORTB,0       ;Лед диодата се исклучува.

        GOTO   BEGIN      ;Постапката ја повторуваме.

END
```

Заклучоци

Микроконтролерот е микрокомпјутер во облик на интегрирано коло. Тој во својот состав содржи сè што има еден сметач: микропроцесор, мемории и бакарни водови за пренос на податоци.

Кратенката PIC има две значења. PIC може да биде кратенка од Programmable Intelligent Computer. Но, во некои литератури PIC се толкува и како кратенка од зборовите Programmable Interrupt Controller што преведено значи програмибилен контролер на прекини.

PIC 16f84 работи во три различни мода: програмски мод, работен мод и мод на мирување. Во програмски мод се внесува корисничката програма во програмската меморија. Во работен мод микроконтролерот обработува податоци и управува со разни процеси. Микроконтролерот се поставува во мод на мирување (sleep) со цел да се заштеди во потрошувачката на енергија.

PIC16f84 содржи три мемории: RAM, EEPROM и програмска меморија.

EEPROM меморијата ги чува податоците добиени од извршувањето на програмата, но и константи важни за самото процесирање. За читање и за пишување во EEPROM меморијата се користат три специјални регистри. EEADR ја чува адресата на локацијата во која се пишува или чита. EEDATA го чува податокот што треба да се впише или податокот што е прочитан. EECON е 5-битен регистар и содржи контролни битови поврзани со пишувањето и читањето од EEPROM меморијата.

RAM меморијата е поделена во две банки означени со банка 0 и банка 1. На банката 0 и припаѓаат локациите чија адреса почнува на нула, (од 00H до 4F), а на банка 1 локациите со адреси кои почнуваат на еден (од 80H до CFH).

Во микроконтролерот PIC 16f84 регистрите со општа и специјална намена се наоѓаат во RAM меморијата. Првите 12 мемориски локации од секоја банка на RAM меморијата претставуваат регистри со специјална намена.

За комуникација со надворешниот свет микроконтролерот PIC 16f84 ги користи пиновите на портите A и B. Портата B е осумпинска, а портата A е петпинска. Тие можат да бидат или влезни или излезни, но не и двонасочни. Дали портите A и B ќе бидат влезни или излезни зависи од битовите во регистрите TRISA и TRISB.

Микроконтролерот PIC 16f84 многу често се користи како тајмер или бројач. Тајмерот на микроконтролерот се бележи со кратенката TMR0, што е всушност, кратенка од timer0. Тој е осумбитен регистар и може да изброи од 00H до FFH. Кога ќе ја достигне максималната вредност, тајмерот повторно почнува да брои од нула. Кога работи како тајмер, тогаш микроконтролерот се поврзува со кристален осцилатор. Покрај дигитски тактови од кристалниот осцилатор, регистарот TMRO може да брои и промени на состојбата на пинот RA4/TOCKI и тогаш велиме дека микроконтролерот е бројач на надворешни случајни промени.

Микроконтролерот PIC 16f84 има четири вида на прекини: крај на запишување на податок во EEPROM меморијата, прекин предизвикан од пречекорување на слободниот бројач, прекин при промена на состојбата на седмиот, шестиот, петтиот и четвртиот пин од портата B и надворешен прекин на пинот RB0/INT.

За секој вид на прекин постојат два вида на битови: Interrupt Enable bit (IE) и Interrupt Flag bit (IF). Првиот бит софтверски се поставува на високо ниво за да се овозможат прекинот. Знаменцата за прекини се сетираат хардверски. Микроконтролерот ги поставува на високо ниво за да го извести корисникот дека настанал прекин.

За контрола на бројачите и на тајмерот се користи регистарот OPTION. Преку него се дефинира видот на бројач, изворот на такт и делителот на фреквенција (прескалерот).

Кај микроконтролерот PIC16f84 инструкциите за работа со константи завршуваат со буквите LW. Едниот операнд се наоѓа во работниот регистар W, а другиот операнд е самата константа. Резултатот од извршената инструкција се запишува во регистарот W.

MOVWF f е инструкција за пренос на податок при што извор е работниот регистар W, а дестинација е регистарот f. За пренос на податок во спротивна насока, од регистарот FILE во работниот регистар W, се користи инструкцијата MOVF FILE, W.

Со бит-инструкциите се врши сетирање или ресетирање на еден бит од некој регистар. Во инструкцијата мора да се наведат регистарот и редниот број на битот во тој регистар.

За дефинирање на променливите и константите се користи асемблерската директива EQU. Во директивата EQU покрај името на регистарот е наведена неговата адреса во RAM меморијата. Со директивата LIST се одредува видот на микроконтролер. Со директивата ORG се одредува почетокот на главната програма во програмската меморија. Конфигурациониот збор е 14-битен и со него се дефинира осцилаторот, примената на сигурносниот бројач и тајмерот за вклучување на напојувањето.

Микроконтролерот PIC16f84 испраќа сигнали во бинарен облик до LED диодите со времетраење од неколку милисекунди. Програмите за доцнење се користат за да се продолжи времетраењето на побудните сигнали. Овие програми се користат и при поврзување на микроконтролерот со копчиња (тастери) со цел да им се даде доволно време на копчињата да се отпуштат пред да бидат повторно притиснати.

Иницијализацијата на регистрите значи нивно поставување во почетна состојба. Насоката на движење на битовите низ пиновите на портите A и B ја одредуваат регистрите TRISA и TRISB. Регистрите PORTA и PORTB треба да се ресетираат, односно да се избришат, за да не се случи некоја нивна стара содржина да предизвика грешка во работата на микроконтролерот.

Прашања и задачи

1. Наброј неколку примери за практична примена на PIC16f84!

2. PIC16f84 може да работи во три мода: програмски, работен и мирување. Објасни што значи секој од нив!

3. Во кој дел од микроконтролерот PIC16f84 се наоѓаат регистрите со специјална намена?

4. Колку изнесува максималната големина на корисничката програма што може да се внесе во програмската меморија на PIC16f84?

5. Објасни што е потребно за да се испрограмира еден микроконтролер!

6. Кои податоци се чуваат во EEPROM меморијата?

7. По што се разликуваат адресите на локациите од нултата и првата мемориска банка?

8. Каде се наоѓа адресата на мемориската локација кај директното и индиректното адресирање кај PIC16f84?

9. Опиши ги портите A и B на микроконтролерот PIC16f84! Објасни за што служат тие?

10. За што служат пиновите OSC1 и OSC2?

11. Објасни каква функција има регистарот TMRO?

12. Пинот RA4/TOCKI има двојна функција. Објасни!

13. Што претставува прескалерот и каква функција има истиот?

14. Објасни ја функцијата на сигурносниот бројач!

15. Микроконтролерот PIC16f84 има 4 вида прекини. Набројги!

16. Објасни каква функција имаат регистрите TRISA и TRISB?

17. Кој регистар се користи за контрола на прекините?

18. Објасни ги термините хардверско и софтверско ресетирање на битови во контролните регистри?

19. Битот INTEDG е кратенка од Interrupt Edge. Објасни за што служи овој бит!

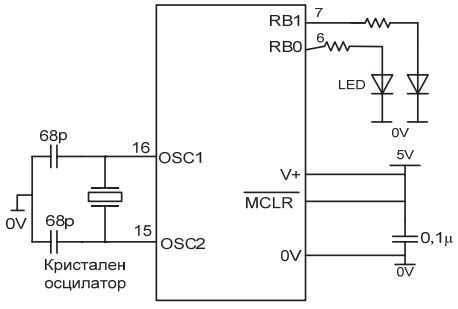
20. Битот EEIF е кратенка од EEPROM Interrupt Flag. Објасни кога се сетира овој бит?

21. Спореди ги инструкциите MOVF FILE и MOWF FILE!

22. Искоментирај ги инструкциите:
    CLRF
    BCF
    INCF

23. Искоментирај ги инструкциите:
    RLF COUNT ,W
    DECSZ COUNT
    ADDWF COUNT
    BTFSS PORTB,3
    INCFSZ TMRO

24. На нултиот и на првиот пин од портата B се приклучени две лед диоди со анодите свртени кон пиновите. Објасни го ефектот од следнава програмска секвенца:




```
BEGIN   BSF PORTB, 0
        BCF PORTB, 1
        CALL DELAY5
        BCF PORTB, 0
        BSF PORTB, 1
        GOTO BEGIN
```

25. Колку изнесува работната фреквенција на бројачот, ако микроконтролерот PIC16F84 е приклучен на кристален осцилатор со f=200KHz и прескалерот има вредност 011? Колку такта треба да се избројат за да се добие доцнење од 0,5 секунди?

26. Зошто е потребно да се воведе време на доцнење меѓу две последователни притискања на копчето?

27. За што служи директивата EQU?

28. Која директива се користи за избор на вид на микроконтролер?

29. Што се дефинира со конфигурационият збор?

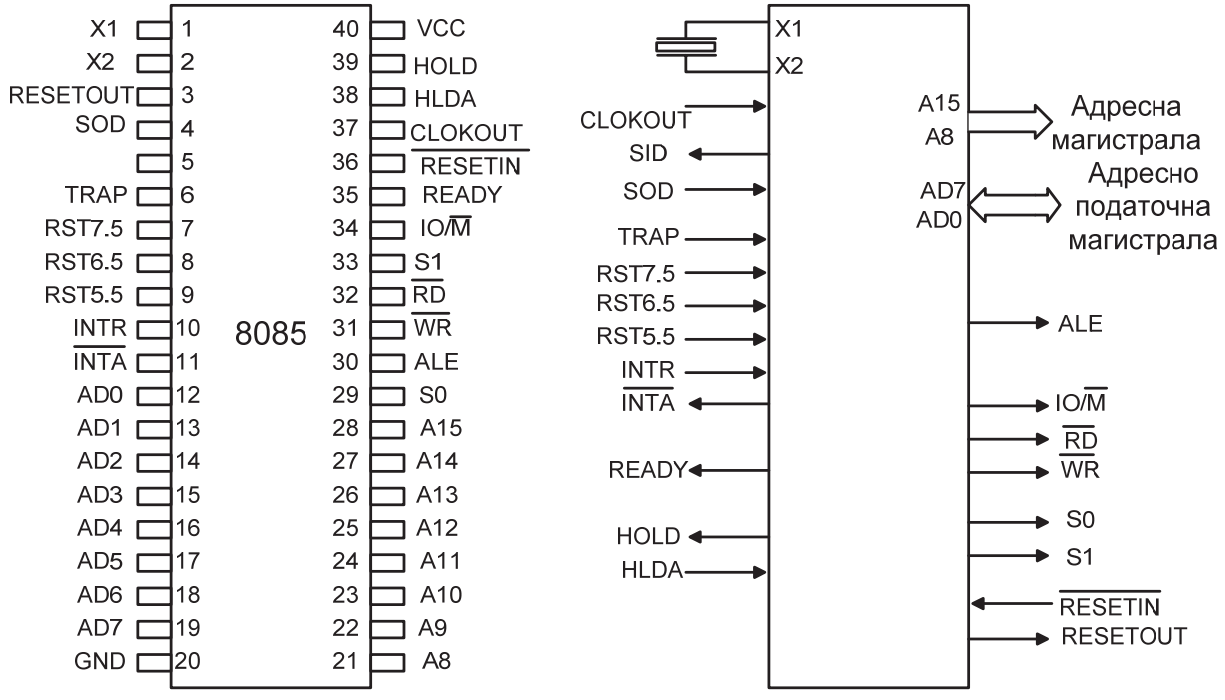
30. На осумте пина од порта B се приклучени осум лед диоди со анодата свртени кон пиновите. Објасни каков е ефектот од следнава програмска секвенца:

```
POCETOK   MOVLW 11110000B
          MOVWF PORTB
          CALL DALAY5
           MOVLW   00001111B
           MOVWF PORTB
           CALL DELAY 5
           GOTO BEGIN
```

7. Осумбитни микропроцесори (микропроцесор 8085)

7.1. Пин-дијаграм на микропроцесор 8085

На слика 7.1 е претставен пин дијаграм на микропроцесорот 8085. Стрелките што влегуваат во пинот се влезни сигнали, а оние што излегуваат се излезни сигнали (од микропроцесорот кон останатите делови на сметачот).



Слика 7.1. Пин дијаграм на микропроцесор 8085

Некои пинови се активни на високо ниво, а некои на ниско ниво. Пиновите активни на ниско ниво се обележани со негација врз нивната кратенка. Покрај овие две состојби, постои и трета состојба наречена состојба

на висока импеданса. Во ваква состојба пинот не реагира на никаква побуда, односно тој е блокиран.

A_{15}-A_8 Адресна магистрала
Преку неа се пренесуваат осумте позначајни битови од 16- битната адреса. Оваа магистрала се наоѓа во состојба на висока импеданса за време на состојбата **HOLD** и инструкцијата **RESET**.

AD_{0-7} Мултиплексирана адресно/податочна магистрала
За време на првиот такт магистралата се користи за пренос на првите значајни 8 адресни бита, а во вториот и третиот такт на машинскиот циклус се испраќаат податоците.

ALE Address Latch Enable-контролен сигнал за AD магистралата
Ако ALE=1 тогаш адресно-податочната магистрала пренесува адресни битови. Ако ALE=0 тогаш магистралата пренесува податочни пинови.

S0,S1, IO/ $\overline{M}$ Статус-пинови
Под машински циклус се подразбира времето потребно да се пренесе информација од еден бајт од микропроцесорот до меморијата или обратно. На почетокот на секој машински циклус се активираат статус-сигналите. Преку трите статусни линии микропроцесорот дава информација за тоа каков машински циклус ќе се реализира. Разликуваме шест видови машински циклуси.

IO/$\overline{M}$	S1	S0	Машински циклус
0	0	1	Пишување во меморија
0	1	0	Читање од меморија
1	0	1	Пишување во излезен уред
1	1	0	Читање од влезен уред
0	1	1	Пренос на операциски код
1	1	1	Обработка на прекин

Статусните линии се активираат на почетокот на машинскиот циклус и остануваат во таа состојба до неговиот крај.
Инструкцискиот циклус е составен од три до четири машински циклуси, во зависност од тоа колку бајти треба да се пренесат за да целосно се изврши инструкцијата.

$\overline{RD}$ Пин за читање (Read)
Микропроцесорот го испраќа контролниот сигнал $\overline{RD}$ до меморијата или периферните уреди кога треба да се прочита податок од нив. Овој пин е активен на логичка нула.

$\overline{WR}$ Пин за пишување (Write)
Има обратно значење од $\overline{RD}$ и се активира кога микропроцесорот треба да запише податок во мемориски или периферен уред.

READY Кога овој контролен сигнал е на високо ниво, тоа значи дека меморијата или периферниот уред е спремен за впишување или за читање податоци. Ако овој сигнал е на ниско ниво, тогаш централниот микропроцесор ќе чека цел број на тактови за да сигналот READY се подигне на високо ниво пред да се заврши читањето од или запишувањето во меморијата или периферниот уред.

HOLD Пин за барање директен пристап до меморија
Овој влез се активира (се крева на високо ниво) кога некој периферен уред бара директен пристап до меморијата (Direct Memory Access). Тогаш централниот процесор не е господар, туку е роб. Тој не може да ги користи магистралите за трансфер на податоци, но може да го искористи времето за да заврши некоја интерна работа. Додека трае ваквата состојба адресните, податочните, $\overline{RD}$, $\overline{WR}$ и IO/$\overline{M}$ пинови се наоѓаат во состојба на висока импеданса.

HLDA Hold Acknowledge-Пин за дозвола за директен пристап до меморија
Микропроцесорот го прифатил барањето за директен пристап до меморија и кога овој сигнал ќе се крене на високо ниво, микропроцесорот ќе ги ослободи магистралите на некој периферен уред поради многу брз трансфер на податоци.

INTR Барање за прекин (Interrupt Request)
Микропроцесорот проверува дали овој пин е активен или не по завршувањето на секој инструкциски циклус. Овој пин е влезен за микропроцесорот.
Под рестартирачка адреса се подразбира адресата на мемориската локација од која започнува потпрограмата за сервисирање на активниот прекин. INTR нема фиксна рестартирачка адреса. Која прекинувачка потпрограма ќе ја изврши микропроцесорот, зависи од прекинувачкиот вектор што го испраќа периферниот уред кој бара прекин. Бројот на прекинувачки вектори е еднаков на бројот на периферни уреди што можат да иницираат прекин.

$\overline{INTA}$ Пин за дозвола за прекин (Interrupt Acknowledge)
Овој пин се активира кога микропроцесорот ќе го прифати

барањето за прекин. Притоа доаѓа до активирање на прекинувачкиот чип 8259 или на некоја друга прекинувачка порта.

TRAP Прекин од највисок приоритет.
Прекинот TRAP е со највисок приоритет. Откако ќе се активира овој пин, микропроцесорот мора да го изврши барањето за прекин. TRAP не може да се маскира, ниту да биде отфрлен. Зборот trap во превод значи стапица, замка. Тој се применува во случај на катастрофални грешки, како на пример, проблеми со напојувањето или измешан пренос на податоци по магистралата. Микропроцесорот го препознава прекинот TRAP само ако по скокот од ниско ниво на високо (растечки раб) активаторот остане во таква состојба најмалку 160 микросекунди. Микропроцесорот нема да го прифати барањето за прекин ако активаторот скокне на високо ниво, па повторно падне на ниско за да подоцна по вторпат скокне на високо ниво.

RST 5.5 Рестартирачки прекини (Restart)
RST 6.5 По извршувањето на потпрограмата за сервисирање на прекинот,
RST 7.5 микропроцесорот не се враќа назад на прекинатата програма, туку таа почнува одново. Рестартирачката адреса е фиксна за прекините TRAP и RESET.
RST 6.5 и RST 5.5 се маскирачки прекини. За нивно маскирање и читање се користат инструкциите SIM (set interrupt mask) и RIM(read interrupt mask). За да микропроцесорот ги препознае барањата за прекин на пиновите RST 6.5 и RST 5.5 активаторот мора да е на високо ниво сè додека барањата не бидат прифатени.

RESET IN Кога ќе се активира овој пин, доаѓа до поставување на програмскиот бројач во положба нула и ресетирање на знаменцето за прекин и на HLDA флип- флопот. Во одредени случаи може да дојде и до промена на состојбата на статус-регистарот и на регистрите со општа намена.

RESET OUT Преку овој сигнал микропроцесорот ги известува останатите склопови во сметачот дека бил ресетиран.

CLK Дигитски такт (низа од периодични правоаголни импулси) што го дава генераторот на такт. Работната фреквенција добиена на излез од микропроцесорот е двапати помала од фреквенцијата на кристалниот осцилатор приклучен на пиновите X1 и X2.

X1,X2 Пинови за влез на кристалниот осцилатор

X1 и X2 се по фаза спротивни сигнали (додека едниот е на високо ниво, другиот е на ниско) и се користат за внатрешен тајминг на самиот микропроцесор.

SID Влезен пин за сериска комуникација (Serial Input Data)

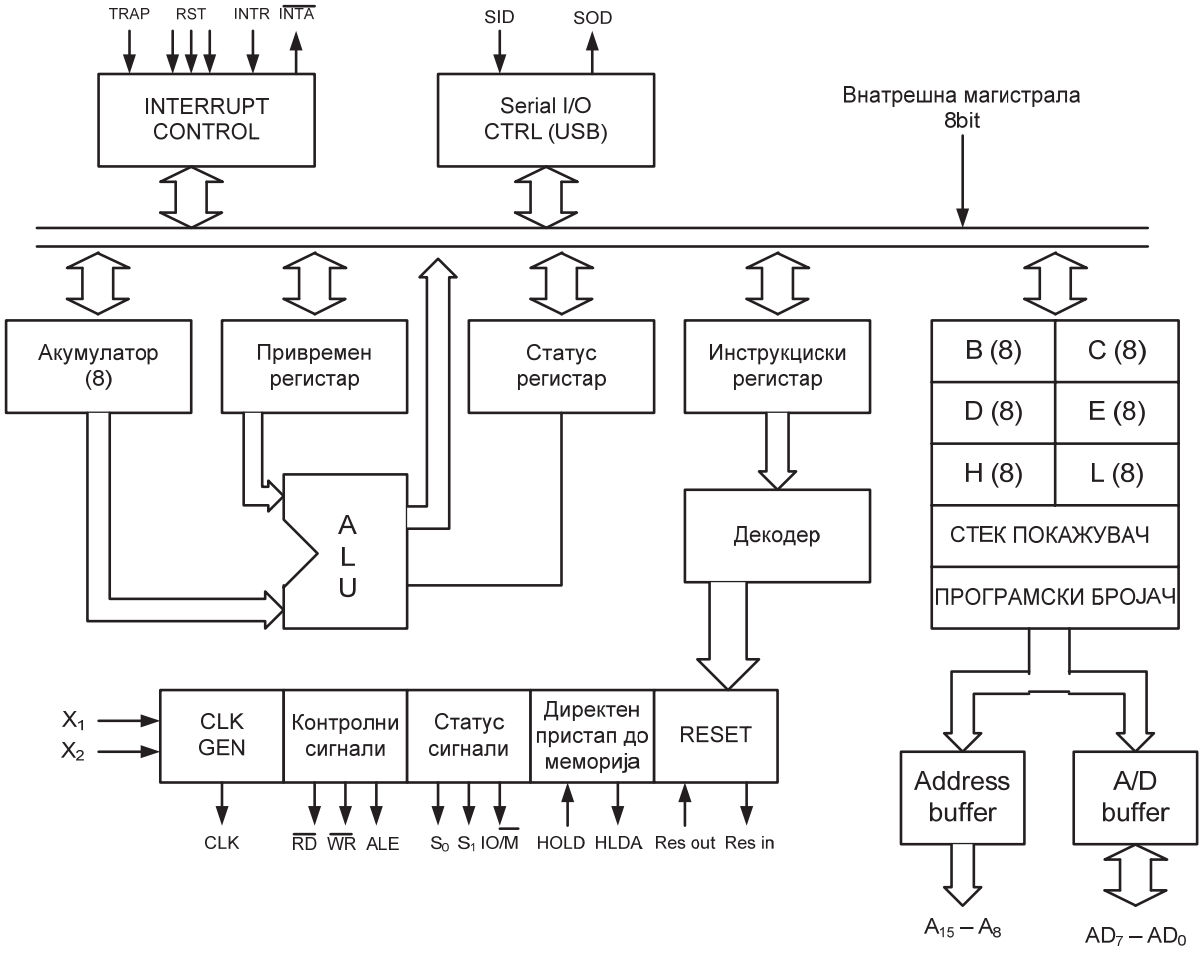
SOD Излезен пин за сериска комуникација (Serial Otput Data)

7.2. Архитектура на микропроцесор 8085

Интеловиот микропроцесор **8085** излезе на пазарот во 1977 год. Овој микропроцесор бара многу мал број хардверски компоненти, но нивната поставеност и функционалност може да обезбеди многу голема ефикасност во работата.
Компанијата „Интел“ има произведено над 200 милиони вакви чипови. Компанијата „Зилог“ е втора компанија што има продадено над 500 милиони микропроцесори, кои се речиси идентични со 8085, но се продаваат под името Z-80. Микропроцесорот 8085 по својата организација е многу сличен со микроконтролерите, како што е PIC 16f84, кои наоѓаат голема примена во процесното управување. По својата моќ 8085 не може да се споредува со микропроцесорот Пентиум, но тој уште долго време ќе се произведува бидејќи наоѓа голема примена во поедноставните електронски уреди, кои и не бараат толку моќни микропроцесори. На слика 7.2. е прикажана функционалната блок шема на микропроцесорот 8085, со сите негови општи и специјални регистри. Микропроцесорот 8085 е осумбитен микропроцесор, односно обработува податоци со големина од 1 бајт. Располага со генератор на такт кој произведува дигитски такт со **фреквенција 8 MHz**, посебен контролер за работа со прекини и контрола на тајмингот. Има пристап до **меморија** со капацитет **64KB**, односно има **16 адресни пина** наменети за директно адресирање на меморискиот простор (2^{16} =64K).

Регистрите се внатрешна меморија на микропроцесорот, но имаат мал капацитет. Податоците што ги обработува микропроцесорот се сместени во последователни мемориски локации. Микропроцесорот ја испраќа адресата на првиот податок до меморијата, а таа му го испраќа саканиот податок. Кога микропроцесорот ќе го обработи првиот податок, го повикува следниот итн. Резултатот од извршената инструкција може да се смести на различни места. Ако добиениот резултатот треба уште еднаш да се обработи тогаш тој ќе се смести во акумулаторот. Ако добиениот резултат треба да се обработи во блиска иднина, но не во следниот чекор, тогаш тој се сместува во некој од регистрите на микропроцесорот. Исто така, ако се работи за завршен резултат

или ако тој ќе се обработува многу подоцна, тогаш резултатот може да се смести назад во меморијата.



Слика 7.2. Архитектура на микропроцесор 8085

Акумулаторот е работен регистар на микропроцесорот и се наоѓа на влез од аритметичко логичката единица. Покрај акумулаторот во **регистри со општа намена** влегуваат уште шест други регистри, кои се бележат со буквите B, C, D, E, H, L. Овие регистри можат да соберат 8 бита и да бидат извор на податоци или цел за добиениот резултат. Секој од овие регистри се идентификува со комбинација од 3 бита што е прикажано во табелата 7.1.

A	111
B	000
C	001
D	010
E	011
H	100
L	001

Табела 7.1. Кодови за општи регистри на микропроцесорот 8085

Често пати општите регистри се користат во пар за сместување и чување на 16-битни адреси. Во овој случај се користат следниве ознаки: BC, DE, HL. Регистрите B, D и H ги чуваат битовите со поголема тежина, а регистрите C, E и L битовите со помала тежина.

Програмата претставува множество од бајти. Мемориските локации во кои се сместени бајтите не се расфрлани случајно низ целата меморија, туку се наредени во последователни мемориски локации. На пример, адресата на петтиот бајт ќе ја добиеме ако на адресата на четвртиот бајт додадеме еден. При извршувањето на една програма, микропроцесорот ги повикува еден по еден, онака како што се наредени во меморијата. **Програмскиот бројач** е регистар во микропроцесорот, кој ја памти адресата на следниот бајт што треба да се пренесе од меморијата и да се обработи. Микропроцесорот додава еден на содржината на програмскиот бројач по секое донесување нов бајт од меморијата до микропроцесорот. Програмскиот бројач ја содржи адресата на следниот, а не на бајтот што моментално се обработува во микропроцесорот. Исклучок од ова правило имаме кај скок-инструкциите и кај инструкциите за повикување потпрограми. Доколку се изврши скок-инструкција, тогаш содржината на програмскијот бројач нема да се зголеми за еден, туку тој ќе ја прими вредноста на адресата наведена во скок-инструкцијата. Кај скок-инструкциите микропроцесорот не се враќа назад на локацијата од која е извршен скокот.

Стек меморијата се користи за работа со потпрограми. Стекот е посебно организиран мемориски простор. Тој е составен од поголем број мемориски локации. Стек меморија се полни и се празни преку врвот. Секој нов податок се запишува на врвот на стекот. Ако сакаме да земеме некој податок сместен во средината на стекот, треба да ги испразниме сите мемориски локации над саканиот податок, со што автоматски саканиот податок (мемориска локација) ќе стаса на врвот од стекот. Микропроцесорот содржи посебен регистар **стек-покажувач** (stack pointer), кој е 16-битен регистар и ја содржи адресата на меморискатa локација на врвот од стекот.

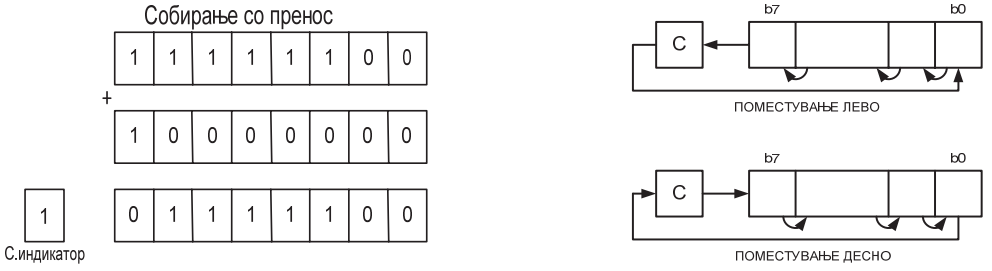
Секоја инструкција што може да ја изврши микропроцесорот има свој 8-битен операциски код. Со 8 бита може да се напрват 256 различни комбинации, односно 256 операциски кода. Откако инструкцијата ќе се пренесе од меморијата во микропроцесорот, таа се чува во посебен регистар наречен **инструкциски регистар.** Во него инструкцијата останува сè додека не биде оперативно извршена. Од инструкцискиот регистар инструкцијата се носи во декодерот. Декодерот има 8 влеза и 256 излеза. Секој од овие излези поттикнува одредена активност. Декодерот е поврзан со управувачката единица на микропроцесорот. Всушност, операциският код влегува во декодерот, а потоа предизвикува побуда на управувачкиот единица, која на својот излез дава управувачки сигнали. Кои управувачки сигнали ќе бидат

активни зависи од операциският код, но и од дигитскиот такт од кристалниот осцилатор (генераторот на такт).

Адресниот регистар **ја содржи адресата на мемориската локација каде што е сместен операндот врз кој треба да се изврши дадената операција.**

Аритметичко-логичката единица е дел од централниот процесор, кој ги извршува аритметичко-логичките инструкции врз бинарни броеви. Секоја аритметичко-логичка единица мора да содржи собирач, кој врши собирање на содржините на два регистра во согласност со бинарната аритметика. Користејќи го само основниот собирач, може да состават програми со кои ќе се врши одземање, множење и делење што, всушност, се основните аритметички операции. Во практика, покрај собирањето, аритметичко-логичката единица може да извршува хардверски и други функции, како што се одземање, булови логички операции и поместување низи. На аритметичко-логичката единица е приклучен таканаречениот **статус-регистар**. Овој регистар содржи индикатори на состојби, знаменца (flags). Знаменцата може да имаат различни улоги и најчесто ја покажуваат состојбата во аритметичко-логичката единица по извршувањето на некоја инструкција. Типични знаменца се :

- **C (carry)** – Индикатор на пренос. Овој бит хардверски се сетира ако постои пренос кај собирањето од осма на деветта позиција во бинарните броеви, односно позајмување кај одземањето од деветта на осма позиција. Исто така, знаменцето C се користи и при операциите на поместување низи. На сликата 7.3. се дадени два примера за употреба на знаменцето C.



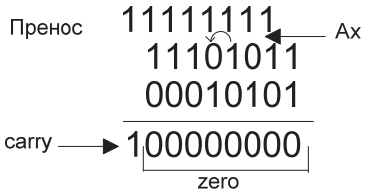
Слика 7.3. Примери за практична примена на знаменцето carry-индикатор за пренос

- **Z (zero)** – знаменцето нула е логичка единица ако резултатот од извршената операција е нула, а ако добиениот резултат е различен од нула, тогаш знаменцето Z е логичка нула.
- **S (sign)** – знаменце за знак. Ако резултатот на операција е негативен, ова знаменце е единица, а во спротивно е логичка нула .

- **P (parity)** – знаменце за парност. Ако резлутатот на операцијата содржи парен број единица знаменцето P е единица, а во спротивно е логичка нула.
- **Ax –** Ова е помошно знаменце за пренос и се активира кога имаме пренос од четврта на петта позиција при собирањето бинарни броеви или позајмување од петта на четврта позиција кај одземањето.

Пример 7.1: Кои знаменца се активираат при собирање на броевите $11101011_{(2)}$ и $00010101_{(2)}$?

Решение:



Се активираат знаменцата C, Z и A_x.

7.3. Машински циклуси за микропроцесор 8085

Секој инструкциски циклус започнува со **машински циклус за пренос на операциски код.** Веќе истакнавме дека кај микропроцесорот 8085 операцискиот код претставува 8-битна комбинација која е единствена за дадената инструкција. Овој машински циклус се разликува од останатите, по тоа што тој трае повеќе од три дигитски такта, бидејќи мора да се изврши декодирање за да микропроцесорот дознае што треба да прави натаму.

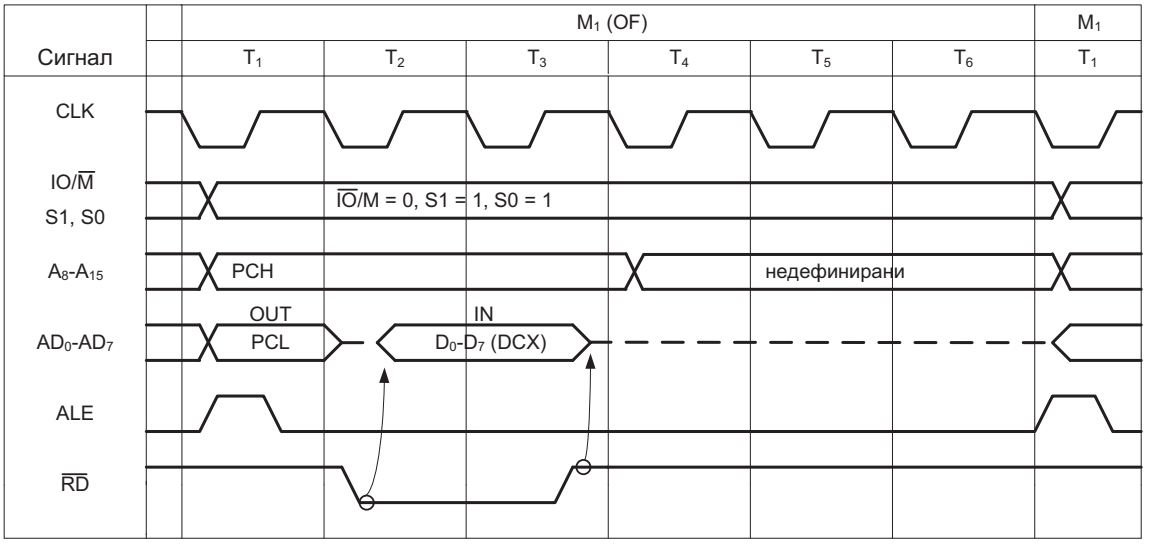
На слика 7.4. е даден временски дијаграм за машински циклус-пренос на операциски код. Да се потсетиме дека временскиот дијаграм претставува графички приказ на зависноста меѓу контролните сигнали што се активираат при извршувањето на инструкцијата. На почетокот од секој машински цуклус микропроцесорот 8085 ги активира статус-линиите со цел да го дефинира видот на машинскиот циклус. Кога се пренесува операциски код од меморијата до микропроцесорот, тогаш статусните линии ги имаат следниве вредности:

IO/$\overline{M}$=0 -процесорот ќе комуницира со меморијата (не со периферен уред)

S_1=1 -ќе се врши операција читање (не пишување)

S_0=1 –бајтот што се пренесува е операциски код (не е податок, операнд)

За време на првиот дигитски такт микропроцесорот 8085 испраќа 16-битна адреса со цел да ја идентификува мемориската локација или влезно-излезната порта до која се обраќа. Во случај на пренос на операциски код по адресната магистрала се испраќа содржината на програмскиот бројач. Позначајниот бајт (PCH- Program Counter High) се пренесува по линиите од A_8-A_{15} и тој бајт останува на нив сè до завршување на четвртиот дигитски такт. Понезначајниот бајт од програмскиот бројач (PCL- Program Counter Low) се пренесува по линиите ADo-AD_7 и тој останува на нив само еден дигитски такт. Имено, по првиот дигитски такт, линиите AD_0-AD_7 треба да бидат слободни за да преку нив се пренесе операцискиот код прочитан од некоја мемориска локација. Сигналот ALE е на високо ниво само за време на првиот дигитски такт кога адресно-податочната магистрала се користи за пренос на адресни битови. Сигналот ALE (Adress Latch Enable) се користи за оспособување на адресниот леч 8212. Откако микропроцесорот ќе ги испрати статусните сигнали и адресата до меморијата, контролниот сигнал $\overline{RD}$ се спушта на ниско ниво, со цел да ја подготви меморијата за читање.



Слика 7.4. Временски дијаграм на машински циклус за пренос на операциски код

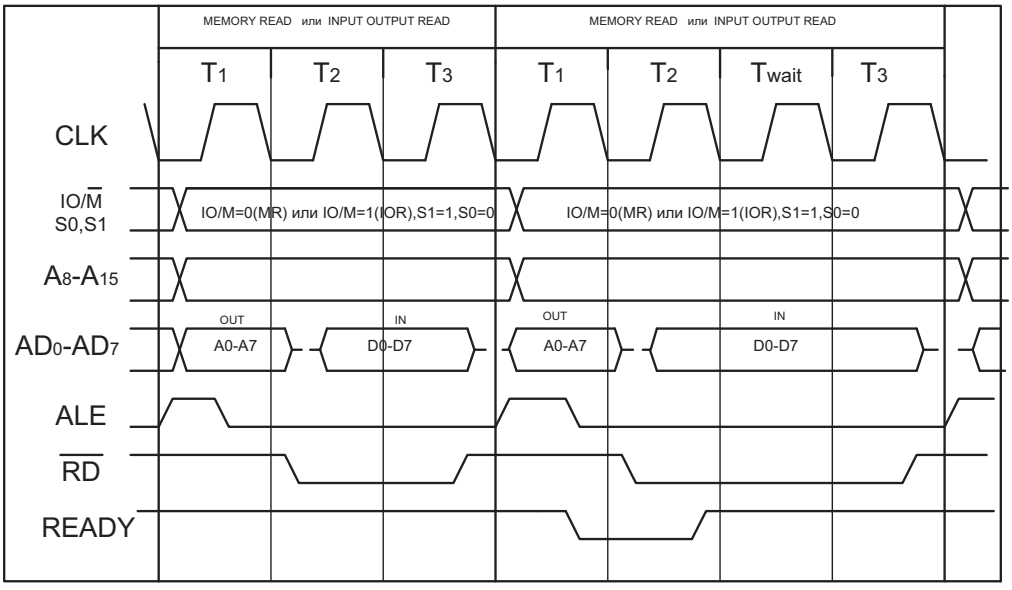
Во вториот дигитски такт меморијата ќе почне да се пребарува и по извесно доцнење ќе го стави бараниот бајт-операциски код на адресно-податочната магистрала. Колку ќе изнесува доцнењето, зависи од времето на пристап на меморијата.

За време на третиот дигитски такт Т3, микропроцесорот 8085 ќе го земе операцискиот код од AD магистралата и ќе го смести во инструкцискиот регистар. Микропроцесорот ќе го подигне сигналот $\overline{RD}$ на високо ниво со што ќе го онеспособи повиканиот мемориски уред.

За време на четвртиот дигитски такт Т4 се врши декодирање на операцискиот код и микропроцесорот одлучува дали со следниот дигитски такт

ќе влезе во T5 или, пак, во T1 на следната инструкција. За време на T5 и T6 адресните битови од Ao-A_7 можат да се променат и поради тоа е важно за време на овие дигитски такта сите мемориски и влезно-излезни уреди да бидат онеспособени со поставување на сигналот $\overline{RD}$ на високо ниво. Во спротивно, може да се случи со адресните битови Ao-A_7 да се селектира некој уред што би претставувало грешка.

На слика 7.5. се прикажани два последователни **машински циклус за читање од меморија**, при што првиот е без состојба на чекање (T-wait), а вториот е со една состојба на чекање. За машински циклус читање од меморија статусните сигнали ги имаат следниве вредности: IO/$\overline{M}$=0, S1=1, So=0.



Слика 7.5. Временски дијаграм за машински циклус за читање од меморија

Во споредба со машинскиот циклус-пренос на операциски код, само сигналот So е различен. Да напоменеме дека кај тој вид машински циклус пренесениот бајт е операциски код, а кај машинскиот циклус читање од меморија пренесениот бајт е податок, операнд врз кој треба да се изврши соодветна инструкција. Втора разлика меѓу машинските циклуси за пренос на операциски код и читање од меморија е што првиот трае минимум 4 такта, а вториот 3 такта. Ова е логично, бидејќи тука не е потребно декодирање. Трета разлика е тоа што кај преносот на операциски код адресата секогаш се зема од програмскиот бројач, а кај циклусот читање од меморија мемориската адреса може да потекнува од различни извори. Таа може да биде зададена во самата инструкција (MOV addr) или да се содржи во регистарскиот пар HL.

Операндите се сместуваат во некој од општите регистри или во акумулаторот ако не е поинаку нагласено. Кога машинскиот циклус содржи состојба на чекање (Twait), тогаш задолжително се активира сигналот READY..Во вториот такт микропроцесорот секогаш го проверува сигналот READY. Ако овој сигнал е на високо ниво, микропроцесорот ќе продолжи во

состојба T3 и ќе заврши со циклусот. Ако READY е на ниско ниво, тогаш микропроцесорот ќе вметне состојба на чекање Twait. Микропроцесорот може да вметне повеќе вакви состојби на чекање во зависност од времето на пристап на меморијата. Кај машинскиот циклус читање од влезно-излезни уреди (I/O read) временскиот дијаграм за зависноста меѓу контролните сигнали е ист како и временскиот дијаграм кај машинскиот циклус читање од меморија, со таа разлика што статусниот сигнал IO/$\overline{M}$=1. При запишување на податок во меморија наместо контролниот сигнал $\overline{RD}$ се користи сигналот $\overline{WR}$ и податоците што се запишуваат на магистралата AD останууваат на неа до крајот на T3 и не зависат од нивото на сигналот $\overline{WR}$ како што тоа беше случај со $\overline{RD}$ кај циклусот читање од меморија.

7.4. Начини на адресирање кај микропроцесор 8085

Во зависност од тоа како се пронаоѓаат податоците за обработка (операндите), кај микропроцесорот 8085 разликуваме **4 начини за адресирање**: непосредно, регистарско директно, директно и регистарско индиректно.

Кај **регистарското дирекното адресирање** податоците за обработка се наоѓаат во општите регистри во микропроцесорот. Инструкциите кои го користат овој начин на адресирање содржат еден бајт и тој претставува операциски код на инструкцијата. Инструкцијата не ги содржи операндите и доволен е само операцискиот код за да микропроцесорот дознае какво дејство да превземе. Инструкциите кои го користат регистарското адресирање се нарекуваат инструкции од прв формат. Тие зафаќаат мемориски простор од еден бајт, односно една мемориска локација.

Пример 7.2: На сликата 7.6. е претставена инструкцијата MOV H, L во машински јазик и таа е од прв формат. Оваа инструкција значи пренос на податокот од регистарот L во регистарот H. Двата регистра се наоѓаат во внатрешноста на микропроцесорот.

	7	6	5	4	3	2	1	0
1B	0	1	0	0	1	1	0	0

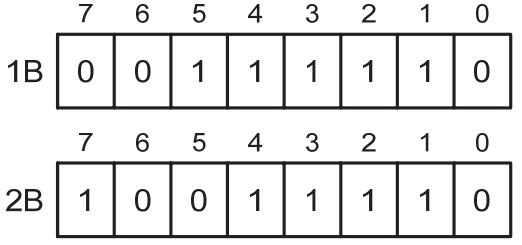
Слика 7.6. Прв инструкциски формат за инструкцијата MOV H,L

Седмиот и шестиот бит ја дефинираат инструкцијата. Во машински јазик само инструкцијата MOV започнува со битовите 01 и ниедна друга инструкција. Петтиот, четвртиот и третиот бит претставуваат код за регистарот H и

регистарот H преставува цел за пренесуваниот операнд. Вториот, првиот и нултиот бит претставуваат код за регистарот L и тој е извор на податокот за пренос. Кодовите на општите регистри беа прикажани во табелата 7.1.

Кај **непосредното адресирање** операндот се наоѓа во самата инструкција. Кај овој начин на адресирање непосредно по операцискиот код следува податокот. За овие инструкции велиме дека се од втор инструкциски формат. Вториот инструкциски формат содржи два бајта. Првиот бајт е секогаш операциски код, а вториот претставува осумбитен податок што треба да се обработи. Запишани во асемблер овие инструкции содржат осумбитен број, т.е. двоцифрен број во хексадецимален броен систем.

Пример 7.3: На сликата 7.7. е прикажан инструкцискиот формат на инструкцијата MVI A, 9EH. Со оваа инструкција податокот 9EH=10011110B се внесува во акумулаторот A.



Слика 7.7. Втор инструкциски формат за инструкцијата MVI A,9EH

Петтиот, четвртиот и третиот бит од првиот бајт претставуваат код на регистарот A. Останатите битови од првиот бајт се кодот на инструкцијата MVI. Вториот бајт го претставува бинарниот број што треба да се внесе во акумулаторот 10001110B=9EH

Кај **директното адресирање** не го знаеме податокот, но инструкцијата ни ја дава адресата на мемориската локација од каде што може да го прочитаме саканиот податок. Гледано во однос на претходните два начини на адресирање кај директното адресирање има најголем број на преноси од меморија пред операндот да пристигне во микропроцесорот. Инструкциите кои го користат овој начин на адресирање се од трет инструкциски формат. **Третиот инструкциски формат** содржи три бајти. Првиот бајт е операциски код, а вториот и третиот бајт заедно ја претставуваат 16-битна адреса на мемориска локација. Овој формат се користи кога податокот за обработка треба да прочита или да се запише во некоја мемориска локација од RAM меморијата. Асемблерските инструкциите од трет формат содржат 16- битен број или четирицифрен број во хексадецимален броен систем.

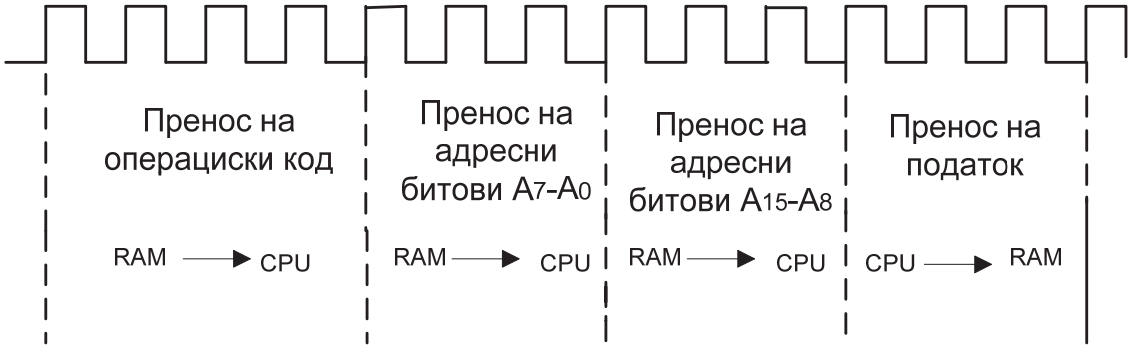
Пример 7.4: На сликата 7.8. е прикажан инструкцискиот формат на инструкцијата STA 8FFFH. Со оваа инструкција податокот се пренесува во акумулаторот од мемориска локација со адреса 8FFFFH. Првиот бајт 32H=00110010B претставува операциски код на инструкцијата STA. Вториот

бајт 11111111B=FFH се понезначајните битови од мемориската адреса A_7-A_0. Третиот бајт 8FH=10001111B се позначајните адресни битови A_{15}-A_8.

	7	6	5	4	3	2	1	0
1B	0	0	1	1	0	0	1	0
	7	6	5	4	3	2	1	0
2B	1	1	1	1	1	1	1	1
	7	6	5	4	3	2	1	0
3B	1	0	0	0	1	1	1	1

Слика 7.8. Трет инструкциски формат за инструкцијата STA 8FFFH

На слика 7.9. се прикажани четирите машински циклуси за извршување на инструкцијата STA 8FFFH.



Слика 7.9. Инструкциски циклус за инструкцијата STA 8FFFH

Во првиот машински циклус се пренесува операцискиот код (првиот бајт) од меморијата во микропроцесорот. Тој се сместува во инструкцискиот регистар и се врши декодирање. Во вториот машински цилкус се пренесуваат адресните битови од A_0 до A_7 (вториот бајт) од меморијата во микропроцесорот. Тие се сместуваат во адресниот регистар (address latch). Во третиот машински циклус истата постапка се повторува само за повисоките адресни битови од A_8 до A_{15} (третиот бајт). Конечно во четвртиот машински циклус содржината од акумулаторот се пренесува во назначената мемориска локација.

Кај **регистарското индиректно адресирање** регистарскиот пар HL ја содржи адресата на мемориската локација во која е сместен саканиот податок. Пример за регистарско индиректно адресирање е инструкцијата MOV A, M, со која се зема податокот од мемориската локација со адреса во регистарскиот пар HL и се копира во акумулаторот.

7.5. Инструкциско множество

Инструкциско множество на 8085 микропроцесорот е составено од 246 инструкции, но за почеток ќе се запознаеме само со 40 поважни инструкции. За секоја од наведените инструкции се дадени следниве информации: ефектот што го предизвикува таа инструкција, операцискиот код, големината на инструкцијата изразено во број на бајти и начинот на применето адресирање. Исто така, по објаснувањето на инструкцијата, даден е мал пример-извадок од некоја асемблерска програма. Во овие примери се дадени следниве информации:

- Адреса на мемориска локација зададена во хексадецимален систем (една цифра во хексадецимален систем е еквивалентно на 4 цифри во бинарен броен систем).
- Содржината на меморискатa локација која е претходно адресирана.
- Инструкција зададена во асемблер
- Коментар

7.5.1. Инструкции за пренос на податоци

Со еден пренос може да се пренесе информација од само еден бајт. Кај овие инструкции се среќаваат поимите извор (source) и цел (destination). Од изворот се зема податокот и се става во целта. Преносот на податоци може да се врши на релација регистар-регистар, регистар-мемориска локација или мемориска локација-регистар. Преку асемблерска инструкција не може да се изврши директен пренос од една мемориска локација во друга, односно со други зборови, податокот од првата локација мора да се префрли во некој регистар во микропроцесорот, а потоа од микропроцесорот во втората локација. При преносот податокот останува неизменет. Интересно е да се нагласи дека по извршениот трансфер содржината на изворот на податок е иста со содржината на целта. Всушност, можеме да кажеме дека наместо пренос вршиме копирање на податокот. Сепак, како термин не се користи зборот копирање, туку зборот пренос.

MVI r, податок

Функција	Податокот се сместува во регистарот r
Операциски код	00DDD110
Број на бајти	2 бајти
Адресен начин	Непосредно адресирање

Буквата D е прва буква од англискиот збор destination што во превод значи цел, дестинација. Кај оваа инструкција цел за податокот преставува некој општ регистар, чиј три-битен код DDD го одредуваме со помош на табелата 7.1.

Пример 7.5:

```
8200 3E MVI A ,9EH  ;Хексадецималниот број 9EH смести го во
                    ;акумулаторот
8201 9E             ;Податок
```

MOV r2, r1 или MOV destination, source

Функција	Содржината од регистарот r1 се копира во регистарот r2
Операциски код	01DDDSSS (D-destination , S-source)
Број на бајти	1 бајт
Адресен начин	Регистарско директно

Пример 7.6:

```
8150  26    MVI H, 00H    ;Регистарот H го полниме со вредност нула.
8151  00                  ;Податок
8152  6C  MOV L, H        ;Содржината од регистарот H се копира во
                          ; регистарот L.
```

LDA adr (LOAD –полнење)

Функција	Содржината од локацијата со адреса adr се пренесува во акумулаторот
Операциски код	00111010
Број на бајти	3 бајта
Адресен начин	Директно адресирање

Пример 7.7:

```
8200  3A  LDA 81C0H   ;Содржината од локацијата со адреса 81C0(16) се
                      ;копира во  акумулаторот.
8201  C0              ;Адреса A7-A0
8202  81              ;Адреса A15-A8
8203  6F   MOV L,A    ;Содржината од акумулаторот се копира во
                      ;регистарот L.
```

STA adr (**STORE**- складира, обратно од **LDA**)

Функција	Содржината од акумулаторот се копира во мемориска локација со адреса adr
Операциски код	00110010
Број на бајти	3 бајта
Адресен начин	Директно адресирање

Пример 7.8 :

```
8200  3E   MVI A,55H      ;Акумулаторот се полни со податок 55H.
8201  55                  ;Податок
8202  32   STA 8330H      ;Содржините од акумулаторот се копираат во
                          ; локацијата со адреса 8330.
8203  30                  ; A7-A0
8204  83                  ; A15-A8
```

IN adr

Функција	Бајтот од влезниот уред со интерфејс адреса „adr“ се пренесува во акумулаторот
Операциски код	11011011
Број на бајти	2 бајта
Адресен начин	непосредно

Пример 7.9

```
      8000   DB    IN 01H        ;Податокот од влезниот уред со интерфејс
                                 ;адреса01H копирај го во акумулаторот.
      8001   01                  ;Интерфејс адреса
      8002   67    MOV H,A       ;Содржината од акумулаторот копирај ја во
                                 ;регистарот H.
```

OUT adr (обратно од **IN**)

Функција	Содржината од акумулаторот се копира во излезниот уред со интерфејс адреса „adr“
Операциски код	11010011
Број на бајти	2 бајта
Адресен начин	Непосредно адресирање

Пример 7.10:

```
8200   3E      MVI A,55H      ;Податокот 55H се сместува во акумулаторот.
8201   55                     ;Податок
       8202   D3  OUT 02H     ;Содржината на акумулаторот се копира во
                              ;уред со интерфејс адреса 02.
8203   02                     ;Интерфејс адреса
```

7.5.2. Аритметички инструкции

Хардверски можат да се извршат две аритметички операции: собирање и одземање. При извршувањето на овие операции еден од операндите се наоѓа во акумулаторот, а другиот во некој друг регистар во микропроцесорот. Исто

така, по извршувањето на инструкцијата, добиениот резултат се запишува во акумулаторот со што се губи еден од операндите.

ADD r (аритметичко собирање)

Функција	Содржината на регистарот r се додава на содржината на акумулаторот
Операциски код	10000SSS
Број на бајти	1 бајт
Адресен начин	Регистарско директно
Знаменца	Zero, carry

Пример 7.11:

```
8200   7C   MOV A,H      ;Содржината на регистарот H се копира во
                          ;акумулаторот.
8201   85   ADD L        ;Содржината на регистарот L се додава на
                          ;содржината на акумулаторот.
8202   67   MOV H, A     ;Содржината на акумулаторот се копира во
                          ;регистарот H.
```

SUB r (**substract**-одземање)

Функција	Содржината на регистарот r се одзема од содржината на акумулаторот
Операциски код	10010SSS
Број на бајти	1 бајт
Адресен начин	Регистарско директно
Знаменца	Zero, carry

INR r (increment – зголеми) и **DCR r (decrrement** – намали)

Функција	Содржината на регистарот се зголемува/намалува за еден
Операциски код	000DDD0100/00DDD101
Број на бајти	1бајт
Адресен начин	Регистарско директно

Пример 7.12 :

```
8000  3A   LDA 81D8H     ;Содржината на локацијата со адреса
                          ;81D8H  се копира во акумулаторот.
8001   D8                 ; A7-A0
8002   81                 ; A15-A8
8003   3C   INR A         ;Содржината на акумулаторот се
                          ;зголемува за еден.
```

```
8004  32   STA 81D8H      ;Содржината на акумулаторот се носи со
                          ;адреса 81D8H.
8005  D8                  ; A7-A0
8006  81                  ; A15-A8
```

7.5.3. Логички операции

ANA r (AND ACCUMULATOR)

Функција	Врз содржината на акумулаторот и содржината на регистарот „r“ се применува логичката операција „И“ и резултатот се запишува во акумулаторот
Операциски код	10100SSS
Број на бајти	1 бајт
Адресен начин	Регистарско директно

ORA r (OR ACCUMULATOR)

Функција	Врз содржината на акумулаторот и содржината на регистарот „r” се применува логичката операција „ИЛИ“
Операциски код	10110SSS
Број на бајти	1 бајт
Адресен начин	Регистарко директен

CMA (Complement accumulator)

Функција	Сите битови во акумулаторот се инвертираат
Операциски код	00101111
Број на бајти	1 бајт
Адресен начин	Регистарско директно

Пример 7.13:

```
8000 3A   LDA 8250H     ;Содржината од локацијата со адреса 8250
                        ;се копира во акумулаторот.
8001  50                ; A7-A0
8002  82                ; A15-A8
8003  6F   MOV L,A      ;Содржината од акумулаторот се копира во
                        ;регистарот L.
8004  3A   LDA 8251H    ;Содржината на локацијата со адреса 8251
                        ;се копира во акумулаторот.
8005  51                ; A7-A0
8006  82                ; A15-A8
```

```
8007  A5    ANA L          ;Се пременува логичка операција „И“ врз
                           ;содржините на акумулаторот и регистарот L.
8008 32   STA 8252H        ;Содржината од акумулаторот се сместува во
                           ; локација  со адреса 8252.
800A 52                    ; A7-A0
800B 82                    ; A15-A8
```

Всушност, со оваа секвенца од инструкции вршиме логичка операција „НИ“ врз содржините на локациите со адреси 8250H и 8251H и добиениот резултат го запишуваме во локација со адреса 8252H.

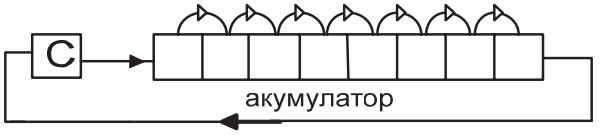
7.5.4. Инструкции за ротација

Според насоката, постојат два вида ротации: ротација влево и ротација во десно. Во зависност од тоа дали е вклучено знаменцето C или не, ротацијата може да биде преку знаменцето C или без него. Кај сите инструкции за ротација се ротира содржината на акумулаторот и ниеден друг регистар. Постојат две инструкции за ротирање на десно.

ROL(Rotate Left) - Ротацијата налево значи поместување на сите битови налево, а најлевиот и најзначаен бит се завртува и се носи на местото од нанејзначајниот, најдесниот бит. Оваа инструкција е прикажана на слика 7.10.

Слика 7.10. Ротација на акумулаторот налево

RRC (Rotate Right Carry)Ротацијата надесно преку знаменцето C (carry) значи поместување на битовите надесно за едно место, со тоа што најдесниот, најнезначаен бит влегува во знаменцето C, A вредноста на знаменцето C се носи на местото од најзначајниот, најлев бит. Ова е прикжано на слика 7.11.



Слика 7.11. Ротација на акумулаторот надесно

Пример 7.14 :

```
8140   DB   IN 01H         ;Пренеси го бајтот од влезниот уред во
                           ;акумулаторот.
8141   01                  ;Интерфејс адреа на влезната порта
```

```
8142   0F   RRC        ;Ротирај ја содржината на акумулаторот надесно.
8143   D3   OUT 02     ;Пренеси ја содржината на акумулаторот во
                        ;уредот  со интерфејс адрса 02H.
8143    02             ;Интерфејс адреса
```

7.5.5. Скок- инструкции (Jump-скок)

Разликуваме условни и безусловни скокови. Кога микропроцесорот ќе добие инструкција за безусловен скок, тој директно скока на мемориската локација чија адреса е наведена во инструкцијата. Кај условните скокови, за да се изврши скок, потребно е претходно да биде исполнет некој услов (на пример, некое од знаменцата да биде на високо ниво)

JMP adr

Функција	Наведената адреса се копира во програмскиот бројач
Операциски код	11000011
Број на бајти	3 бајта
Адресен начин	Директно адресирање

Пример 7.15 :

```
8100  3E   MVI A, 01H    ;Акумулаторот се полни со податокот
                         ;01_16=00000001_2.
8101  01                 ;Податок
8102  D3   OUT 02H       ;Содржината на акумулаторот се копира
                          ;во излезна порта.
8103 02                   ;Интерфејс адреса на влезната порта
8104 07    ROL           ;Содржината на акумулаторот се ротира
                         ;во лево.
 8105 C3   JMP 8102 H    ;Скок на локација со адреса 81202 (се
                          ;враќаме назад).
8106 02                  ; A7-A0
 8107 81                 ; A15-A8
```

Оваа програмска секвенца се користи за постигнување на ефектот „подвижно светло“ кај дисплеј составен од 8 полиња.

JZ adr (JUMP if ZERO-скокни ако знаменцето нула е единица-условен скок)

Функција	Наведената адреса во инструкцијата се копира во програмскиот бројач ако знаменцето нула е 1.
Операциски код	11001010
Број на бајти	3 бајти
Адресен начин	Директно адресирање

Пример 7.16 :

```
8200 7C   MOV A, H      ;Содржината на регистарот H копирај ја во
                        ;акумулаторот.
8201 95   SUB L         ;Содржината на регистарот L се одзема од
                        ;акумулаторот.
8202 CA   JZ 8266H      ;ако резултатот е нула, скокни на адресата
                        ;8266.
 8203 66                ; A7-A0
 8204 82                ; A15-A8
 8205 XX
```

Во оваа програмска секвенца се врши споредба меѓу содржините на регистрите и ако овие две содржини се еднакви, се врши скок.

JNZ adr (jump if not zero)

Функција	Скок се врши ако знаменцето на нула е на ниско ниво
Операциски код	11000010
Број на бајти	3 бајта
Адресен начин	Директно адресирање

Пример 7.17:

```
8300  26    MVI H,47H   ;Наполни го регистарот H со вредноста 47(16)
8301  47                ;податок (вредност).
8302  25    DCR H       ;Намали ја содржината на регистарот за еден.
8303  C2    JNZ 8302H   ;Ако добиениот резултат сè уште не е нула,
                          ;врати се на локацијата со адреса 8302H.
8304  02                ; A7-A0
8305  83                ; A15-A8
```

Оваа програмска секвенца, всушност, претставува бројач, чија вредност во секој чекор се намалува за еден.

7.5.6. Инструкции за работа со потпрограми

Потпрограмата значи независна програмска целина која извршува одредена функција. Овие потпрограми може да бидат употребни во повеќе главни програми.

RET – Оваа инструкција означува крај на повиканата потпрограма и враќање кон главната потпрограма.

HLT (halt) – Оваа инструкција треба секогаш да биде последна инструкција во програмата за да микропроцесорот правилно се подготви за извршување на програмата. Без оваа инструкција, микропроцесорот ќе се обидува да ги

протолкува содржините и на следните мемориски локации и тогаш може да дојде до хаос.

CALL adr

Функција	Со оваа инструкција главната програма ја повикува потпрограмата да биде извршена во микропроцесорот. Наведената адреса претставува адреса за првата локација од која почнува потпрограмата.
Операциски код	11001101
Број на бајти	3 бајта
Адресен начин	Директно адресирање

Пример 7.18:

Главна програма

```
8000 26     MVI H,33H       ;Наполни го регистарот H со податокот 33H
8001 33                     ;податок.
8002 2F     MVI L,8FH       ;Наполни го регистарот L со податокот  8FH
8003 8E                      ;податок.
8004 CD     CALL 8081H      ;Повикување на потпрограмата со почетна
                             ;адреса 8010H.
8005 10                     ; A7-A0
8006 80                     ; A15-A8
8007 32     STA 8300H       ;Внеси го добиениот резултат во локацијата
                             ;8300H.
8008 00                     ; A7-A0
8009 83                     ; A15-A8
800A 76 HLT                 ;Крај на програмата
800B XX
```

Потпрограма

```
8010 7C     MOV A,H         ;Содржината на регистарот се копира во
                             ; акумулаторот.
8011 85      ADD L          ;На содржината на акумулаторот додај ја вредноста
                            ; регистарот L.
8012 C9     RET             ;Врати се на главната програма.
```

7.5.7. Инструкции за работа на стек меморија

Овие инструкции се користат кога сакаме да преминеме од главна програма во некоја потпрограма. Акумулаторот и статус-регистарот се даваат на располагање на потпрограмата. Пред да почне извршувањето на

потпрограмата, вредноста на овие два регистра треба да се зачува, бидејќи тие ќе бидат потребни кога микропроцесорот ќе се врати во главната програма. Вредноста на акумулаторот и статус-регистарот се сместуваат на врвот од стекот. Кога микропроцесорот ќе заврши со потпрограмата, тие ќе бидат пак вратени во микропроцесорот.

PUSH PSW – Вредноста на акумулаторот и статус-регистарот се сместува на врвот од стекот

POP PSW - Зачуваните содржини на акумулаторот и на статус-регистарот се симнуваат од врвот на стекот во регистрите во микропроцесорот.

Пример 7.19:

823C	F5	PUSH PSW	;Зачувај ја содржината на ;акумулаторот и статус регистарот ;на врвот од стекот.
823D	CD	CALL 82E0H	;Повикај ја потпрограмата со почетна ;адреса 82E0H.
823E	E0		; A_7-A_0
823F	82		;A_{15}-A8
8240	F1	POP PSW	;Врати ги содржините на ;акумулаторот и статус регистарот.

7.6. Пишување на програми за микропроцесор 8085

По секоја инструкција, **еден од општите регистри ќе ја смени својата содржина**. Кај поголем број инструкции тоа е акумулаторот (LDA ,STA, ADD, SUB, ANA, ORA, CMA, ROL, RRC) , но може да биде и некој друг регистар (INC r, DCR r , MOV r1,r2 MVI r,data). Во примерите што следуваат по извршувањето на секоја инструкција се утврдува содржината на употребените општи регистри.

Пример 7.20: Дадени се содржините на регистрите A=76H и B=D4H. Пресметај ја нивната содржина по извршувањето на следниве инструкции:

```
AND  B
CMA
DCR  B
```

Решение:
Врз содржините на регистрите A= 01110110B , B=11010100H се применива логичка операција И.

AND B

```
  01110110 ──A
  11010100 ──B
  01010100 ─ ново A
```

Моментална содржина на регистрите: A=01010100B, B=11010100B (регистарот B останал ист)

Се пресметува прв комплемент на содржината на регистарот A=01010100B.

CMA $\overline{01010100}=10101011=A$

Моментална содржина на регистрите : A= 10101011B ,B=11010100B

Вредноста на регистарот B=11010100B се намалува за еден

DCR B

```
  11010100 ── B
-        1
  11010011 ─ Ново B
```

Крајно решение: A=100101011B, B=11010011B

Пример 7.21: Пресметај ја содржината на регистрите A и E после извршувањето на следните инструкции!

```
MVI A,A3H
MVI E,9EH
SUB E
```

Решение:

На почетокот содржината на регистрите е непозната. Преку инструкцијата MVI на регистрите им се задаваат одредени вредности.

```
MVI A, A3H
MVI E,9EH
```

A=A3H=10100011B, E=9EH=10011110B

Од содржината на регистарот A се одзема содржината на регистарот E.

SUB E

```
  10100011 ── A
- 10011110 ── E
  00000101 ── ново A
```

Крајно решение: A=00000101B, E=10011110B (останало исто)

Пример 7.22: Пресметај ја содржината на програмскиот бројач по извршувањето на следнава програмска секвенца. На почетокот од програмската секвенца вредноста на програмскиот бројач изнесува 8000H.

```
MVI  A,65H
MVI  B,D8H
ORA  B
```

STA 1234H

Решение:
Во овој пример е важно да се познаваат инструкциските формати на применетите инструкции. Ако тековната инструкција е од прв формат, тогаш на содржината на програмскиот бројач се додава еден за да се добие адресата на следната инструкција. Ако тековната инструкција е од втор формат, тогаш се додава два, а ако е од трет формат, се додава три.

содржина на програмски бројач	инструкција	коментар
8000H	MVI A,65H	Оваа инструкција е од втор формат. На содржината на програмскиот бројач додаваме два. 8000+2=8002H
8002H	MVI B,D8H	8002+2=8004H
8004H	ORA B	Оваа инструкција е од прв формат. Додаваме еден. 8004+1=8005H
8005H	STA 1234H	Оваа инструкција е од трет формат. Додаваме три. 8005+3=8008H
8008H		

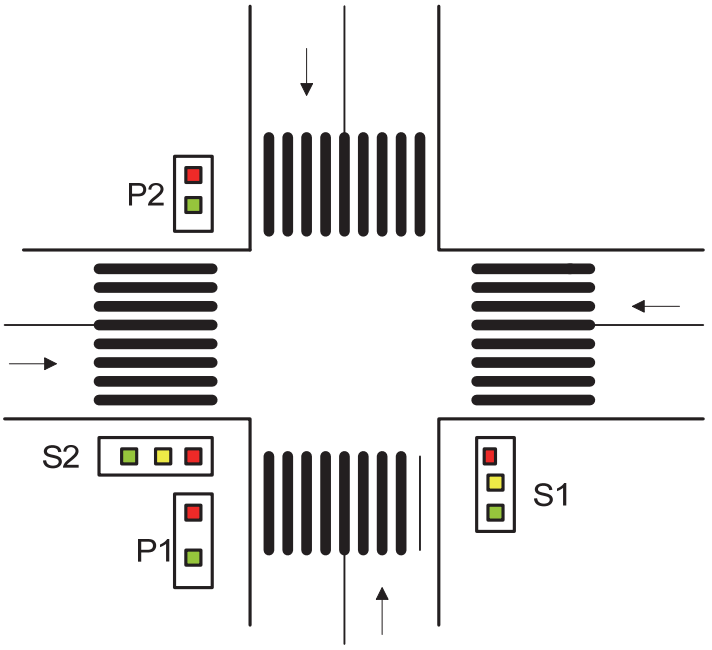
Значи по извршувањето на дадената програмска секвенца, вредноста на програмскиот бројач изнесува 8008H.

7.6.1. Софтвер за регулација на сообраќај на едноставна раскрсница

Сите сме забележале со каква динамика се палат светлата на семафорите. По црвеното светло заедно светат црвено и жолто светло, а потоа се пали заленото. По зеленото светло свети само жолтото за потоа да светне црвеното. Што се однесува до семафорите за пешаци зеленото светло свети само кога на семафорот за автомобили свети црвено светло. За да биде покриена целата раскрсница, потребен е уште еден ваков систем кој ќе се постави дијагонално на системот прикажан на сликата 7.12.

Програмата што се користи е составена од два дела. Првиот дел е со големина од 6B и ги содржи **податоците што треба да се пренесат на излезната порта.** Пиновите од излезната порта се поврзани со светлата на семафорите и кога на пинот ќе дојде логичка единица, светлото свети.

LIGHT_LOC претставува симболично име, лабела за адресата на првата локација во која е сместен првиот податок.



Слика 7.12. Приказ на светлата на семафорот и нивното бележење

Забелешка: Доколку вашата симулација не поддржува лабели, тогаш зададете ја адресата како 16-битен број или четирицифрен хексадецимален број.

семафори и пешаци	S_1	S_2	P1	P2	
бои на светла	CZZ	CZZ	C	C	
Light_loc:	001	100	1	0	(1min)
	010	100	1	1	(3sec)
	100	110	1	1	(3sec)
	100	001	0	1	(1min)
	100	010	1	1	(3sec)
	110	100	1	1	(3sec)

Вториот дел од програмата **ги содржи инструкциите** што треба да ги пренесат податоците од првиот дел до излезните пинови. На почетокот, со двете инструкции MVI се внесува адресата на првиот бајт од податочниот дел во регистарскиот пар HL. Со инструкцијата MOV A,M адресата од HL парот се носи до меморијата, се пронаоѓа саканиот бајт и се сместува во акумулаторот. Со инструкцијата CALL DOCNI 1 мин. се задржува содржината на акумулаторот една минута колку што треба да свети црвеното и зеленото светло. Ако не се повика оваа потпрограма, содржината на акумулаторот многу брзо ќе се промени (за време од неколку ns) што не е доволно време да се запалат светлата на семафорот. Во следниот чекор (INX H) содржината на

регистраскиот пар се зголемува за еден, со што ја добиваме адресата на вториот бајт 01010011 од податочниот дел. Постапката за пренос на бајт од меморијата во акумулаторот, а понатака до излезната порта се повторува шестпати (за секој бајт од податочниот дел посебно). На крајот, со инструкцијата JUMP VRTI сево ова повторно се повторува, односно на семафорот S_1 ќе свети зеленото светло, а на S_2 црвеното светло.

```
vrti:   MVI   H,LIGHT
        MVI   L,LOC
        MOV   A,M
        OUT   PORT A
        JNX   H
        MOV   A,M
        OUT   PORT A
        CALL  DOCNI_3sec
        .
        -------
        INX   H
        MOV   A,M
        OUT   PORT A
        CALL  DOCNI_3sec
        JUMP VRTI
```

7.6.2 Компарација на кодови

На излезната порта од компјутерот се поврзани 8 лед диоди. Диодите се зелени и црвени, наизменично наредени една до друга.

```
RGRGRGRG        G-green
                R-red
```

Диодите светнуваат ако на нивните пинови се донесат логички единици. На влезната порта од сметачот паралелно се поврзани 8 прекинувачи. Ако прекинувачот е затворен, тогаш пинот се поврзува со маса и имаме состојба на логичка нула, а кога тој е отворен пинот е поврзан на напојување и имаме состојба на логичка единица. 8-битната комбинација од влезната порта се споредува со содржината на мемориската локација со адреса 8250. Ако двата бајта се еднакви, тогаш се палат зелените диоди, а ако не, црвените. Компарацијата се врши преку одземање на двата броја. Ако тие се исти, тогаш резултатот е нула и се активира знаменцето ZERO.

```
        LDA 8250        ;Од локацијата со адреса 8250 податокот се пренесува
```

```
                    ;во акумулаторот.
MOV H,A             ;Податокот од акумулаторот се пренесува во
                    ;регистарот H со цел да се заштити.Со следната
                    ;инструкција содржината на акумулаторот може да се
                    ;смени и податокот да биде изгубен.
8204   IN 01H       ;Податокот се пренесува од акумулаторот до излезната
                    ;порта со адреса 01H.
       SUB H        ;Ако биде успешна компарацијата се активира
       JZ 8211      ;знаменцето ZERO и се врши скок на локација со
       MVI A,AA     ;адреса 821. Светнуваат зелените диоди и програмата
       OUT 02       ;завршува.
       JMP 8204     ;Ако компарацијата е неуспешна, знаменцето останува
8211   MVI A,55     ;на ниско ниво, не се врши скок, туку се извршува
       OUT 02       ;следната инструкција MVI A,AAH. Бројот AAH=
                    ;10101010B се пренесува во акумулаторот, а оттаму на
                    ;излезната порта. Единиците во бројот 10101010 имаат
                    ;исти позиции како и црвените диоди на излезната
                    ;порта. Кога ќе се запалат црвените лед диоди се врши
                    ;безусловен скок  8204 и повторно се донесува нова
       RST7         ;комбинација преку влезните прекинувачи.
```

7.7. Интегрирани компоненти за микрокомпјутерски систем 8085

По појавата на микропроцесорот 8085, „Интел“ дизајнираше нови интегрирани компоненти, кои беа целосно компатибилни со микропроцесорот 8085. Овие интегрирани компоненти посредуваат во комуникацијата меѓу микропроцесорот и периферните уреди. Тие се познати под името контролери, бидејќи го контролираат преносот на податоци од периферниот уред до микропроцесорот или обратно. Крајно непрактично би било микропроцесорот и периферниот уред да бидат директно поврзани. Интерфејс-компонентите можат да бидат непрограмибилни и програмибилни. **Програмибилните** компоненти содржат неколку бајти меморија (еден или повеќе), чија содржина ја одредува самиот програмер. На пример, постои бит што одредува дали осумте пина на портата А од интерфејс-компонентата ќе бидат влезни или излезни (единица за влез или нула за излез). При поврзувањето на влезните уреди со микропроцесорот мора да се обезбеди податочно баферирање. Баферирањето значи поврзување на периферниот уред со податочната

магистрала на микропроцесорот само за време на инструкцијата IN. Баферите можат да бидат дел од програмибилните интерфејс-компоненти или да бидат посебни интегрирани кола. При поврзувањето на излезните уреди со микропроцесорот се користат лечови кои треба да го продолжат времетраењето на податоците многу подолго од времетраењето на инструкцијата OUT. Оваа постапка е неопходна ако се знае дека инструкцијата трае помалку од 100ms. Најчесто лечот е во внатрешноста на самите интерфејс-контролери.

7.7.1. Интерфејс-компонента 8212

8212 е едноставна, повеќенаменска, непрограмибилна компонента и се користи во секој микропроцесорски систем 8085. Ова интегрирано коло може да се употреби и за локални потреби во други системи. Таа е еднонасочна, односно може да пренесува податоци само од лево на десно.

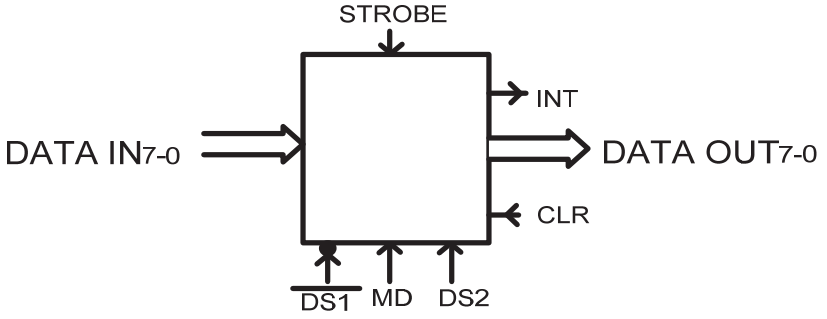
Пинови на компонента 8212

На сликата 7.13. е прикажан пин-дијаграм на компонентата 8212.

- Осумте пина DI (Data In) служат за влез на податоците од микропроцесорот или од периферниот уред, а пиновите DO (Data Out) служат само за излез на податоците од компонентата 8212.

- STB (Strobe) е влезен пин и преку него микропроцесорот или периферниот уред најавуваат пренос на податоци за компонентата 8212. Доколку компонентата работи во стробиран режим, тогаш без STB= 1 влезните податочни линии не ќе може да примат податоци.

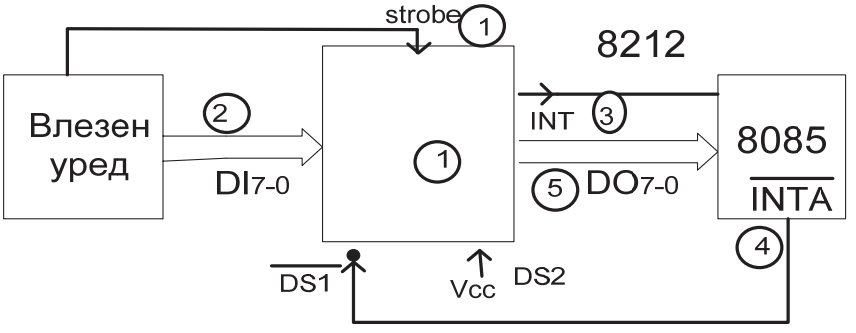
- CLR(Clear) служи за ресетирање на осумте тактирачки флип-флопа во внатрешноста на компонентата 8212.
- INT е излезен пин и служи за генерирање прекин на микропроцесорот 8085.



Слика 7.13. Пин дијаграм на компоненетата 8212

- MD е влезен пин и служи за селекција на режимот на работа. Кога MD е на високо ниво, компонентата 8212 работи како типичен **бафер**. Податоците минуваат директно низ компонентата, транзитно, без да се задржуваат. Само што ќе влезат податоците во 8212 преку портата DATA IN ќе можат да излезат на DATA OUT. Кога MD е на ниско ниво, тогаш компонентата 8212 работи како **леч.** Податоците привремено се чуваат во D флип-флоповите. Со помош на сигналот STB, податоците се вчитуваат преку пиновите DI_7 – DI_0, а се читаат по активирањето на пиновите $\overline{DS1}$ и DS2.

- За оспособување на излезната податочна порта се користи еден од пиновите $\overline{DS1}$ или DS2. Пинот $\overline{DS1}$ е активен на логичка нула, а пинот DS2 на логичка единица. Доколку микропроцесорот ја испрати потврдата за прекин до пинот $\overline{DS1}$, тогаш пинот DS2 треба да се поврзе за напојувањето V_{cc}, а ако микропроцесорот ја испраќа потврдата до пинот DS2, тогаш $\overline{DS1}$ треба да се поврзе на маса.



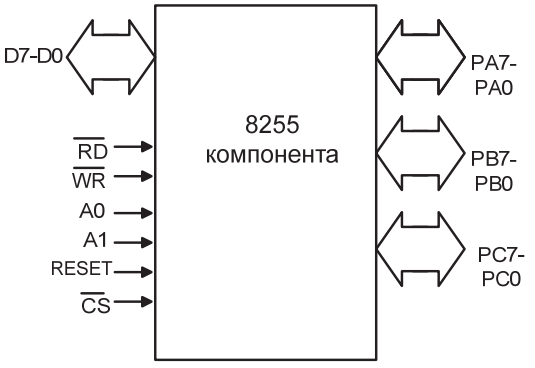
Слика 7.14. Редослед на испраќање на податоци на компонентата 8212

Кога 8212 работи како влезна порта (прима податоци од некој влезен периферен уред и ги препраќа до микропроцесорот), тогаш редоследот на испраќање сигнали е следен. Влезниот уред го активира пинот за најава (strobe). По добиената најава, 8212 ја отвора влезната податочна порта DATA IN и пододоците влегуваат во 8212 . Во 8212 податоците остануваат сè додека 8212 не побара прекин од микропроцесорот преку активирање на пинот INT. Ако микропроцесорот дозволи прекин, тогаш се оспособува излезната податочна порта (DATA OUT) и конечно податоците пристигнуваат до микропроцесорот. На слика 7.14. е прикажано поврзувањето на контролерот 8212 кога тој работи како влезна порта. Со кругчиња е обележан **редоследот на испраќање сигнали**.

7.7.2. Програмибилна компонента 8255

8255 е програмибилна интерфејс-компонента. Таа нема микропроцесор и RAM меморија за да извршува софтверски програми, но има команден регистар чија содржина ја одредува самиот програмер. Оваа компонента наоѓа примена во многу микропроцесорски системи, како што се системите со Пентиум микропроцесори. Контролерот 8255 се користи за поврзување на персоналните компјутери со тастатурата и со принтер.

На слика 7.15 е прикажан **пин-дијаграмот** на контролерот 8255.



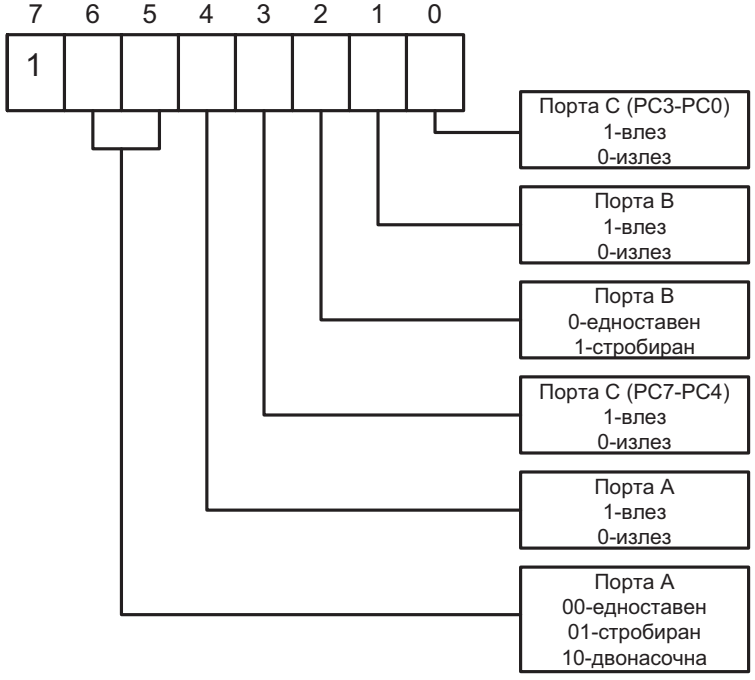
Слика 7.15. Пин дијаграм на компоненета 8255

PA, PB и PC се 8-пински порти и служат за поврзување на 8255 со периферните уреди. За поврзување на 8255 со микропроцесорот се користат податочните пинови (D_7-D_0). За избор на операција (читање или пишување) се користат контролните пинови $\overline{RD}$ и $\overline{WR}$.

За да микропроцесорот пристапи до компонентата 8255, пинот $\overline{CS}$ мора да се активира. Да се потсетиме дека овој сигнал се генерира преку постапката на адресно декодирање. Откако ќе се пристапи до компонентата 8255 се врши избор на еден внатрешен регистар. Под внатрешни регистри се сметаат трите порти на контролерот и командниот регистар. Портите служат за читање и за пишување од периферните уреди, а преку командниот регистар се врши програмирање на контролерот. Изборот на внатрешен регистар се врши преку адресните пинови A_1 и A_0. Ова значи дека микроконтролерот ќе користи четири адреси. Ако компонентата 8255 се поврзе со микропроцесор 8085, тогаш битовите од A_{15} до A_2 ќе бидат исти за сите четири адреси и тие, всушност, ќе учествуваат во генерирањето на сигналот $\overline{CS}$. Битовите A_1 и A_0 ќе бидат различни во секоја од четирите адреси за да се направи одвојување на внатрешните регистри. На пример, каде ќе се запишат битовите што ги испратил микропроцесорот по податочната магистрала зависи од состојбата на битовите A_1 и A_0.

A_1	A_0	
0	0	порта А
0	1	порта В
1	0	порта С
1	1	команден регистар

Компонентата 8255 се програмира преку командниот регистар. **Командниот регистар** е 8-битен регистар и одредувајќи ја неговата содржина, програмерот одлучува во кој режим ќе работат портите на контролерот. Да се потсетиме дека постојат четири режими на пренос: едноставен, стробиран, режим на поздрав и режим на двојно поздравување.



Слика 7.16. Функциски опис на битовите на командниот регистар од компонентата 8255

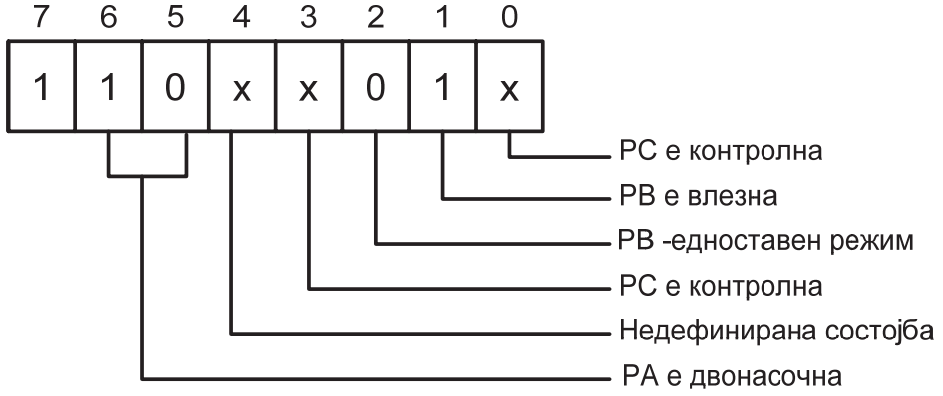
Портата А може да биде влезна, излезна или двонасочна порта и да работи во едноставен или стробиран режим. Портата Б може да биде влезна или излезна (но не и двонасочна) и може да работи во едноставен или во стробиран режим.

Портата С може да биде корисничка или контролна порта. Пората С може да биде корисничка порта само кога портите А и В работат во едноставен режим. Тогаш може да избираме какви ќе бидат пиновита на PC, влезни или излезни. Ако било која порта, А или В, не работи во едноставен режим, тогаш портата С ќе биде контролна и нејзините пинови ќе се користат за пренос на сигналите за најава и за потврда. На слика 7.16. е прикажана содржината на командниот регистар и значењето на секој бит од него. Ако портата С е контролна, тогаш е сеедно дали третиот и нултиот бит од командниот регистар

се 0 или 1. Тогаш нив ги бележиме со X, недефинирана состојба. Исто така, и четвртиот бит ќе биде X ако PA е двонасочна.

Пример 7.23: Одреди ја содржината на командниот регистар на компонентата 8255 ако портата A е двонасочна, а портата B е влезна и работи во едноставен режим.

Решение:



Слика 7.17. Содржина на команден регистар кога портата A е двонасочна и портата B е влезна и стробирана

Портата C е контролна, бидејќи портата A не е во едноставен режим. Кога портата C се користи како **контролна порта**, тогаш наместо корисничка информација, нејзините пинови ќе пренесуваат контролни сигнали. Тоа се сигнали со кои се најавува преносот на кориснички податоци по портите A и B и податочната магистрала. Освен за најава на преносот, постојат контролни сигнали со кои се потврдува добивањето на испратените податоци. Во табела 7.2. се дадени ознаките на сите контролни сигнали од портата C за сите режими на пренос на портите A и B.

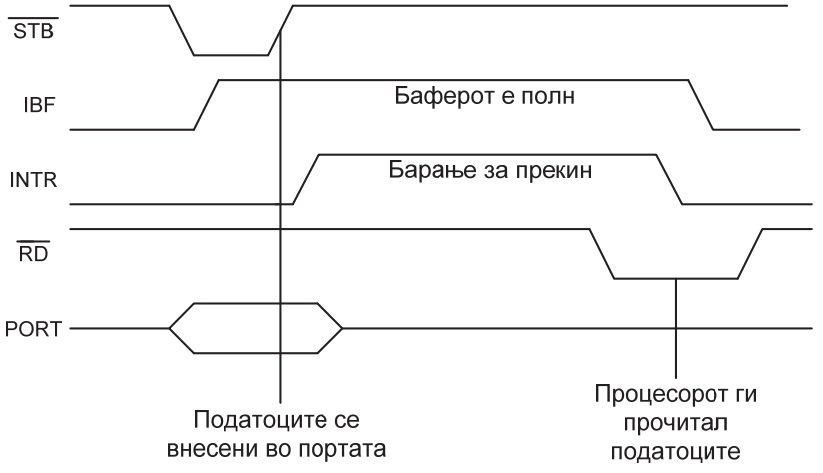
Пин	Стробиран режим		Двонасочна
	Влез	Излез	
0	$INTR_B$	$INTR_B$	$INTR_B$
1	IBF_B	$\overline{OBF}_B$	IBF_B
2	$\overline{STB}_B$	$\overline{ACK}_B$	$\overline{STB}_B$
3	$INTR_A$	$INTR_A$	$INTR_A$
4	IBF_A	/	STB_A
5	$\overline{STB}_A$	/	IBF_A
6	/	$\overline{ACK}_A$	ACK_A
7	/	$\overline{OBF}_A$	$\overline{OBF}_A$

Табела 7.2. Бележење на пиновите од портата C кога таа работи како контролна порта

Сигналите со индекс А се однесуваат на портата А, а оние со индекс В на портата В.

- INTR (Interrupt) е сигнал со кој контролерот 8255 бара прекин во тековната работа на микропроцесорот. Овој прекин е потребен за да контролерот му ги испрати податоците на микропроцесорот, добиени од периферен уред. Доколку микропроцесорот го одобри барањето за прекин, ќе се активира пинот $\overline{RD}$ на компонентата 8255.
- $\overline{STB}$ (strobe) е сигнал што го најавува преносот на податоци и го испраќа влезниот уред до компонентата 8255. Сигналот $\overline{STB}$ ги оспособува пиновите на портата А или портата В за прием на испратените податоци.
- IBF (Input Buffer Full) е сигнал што се активира кога податоците ќе пристигнат во компонентата 8255. Сигналот IBF го испраќа компонентата 8255 до влезниот уред.
- $\overline{ACK}$ (Acknowledge) е сигнал што се користи кога портата А или портата В се излезни порти. Пинот $\overline{ACK}$ го активира уредот откако ќе ги прими податоците од компонентата 8255.
- OBF (Output Buffer Full) е сигнал што го испраќа компонентата 8255 до излезниот уред, со цел да го најави преносот на податоци. Овие податоци компонентата 8255 ги примила од микропроцесорот, но тие треба да се испратат до излезниот уред.

На слика 7.18. е прикажан временски дијаграм на сигналите кога една од двете порти А или В работи како **влезна порта**.



Слика 7.18. Временски дијаграм кога 8255 работи како влезна порта

Редоследот на активирање на сигналите е следен:

1. Влезниот уред го активира пинот $\overline{STB}$ и ги оспособува пиновите на портата А или В за прием на податоци.
2. Податоците влегуваат во компонентата 8255 преку пиновите на портата А или В.

3. Кога 8255 ќе ги прими податоците ќе го активира пинот IBF и му потврдува на влезниот уред дека податоците пристигнале до 8255.
4. Но, податоците не треба да останат во 8255. Тие треба да се препратат до микропроцесорот. Поради тоа 8255 испраќа барање за прекин до микропроцесорот преку пинот INTR.
5. Ако микропроцесорот го одобри барањето за прекин, тој ќе го активира контролниот пин $\overline{RD}$.
6. Податоците преку податочната магистрала ќе пристигнат до микропроцесорот.

Заклучоци:

8085 микропроцесорот е 8 битен, располага со генератор на такт кој произведува дигитски такт со фреквенција 8 MHz, посебен контролер за работа со прекини и контрола на тајмингот. Има пристап до меморија со капацитет 64KB, односно има 16 адресни пина наменети за директно адресирање на меморискиот простор (2^{16} =64K).

ALE е контролен сигнал за адресно податочната магистралата. Ако ALE=1 тогаш адресно-податочната магистрала пренесува адресни битови. Ако ALE=0 тогаш магистралата пренесува податочни пинови.

Кај микропроцесорот 8085 постојат пет видови прекини: TRAP, RST7.5, RST6.5, RST5.5 и INTR. TRAP е прекин од најголем приоритет. Трите рестстартирачки прекини имаат фиксна рестартирачка адреса. INTR прекинот нама фиксна рестартирачка адреса туку која прекинувачка подпрограма ќе се изврши зависи од видот на периферниот уред кој генерирал прекин.

Преку трите статусни линии S0,S1, IO/ $\overline{M}$ микропроцесорот дава информација за тоа каков машински циклус ќе се реализира. Разликуваме шест видови машински циклуси: пишување во меморија, читање од меморија, пишување во излезен уред, читање од излезен уред, пренос на операциски код и обработка на прекини.

Акумулаторот е работен регистар на микропроцесорот и се наоѓа на влез од аритметичко логичката единица. Покрај акумулаторот во регистри со општа намена влегуват уште шест други регистри, кои се бележат со буквите B, C, D, E, H, L.

Програмскиот бројач е регистар во 8085 микропроцесорот, кој ја памти адресата на следниот бајт што треба да се пренесе од меморијата и да се

обработи. Микропроцесорот додава еден на содржината на програмскиот бројач по секое донесување нов бајт од меморијата до микропроцесорот.

Микропроцесорот содржи посебен регистар стек-покажувач (stack pointer), кој е 16-битен регистар и ја содржи адресата на мемориската локација на врвот од стекот.

Статус-регистарот содржи индикатори на состојби, знаменца (flags). Знаменцата може да имаат различни улоги и најчесто ја покажуваат состојбата во аритметичко-логичката единица по извршувањето на некоја инструкција. Типични знаменца се : знаменцето за пренос, знаменце за парност, знаменце за нула, знаменце за знак и знаменце за помошен пренос.

Кај регистарското директното адресирање податоците за обработка се наоѓаат во општите регистри во микропроцесорот. Инструкциите кои го користат регистарското адресирање се нарекуваат инструкции од прв формат. Тие зафаќаат мемориски простор од еден бајт, односно една мемориска локација.

Кај непосредното адресирање операндот се наоѓа во самата инструкција. За овие инструкции велиме дека се од втор инструкциски формат. Вториот инструкциски формат содржи два бајта. Првиот бајт е секогаш операциски код, а вториот претставува осумбитен податок што треба да се обработи.

Кај директното адресирање инструкцијата ја содржи адресата на мемориската локација од каде што може да го прочитаме саканиот податок. Инструкциите кои го користат овој начин на адресирање се од трет инструкциски формат. Третиот инструкциски формат содржи три бајти. Првиот бајт е операциски код, а вториот и третиот бајт заедно ја претставуваат 16-битна адреса на мемориска локација.

Преносот на податоци може да се врши на релација регистар-регистар (MOV), акумулатор-мемориска локација (STA), мемориска локација-акумулатор (LDA), акумулатор-излезен уред (OUT) и влезен уред-акумулатор (IN). Преку асемблерска инструкција не може да се изврши директен пренос од една мемориска локација во друга.

Кај аритметичките инструкции за собирање и одземање едниот операнд се наоѓа во акумулаторот, а вториот операнд се наоѓа во некој од општите регистри на микропроцесорот 8085. Добиениор резултат се запишува во акумулаторот со тоа што се губи неговата стара содржина. Истото важи и за логичките инструкции AND и ORA.

Според насоката, постојат два вида ротации: ротација влево и ротација во десно. Во зависност од тоа дали е вклучено знаменцето C или не, ротацијата може да биде преку знаменцето C или без него.

Инструкциите за скок и подпрограми секогаш содржат адреса на локација на која треба да се скокне или започне потпрограмата.

8212 е едноставна, повеќенаменска, непрограмибилна компонента и се користи во секој микропроцесорски систем 8085. Кога интерфејс компонентата работи како влезна порта тогаш прима барања за пренос од приферниот уред преку пинот за најава (strobe). 8212 компонентата може да предизвика прекин во работата на микропроцесорот преку активирање на пинот за прекин INT. Доколку 8085 микропроцесорот го прифати барањето за прекин тогаш ги активира пиновите за селекција на 8212 компонентата, $\overline{DS1}$ и DS2.

Компонентата 8255 се програмира преку командниот регистар. Командниот регистар е 8-битен регистар и одредувајќи ја неговата содржина, програмерот одлучува во кој режим ќе работат портите на контролерот. Портата А може да биде влезна, излезна или двонасочна порта и да работи во едноставен или стробиран режим. Портата Б може да биде влезна или излезна (но не и двонасочна) и може да работи во едноставен или во стробиран режим. Портата С може да биде корисничка или контролна порта.

Прашања и задачи

1. Процесорот 8085 е 8-битен. Објасни!

---

2. Колку изнесува широчината на адресната и на податочната магистрала на микропроцесор 8085?

---

3. Објасни како сигналот ALE (Address Latch Enable) го контролира мултиплексот на адресно-податочната магистрала?

---

4. Каква операција ќе изврши микропроцесорот 8085 ако контролните сигнали се на следниве логички нивоа $\overline{RD}$=1, $\overline{WR}$=0, IO/$\overline{M}$=1?

---

5. Кои сигнали се користат за дефинирање машински циклус?

---

6. Кои уреди го активираат пинот INTR, а кој уред дава дозвола за прекин?

---

7. Објасни го концептот на директен пристап до меморија (Direct Memory Access)?

---

8. Како се бележи пинот за дигитски такт на микропроцесорот 8085? Колку изнесува фреквенцијата на дигитскиот такт на микропроцесорот 8085?

9. Наброј ги регистрите со специјална намена на микропроцесорот 8085?

10. Како се бележат општите регистри на микропроцесорот 8085?

11. Наброј ги видовите машински циклуси кај микропроцесорот 8085!

12. Кои се разликите меѓу машинските циклуси:
    А) пренос на операциски код и читање од меморија
    Б) читање од меморија и читање од периферен уред

13. Објасни од каде доаѓа името рестартирачки прекин?

14. Кој начин на адресирање е применет ако инструкцијата содржи адреса на мемориска локација?

15. Што претставува вториот бајт од третиот инструкциски формат кај 8085 микропроцесор?

16. Пресметај ја почетната адреса на следната инструкција ако тековната е од втор формат и почнува на адреса 4500H?

17. Објасни ја разликата меѓу инструкциите ROR и RRC!

18. Познати се содржините на регистрите A=D6H и C=2AH. Кои знаменца ќе се активираат по извршувањето на инструкцијата ADD C?

19. Дадени се содржините на регистрите A=EDH и L=669H. Пресметај ја содржината на акумулаторот по извршувањето на инструкцијата SUB L!

20. Позната е содржината на акумулаторот, A = 3AH и состојбата на знаменцето за пренос, C=1. Пресметај ги по извршувањето на инструкцијата.
    А) ROL Б) RLC

21. Познати се содржините на регистрите A=12H и E=DEH. Пресметај ги нивните содржини по извршувањето на следната програмска секвенца:

```
ADD E
CMA
MOV E,A
```

22. Мемориската локација со адреса 1234H содржи податок 7CH, акумулаторот A= 9EH и регистарот B=38H. Пресметај ја содржината на акумулаторот по извршувањето на следниве инструкции:

```
SUB B
INC A
LDA 1234H
```

23. Мемориската локација со адреса 8FFFH содржи податок 4CH, регистарот A= 12H и регистарот L=4DH. Пресметај ја содржината на акумулаторот по извршувањето на следниве инструкции:

```
LDA 8FFFH
ANA L
MOVA,L
STA 8FFFH
```

24. Каква функција имаат интерфејс-компонентите во микропроцесорските системи?

25. 8212 е непрограмибилна, еднонасочна компонента. Објасни !

26. Кои пинови од контролерот 8212 ги оспособуваат портите DATA IN и DATA OUT кога контролерот 8212 работи како леч?

27. Објасни за што служи пинот MD на контролерот 8212?

28. Објасни го адресирањето на портите A,B и C и командниот регистар на контролерот 8255?

29. Кои пинови од контролерот 8212 се користат за пренос на податоци меѓу контролерот и микропроцесорот и за пренос на податоци меѓу контролерот и периферните уреди?

30. Кога портата C на контролерот 8212 се користи како контролна, а кога како корисничка порта?

31. Определи ја содржината на командниот регистар на контролерот 8255 ако портата A работи во режим на поздравување и е влезна порта и портата B е излезна порта и работи во едноставен режим!

32. Подреди ги пиновите DATA, INTR, RD, PORTA, IBF, STB според редоследот на нивното активирање кога контролерот 8255 се користи како влезна порта!

8. 16 и 32 битни микропроцесори

8.1. Основни карактеристики на микропроцесор 8086

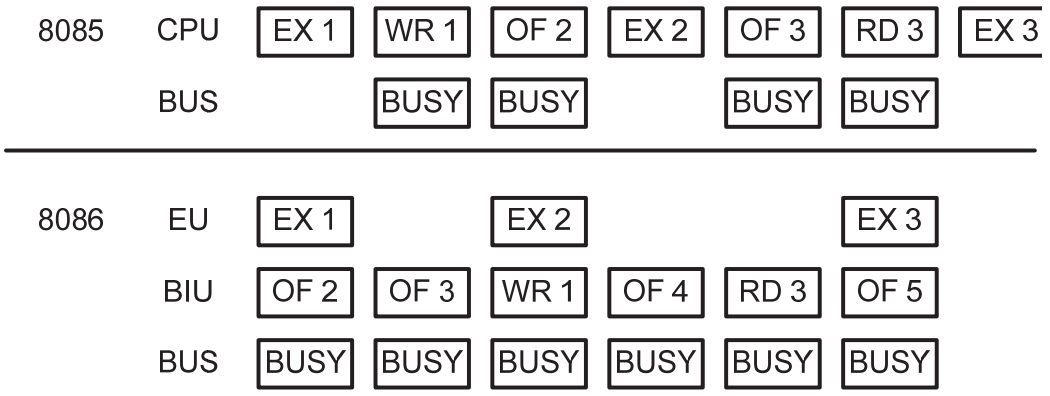
8086/8088 се микропроцесори од третата генерација. **8088** е дизајниран со **8-битна податочна магистрала** до меморијата и влезно-излезните уреди, додека **8086** има **16-битна податочна магистрала**. Во секој друг поглед микропроцесорите се идентични и софтверот напишан за едниот микропроцесор ќе функционира и за другиот. Микропроцесорите 8086/8088 имаат меморија со капацитет од 1MB и за адресирање на мемориските локации се користат 20-битни адреси.

Сите микропроцесори извршуваат програми инструкција по инструкција со постојано движење низ следниве фази:

- пренос на операциски код од меморијата во микропроцесорот (Opcode fetch)
- пренос на операндот доколку тој е побаран (READ)
- ефективно извршување (EXECUTION)
- враќање на резултатот во меморијата (WRITE)

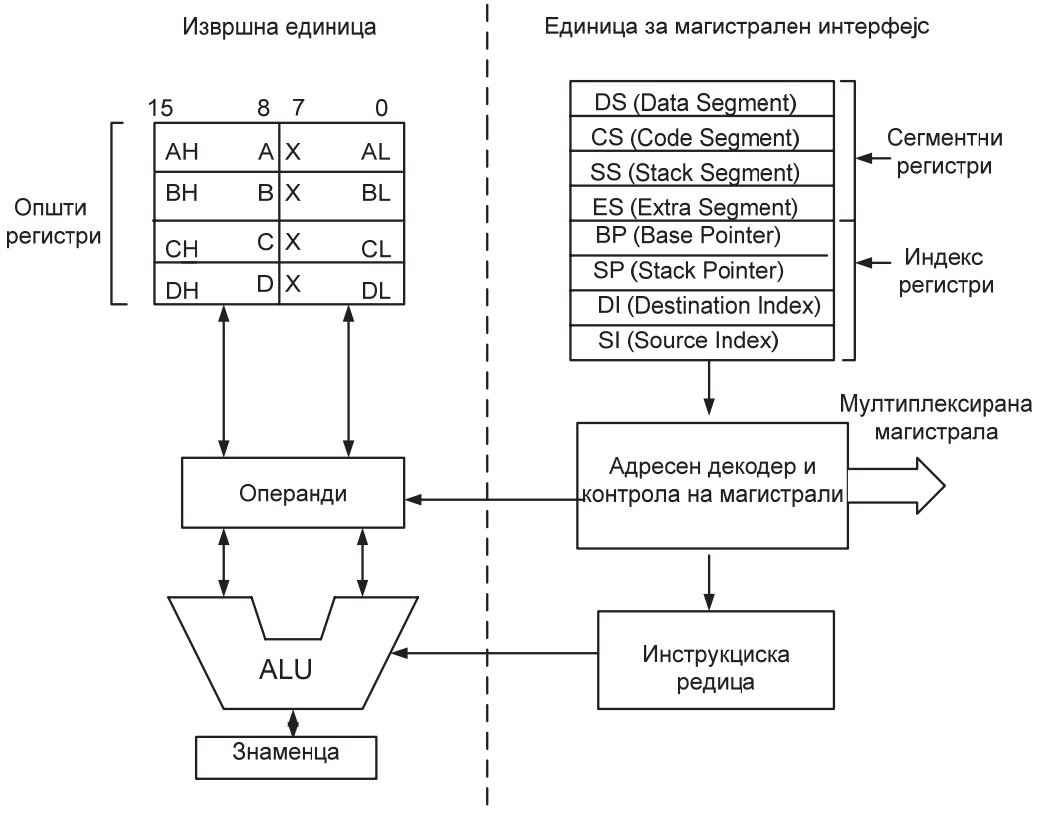
Процесорот 8086 содржи посебни единици кои паралелно извршуваат некои од наведените фази. Извршната единица (execution unit) ги извршува инструкциите, а единицата за магистрален интерфејс (bus interface unit) врши пренос на операциските кодови, операндите и резултатите. Двете единици работат истовремено.

На слика 8.1. е извршена споредба меѓу работата на микропроцесорот 8085 и микропроцесорите 8086/88. Паралелното работење на двете составни единици на микропроцесорот 8086 обезбедува максимално искористување на хардверските ресурси и поголема брзина.



Слика 8.1. Споредба во извршувањето на инструкциските фази за микропроцесори 8085 и 8086

На слика 8.2. е прикажана **функционална блок-шема** на двете единици.



Слика 8.2. Функционална блок шема на микропроцесор 8086

Извршна единица

16-битната аритметичко-логичка единица влијае врз состојбите на статусните и на контролните знаменца и манипулира со општите регистри и инструкциските операнди. Сите регистри и интерни магистрали во извршната единица се 16-битни со цел да се обезбеди брз пренос на податоци. Извршната единица нема никаква врска со надворешниот свет и инструкциите ги добива од единицата за магистрален интерфејс. Кога некоја инструкција бара пристап до меморијата или до периферен уред, извршната единица

испраќа барање до единицата за магистрален интерфејс да прими или да испрати податок. Извршната единица користи 16-битни адреси. **Единицата за магистралниот интерфејс врши адресна релокација (генерира физичка адреса) што и овозможува на извршната единица пристап до 1MB меморија.**

Единица за магистрален интерфејс

Единицата за магистрален интерфејс ја извршува целата комуникација на извршната единица со периферните уреди и меморијата. Додека извршната единица е зафатена, единицата за магистрален интерфејс гледа напред и меморира до 4 бајти однапред. Инструкциите што треба да се извршат во извршната единица се стават во таканаречена **индустриска редица на чекање.** Единицата за магистрален интерфејс иницира пренос на нова инструкција кога барем два бајти од инструкциската редица се слободни. Доколку извршната единица изврши инструкција која ја менува контролата на програмата (инструкции CALL и JUMP), тогаш единицата за магистрален интерфејс врши ресетирање на инструкциската редица, врши пренос на операциски код од новата адреса и ја предава на извршната единица, по што продолжува да ја полни редицата со новите адреси. По барање на влезно-излезните уреди, единицата за магистрален интерфејс го паузира полнењето на инструкциската редица.

Регистри

Регистри AX, BX, CX, DX се нарекуваат **општи регистри**. Општите регистри се користат како податочни регистри односно за привремено чување на податоците што треба да се обработат. Податочните регистри се единствени по тоа што можат да се користат за чување податоци со различна големина од 8 или 16 бита. Доколку сакаме да чуваме 8-битни податоци, 16-битните регистри се делат на половина. Првите помалку заначајни битови од 0 до 7 се бележат со AL, BL, CL, DL (L-low ниско), а повеќе значајните податочните битови од 8 до 15 се бележат со AH, BH, CH, DH (H-high високо).

Покажувачките и **индекс-регистрите** се користат при аритметичките и логички операции, но и при пресметување на физичките адреси за што подоцна ќе стане збор. Подолу ни е дадено значењето на кратенките за општите и покажувачките регистри.

A – акумулатор
B – основа (base)
C – бројач (counter)
D – податок (data)
DI – Destination Index – индекс на цел
SI – Source Index – индекс на извор
BP – Base Pointer – базен покажувач

SP – Stak Pointer– стек-покажувач

Сегментазицијата претставува поделба на меморијата на повеќе делови наречени сегменти. Постои разлика меѓу поимите сегмент и **сегментен регистар**. Сегментот е дел од меморијата, а сегментниот регистар е мемориска локација во микропроцесорот, кој помага да се пронајде почетокот на саканиот сегмент. Почетната адреса на сегментот се добива кога содржината на сегментниот регистар ќе се помножи со бројот 16, односно на содржината на сегментниот регистар се додаваат 4 нули од десната страна. Максималната големината на сегментот изнесува 2^{16} = 64 KB. Постојат **четири вида сегменти**. Кратенките на сегментите и нивното значење се следниве:

DS-Data Segment- Податочен сегмент
CS-Code Segment-Коден сегмент
ES- Extra Segment-Екстра сегмент
SS- Stack Segment- Стек-сегмент

Кодниот сегмент ги содржи инструкциите на програмата. Податочниот сегмент ги содржи податоците, операндите што треба да се процесираат. Екстра сегментот се користи при работа со низи и тој ја содржи целната низа. Стек-сегментот се полни и се празни преку неговата најгорна локација.

Секоја програма има свои сегменти. Колку програми имаме толку кодни сегменти ќе постојат. Но, во моментот ќе биде активен само кодниот сегмент на тековната програма и неговата почетна адреса ќе ја добиеме преку содржината на кодниот регистар. Доколку се смени програмата што се обработува, тогаш ќе се смени и содржината на сегментните регистри.

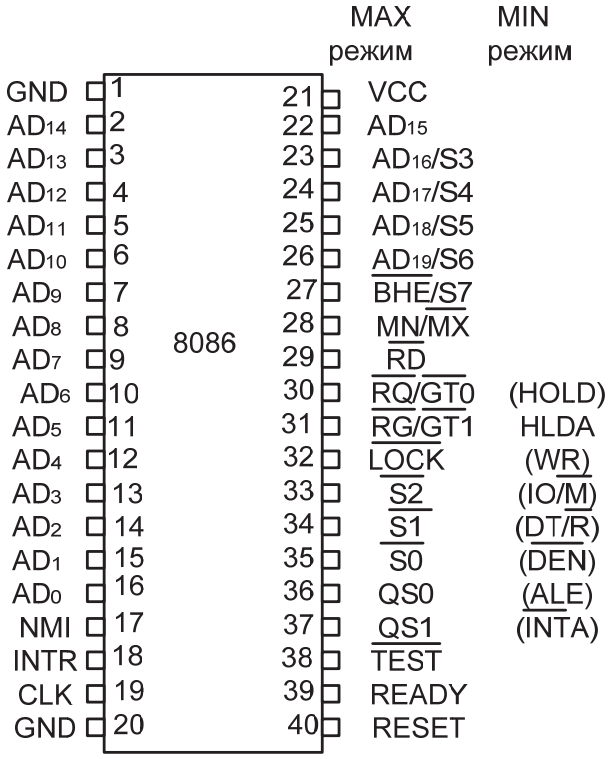
8.2. Пин дијаграм на микропроцесор 8086

Микропроцесорот 8086 има **40-пинско куќиште** и користи напојување од +5V. На слика 8.3. е прикажан пин дијаграмот на микропроцесорот 8086. Во зависност од големината и сложеноста на програмите, микропроцесорот 8086 може да работи во **два мода: минимален и максимален**. Со овие два режими ќе се запознаеме подоцна. Дел од 40-те пина имаат различни функции при минимален и максимален режим на работа. Прво ќе се запознаеме со пиновите кои имаат иста функција во двата режими на работа.

AD15-AD0 Мултиплексна адресно податочна магистрала

A19/S6-A16/S3 Мултиплексна адресно статусна магистрала. Статусниот бит S0 секогаш е на ниско ниво, битот S1 ја дава состојбата на знаменцето за прекин (IF-Interrupt Flag). Битовите S2 и S3

покажуваат кој од четирите сегменти на програмата е моментално активен.



Слика 8.3. Пин дијаграм на микропроцесор 8086

RD Овој излезен пин е активен на логичка нула и тој иницира операција читање од некој периферен уред или меморија.

READY Ако овој пин е на ниско ниво тогаш микропроцесорот влегува во состојба на чекање и останува во неа се додека не дојде до промена на состојбата на пинот.

INTR Влезен пин преку кој микропроцесорот ги прима барањата за хардверски прекини.

TEST Овој пин се тестира за време на WAIT инструкцијата. Овој пин е најчесто поврзан со нумеричкиот копроцесор 8087.

NMI Кратенката на овој пин доаѓа од зборовите Non Maskable Interrupt што во превод значи немаскирачки прекин.

RESET Овој влезен пин служи за ресетирање на микропроцесорот после што тој продолжува да ги извршува инструкциите од локација со адреса FFFF0H.

MN/MX Ако овој пин е на ниско ниво тогаш микропроцесорот 8086 работи во максимален режим, а ако е на високо ниво тогаш тој

работи во минимален режим.

$\overline{BHE}$/S7 Овој пин се бележи со кратенката на зборовите Bus High Enable и ако е нула тогаш микропроцесорот има пристап до позначајните податочни битови од D8 до D15, без разлика дали тие се дел од 16 битен податок или посебен бајт.

Во минимален режим микропроцесорот ги генерира и пренесува контролните сигнали до меморијата до периферните уреди.

M/$\overline{IO}$ Кога овој пин е на високо ниво микропроцесорот комуницира со меморијата, а кога е на ниско ниво тогаш со периферните уреди.

$\overline{WR}$ Ова е излезен пин и иницира операција пишување.

$\overline{INTA}$ Овој сигнал служи како потврда на барањето за прекин односно преставува одговор на сигналот INTR.

ALE Кога овој влезен пин е активен тогаш по адресно податочната магистрала ќе се пренесуваат адресни битови

DT/$\overline{R}$ Ознаката на овој пин е кратенка од зборовите Data Transmit /Receive што во превод значи испратени/ примени податоци. Овој сигнал ја дава насоката на движање на податоците, кон микропроцесорот или од него.

$\overline{DEN}$ Ознаката е кратенка од зборовите Data Enable што во превод значи оспособување на податоци. Овој пин е на ниско ниво секогаш кога треба да се реализира пренос на податоци по магистралата.

HOLD Ова е влезен пин преку кој DMA контролерот доставува барање за превземање на магистралите од микропроцесорот.

HLDA Ова е излезен сигнал преку кој микропроцесорот одговара на барањето за директен пренос на податоци

$\overline{SS0}$ Овој статусен сигнал заедно со сигналите $\overline{IO}$/M и DT/$\overline{R}$ служи за дефинирање на видот на машински циклус во минимален режим на работа.

Во максимален режим на работа истите пинови служат за поврзување на микропроцесорот со математичкиот копроцесор. Во тој случај контролните функции ги извршува контролерот на магистралите.

$\overline{S2}$, $\overline{S1}$ и $\overline{S0}$ Статусните битови го дефинираат видот на машински

циклус во максимален режим на работа.Овие сигнали се излезни за микропроцесорот, а влезни за контролерот на магистрали. Со овој контролер ќе се запознаеме подоцна.

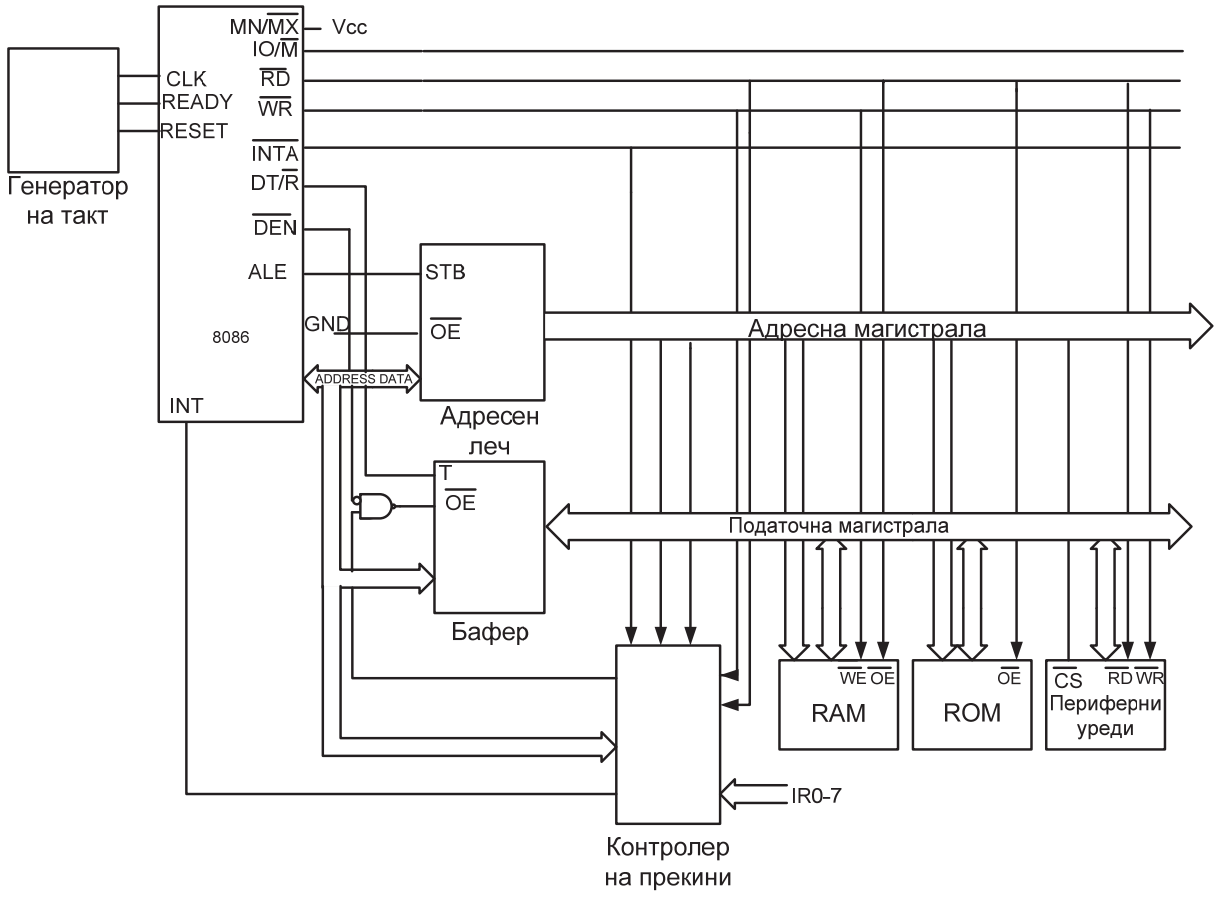
RO/GT1 и RO/GT0 Кратенката на овие пинови доаѓа од зборовите Request /Grant што во превод значи барање/доделување. Овие пинови се двонасочни и се користат за управување со дирекниот пристап до меморијата (DMA).

$\overline{\text{LOCK}}$ Овој пин служи за заклучување на периферните уреди во системот. Пинот се активира со додавање на префиксот LOCK на инструкцијата.

QS1 и QS0 Овие пинови се бележат со кратенката од зборовите queue status и служат за дефинирање на состојбата на инструкцискатa редица. Инструкцискатa редица може да биде надвор од употреба или празна, првиот бајт од неа да биде операциски код или бајтот кој следува после него.

8.3. Минимален и максимален режим на работа

Кога ја објаснувавме функцијата на пиновите на микропроцесорот 8086 спомнавме дека тој може да работи во два режими, минимален и максимален. **Изборот на режим се врши преку пинот MN/$\overline{\text{MX}}$.** Кога овој пин е на ниско ниво тогаш микропроцесорот работи во максимален режим, а кога е на високо ниво тогаш во минимален режим. Минималниот режим е многу сличен со начинот на работа на 8-битниот микропроцесор 8085 и сите програми кои ги користи микропроцесорот 8085 можат да се употребат и за микропроцесорот 8086. На слика 8.4. е прикажан **минимален микрокомпјутерски систем** на микропроцесорот 8086. Оваа варијанта е многу поевтина бидејќи **микропроцесорот сам ги генерира контролните сигнали**. Сигналите IO/M, RD и WR избираат мемориски или приферен уред за комуникација и вид на операција, читање или пишување. ALE врши оспособување на адресниот леч кога по адресно податочната магистрала се пренесуваат адресни битови. Сигналите DEN и DT/R ја одредуваат насоката на движење на податоците и го оспособуваат податочниот бафер. Тука е и сигналот INTA преку кој се одобрува прекинот кој го побарал контролерот на прекини. Сигналите кои ги користи микропроцесорот 8086 се речиси идентични со оние кои ги користи микропроцесорот 8085.



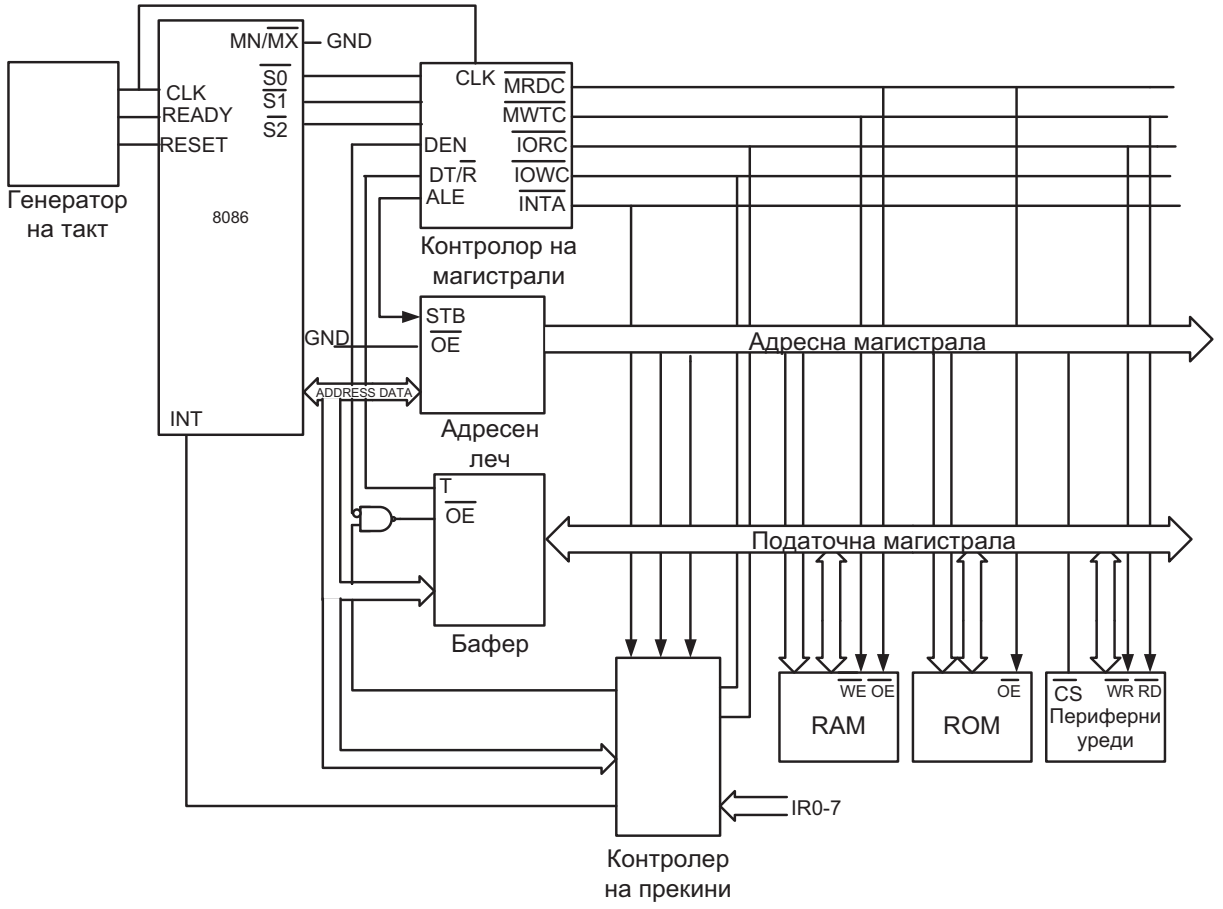
Слика 8.4. Минимален микрокомпјутерски систем 8086

Максималниот режим се користи кога во системот е вклучен еден или повеќе аритматичко копроцесори. Бидејќи микропроцесорот 8086 нема доволно пинови за управување со магистралите оваа функција ја врши контролерот на магистрали со сериски број 8288. На слика 8.5. е прикажан 8086 микрокопроцесорскиот систем кога микропроцесорот работи во максимален режим. Ако го погледнеме контролерот на магистрали ќе видиме дека тој ги генерира истите контролни сигнали кои микропроцесорот 8086 ги генерира во минимален режим.

Во минимален режим статусната линија S0 беше секогаш на ниско ниво, S1 ја даваше состојбата на знаменцето за прекин и S2 покажуваше кој е активен сегмент од тековната програма .Во максимален режим овие **статусни линии** служат за дефинирање на видот на машински циклус : пренос на операциски код , читање или пишување од меморија, читање или пишување од периферен уред, обработка на прекин. Битовите $\overline{S0}$, $\overline{S1}$ и $\overline{S2}$ се декодираат и во зависност од нивната комбинација се активираат некои од излезните контролни сигнали. Наместо сигналите $\overline{RD}$, $\overline{WR}$, IO/$\overline{M}$ се користат контролни сигнали со слична функција, но различна ознака:

- $\overline{\text{IOWC}}$ – Input Output Write Control е сигнал со кој се иницира пишување во периферен уред

- $\overline{\text{IORC}}$– Input Output Read Control се активира кога треба да се прочита податок од периферен уред
- $\overline{\text{MWTC}}$ – Memory Write Control служи за пишување во меморија
- $\overline{\text{MRDC}}$ – Memory Read Control се активира при читање од меморија.



Слика 8.5. Максимален микрокомпјутерски систем 8086

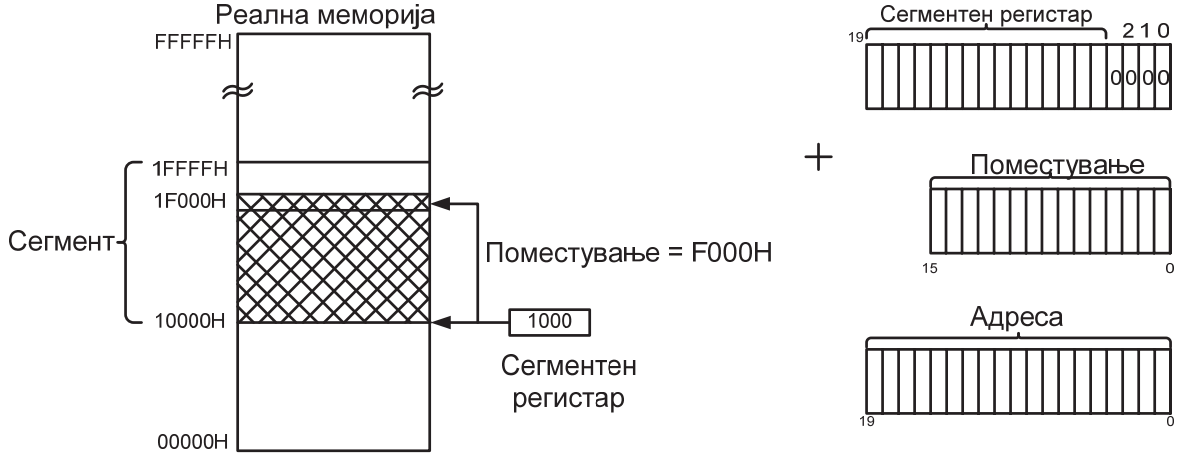
Постоењето на максималниот режим завршува со појавата на 16 битниот микропроцесор 80286.

8.4. Реален мод на работа

80286 и сите микропроцесори што следуваат по него можат да работат во два мода на работа: заштитен и реален. Единствено, микропроцесорот 8086 работи исклучиво во реален мод на работа. Реалниот мод на работа му овозможува на микропроцесорот пристап до мемориски простор од 1MB. Првиот 1MB од меморискиот простор е познат по името реална меморија. Оперативниот систем DOS функционира во реален мод на работа. Реалниот мод на работа опстанал, пред се за да се обезбеди компатибилност на стариот софтвер со новите микропроцесори. За да програмите напишани во реален

мод функционирааат во заштитен мод, тие треба да се преработат што бара вложување голем труд. За адресирање на 1MB меморија потребни се 20 адресни бита (2^{20} = 1MB).

Во реален мод на работа за формирање на физичката адреса, потребни се два дела: почетна адреса на сегментот и поместување. Почетната адреса на сегментот се добива ако 16-битната вредност на сегментниот регистар се помести 4 места влево и од десната страна се додадат 4 нули. Истиот ефект се добива ако содржината на сегментниот регистар се помножи со $16_{16}=1000_{10}$. Доколку содржината на сегментниот регистар е зададена во хексадецимален броен систем, почетната адреса на сегментот се добива ако на хексадецималната содржина на сегментниот регистар се помести за едно место влево и од десната страна се додаде една нула. Така на пример, ако сегментрниот регистар содржи вредност 1000H. тоа значи дека првиот бајт од тој сегмент ќе има адреса 10000H. Поместувањето е 16-битна вредност што се додава на почетната адреса и служи за идентификација 1 бајт од вкупно 64KB. Вредноста **64KB** претставува **максимална големина на мемориски сегмент** и се добива преку равенката 2^{16} = 64 KB, каде што 16 е број на битови во поместувањето. Поместувањето има минимална големина 0000H и тогаш тоа посочува на првиот бајт од сегментот. Максималната вредност на поместувањето изнесува FFFFH и тогаш тоа посочува на последниот бајт од сегментот. Бидејќи во реален мод на работа должината на сегментот изнесува 64KB, крајна адреса на сегментот се добива ако на неговата почетна адреса се додаде вредност FFFFH. На слика 8.6. е прикажано како микропроцесорот 8086 пристапува до RAM меморијата.



Слика 8.6. Пресметка на физичка мемориска адреса кај микропроцесор 8086

Поместувањето што се додава на почетната адреса на сегментот може да се содржи во некој регистар или да биде зададено како 16-битен број. Постојат строги правила за дефинирање на поместувањето. Така, на пример, кога работиме со кодниот сегмент, поместувањето секогаш се содржи во

инструкцискиот покажувач, а кога работиме со стек-меморијата, тогаш поместувањето може да се содржи во еден од двата регистра SP и BP. Во табелата 8.1. се дадени сите **дозволени комбинации за сегментниот регистарт – 16-битно поместување**.

Сегмент	Поместување	Примена
CS	IP	Пронаоѓање на следната инструкција
SS	SP или BP	Пристап до локацијата во стекот
DS	BX,DI,SI или 16 битен број	Пронаоѓање на операндот
ES	DI за работа со низи	Дефинирање на адресата на целта

Табела 8.1.

Во реален мод на работа секој сегмент може да има максимална големина од 64KB. Сегментите можат да се совпаѓаат. Ако некој сегмент не ги користи сите 64KB што му припаѓаат, тогаш некој друг сегмент може да се совпадне со него, односно вториот сегмент може да ги користи неискористените бајти од првиот. На ваков начин програмерот може да ја минимизаира физичката меморија потребна за реализација на програмата.

8.5. Начини за адресирање кај микропроцесор 8086

Адресните видови ќе ги објасниме преку инструкцијата MOV. Со инструкцијата MOV се пренесуваат податоци од едно место до друго. Податоците што се пренесуваат можат да бидат бајти (8 бита), зборови (16 бита), дупли зборови (32 бита) итн. Преносот може да се врши од еден регистар до друг или од регистар до меморија и обратно. Местото од кое се земаат податоци се вика **извор на операнди**, а местото каде што завршуваат саканите податоци се вика **дестинација.** На слика 8.7. дадена е MOV инструкција и насоката на движење на податоците за различни начини на адресирање. Веднаш до операцискиот код е дадена дестинацијата, а потоа следува изворот. Изворот и дестинацијата се разделени со запирка.

Почнувајќи од 8086 до 80286 во употреба се следниве начини на адресирање: адресирање на регистри, непосредно, директно, регистарско индиректно, базно индиректно, регистарско релативно и базно релативно индексно. Кај 80836 и сите подоцнежни микропроцесори постои уште еден вид адресирање скаларно индексно адресирање.

Регисарско адресирање

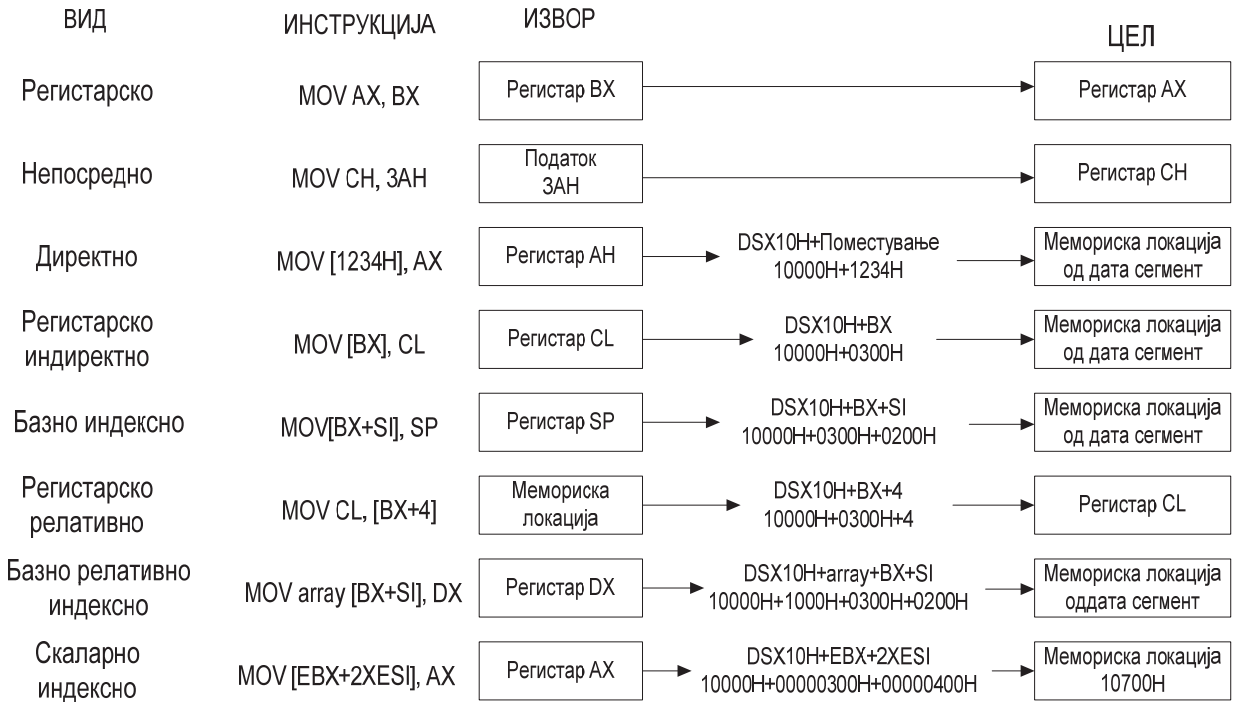
Податокот се пренесува од еден регистар во друг без да го напушти микропроцесорот. Не е дозволено пренос од еден сегментен регистар во друг, пренос меѓу регистри со различна големина и кодниот сегмент не смее да биде дестинација.

Непосредно адресирање

На местото од изворот се наведува податокот што треба да се пренесе до дестинацијата. Ако податокот е даден во бинарен броен систем, тогаш по бројот стои буквата B, а ако податокот е даден во хексадецимален броен систем, тогаш по бројот стои буквата H. Доколку податокот е даден во декаден броен систем, по бројот може да стои буквата D, но ѝ не мора.

Директно адресирање

Кај овој начин на адресирање инструкцијата ја содржи адресата на мемориската локација. Физичката адреса на мемориската локација се пресметува ако вредноста на сегментниот регистар се помножи со хексадецималниот број 10H и на тој производ се додаде поместувањето. Во асемблерските инструкции поместување како вредност е ставено во средна заграда [ ].



Слика 8.7. Начини на адресирање за микропроцесор 8086

На пример, инструкцијата MOV AL, DS:[1234H] значи копирање на еден бајт од податочниот сегмент со поместување 1234H во регистарот AL. Оваа инструкција може да се запише вака: MOV AL, [1234H]. Секогаш кога не е поинаку нагласено во инструкцијата се користи податочниот сегмент.

Програмерот кога го дефинира податочниот сегмент тој на мемориските локации може да им задава симболични имиња преку специјални псеудоинструкции наречени директиви. За програмерот е многу поедноставно да памти симболични имиња отколку хексадецимални адреси на мемориски локации. На пример, инструкцијата MOV AL, NUMBER значи да се копира бајтот од мемориската локација во податочниот сегментот со симболично име NUMBER во регистарот AL.

Регистарско индиректно адресирање

Со овој вид адресирање е дозволен пристап до која било мемориска локација, при што поместувањето се содржи во некој од следниве регистри: BP, BX, DI или SI. За кој сегмент станува збор, треба да заклучиме од индекс-регистарот според табелата од слика 8.5. Регистарот во кој се наоѓа поместувањето е во заграда [ ]. На пример, со инструкцијата MOV CX, [BX] зборот од податочниот сегмент со поместување во регистарот BX се копира во регистарот CX. Притоа, првиот бајт со адреса DSH·10H + BX се сместува во регистарот CL, а вториот бајт со адреса DSH·10H + BX + 1 во регистарот CH. Копирање податоци од една во друга мемориска локација не е дозволено. На пример, инструкцијата MOV [DI], [BX] не е дозволена.

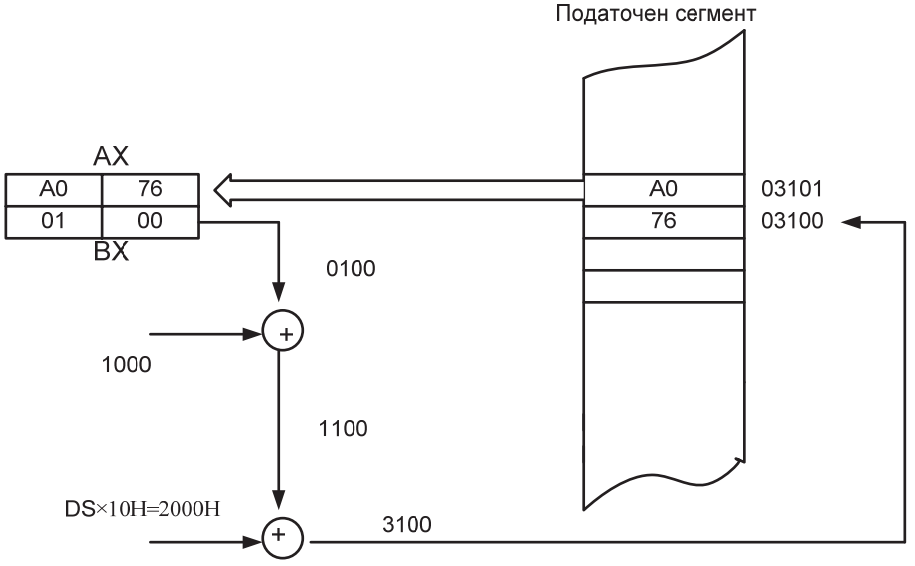
Базно индексно адресирање

Кај овој начин на адресирање поместувањето се добива со собирање на содржината на еден базен регистар (BX или BP) и еден индекс-регистар (DI и SI). Овој вид адресирање се користи при работа со низи. Базниот регистар го содржи поместувањето од почетокот на сегментот до почетокот на низата, а индекс-регистарот го содржи поместувањето од почетокот на низата до саканата локација. На пример, инструкцијата MOV [BP + DI], AH служи за копирање на зборот од регистарот AH во мемориската локација од стек-сегментот, при што поместувањето се добива со собирање на содржините на BP·10H и DI. И овде како и кај претходниот начин на адресирање доколку го користиме регистарот BP, тоа значи дека податокот се наоѓа во стек-сегментот, а доколку се користи регистарот BX тоа значи дека податокот се наоѓа во податочниот сегмент.

Регистарско релативно адресирање

Овој начин на адресирање има сличности со базно индексното и директното адресирање. Кај регистарско-релативното адресирање за да се пристапи до некоја локација од дата-сегментот треба бројот во средната заграда да се додаде на содржината на некој базен или индекс-регистар (BP, BX , DI или SI). На сликата 8.8. е даден пример за регистарско релативно адресирање. Со инструкцијата MOV AX, [BX+1000H] се копира еден збор од податочниот сегментот, кој е адресиран преку базниот регистар BX·10H, на кој се додава бројот 1000H. Да забележиме дека и кај овој вид адресирање се

користат загради [ ]. Доколку ги нема овие загради, може да донесе погрешен заклучок дека збирот од содржината на регистарот BX и вредноста 10000H не претставува адреса на мемориска локација од податочниот сегмент, туку податок. Да нагласиме дека вредноста на регистарот DS изнесува 0200H.



Слика 8.8. Пресметка на адреса кај регистарско релативно адресирање

Базно релативно индексно адресирање

Базното релативно индексно адресирање е слично со базното индексно адресирање, само што тука за да се добие растојанието меѓу почетокот на сегментот и саканата локација се собира вредноста на индекс регистарот со 16-битниот број. Пример за инструкција од ваков начин на адресирање е инструкцијата MOV AX, [BX+SI+1000H].

8.6. Инструкциско множество за микропроцесор 8086

Процесорот 8086 располага со стотина инструкции, кои можат да се поделат во неколку групи:

- Инструкции за пренос на податоци
- Аритметички инструкции
- Инструкции за поместување и ротации
- Логички инструкции
- Управувачки скок инструкции
- Инструкции за работа со низи
- Инструкции за работа со потпрограми и прекини
- Инструкции за контрола на микропроцесорот

8.6.1. Инструкции за пренос на податоци

Со значењето на инструкцијата **MOV** се запознавме кога ги проучувавме адресните видови на микропроцесорот 8086. За да се потсетиме, ќе наведеме неколку конкретни примери:

Пример 8.1.

MOV BX, AX - директен пренос на податок од еден регистар во друг
MOV BX, 1234H - пренос на податокот 1234H во регистарот BX

MOV BX , [1234H] - пренос на податок во регистар BX од мемориска локација од податочниот сегмент оддалечена од почетокот за растојание 1234H

MOV [1234H] , BX - пренос на податок од регистар BX во меморија

Постојат извесни ограничувања во употребата на MOV. Со оваа инструкција не може да се врши пренос од една мемориска локација во друга, туку со посредство на општите регистри. Не може да се пренесе податок во некој сегментен регистар, ниту да се пренесуваат податоци од еден сегментен регистар во друг.

Микропроцесорот 8086 има шест различни видови на **PUSH** и **POP** инструкции. Податокот што се пренесува во стек-меморијата може да биде сместен на различни места.

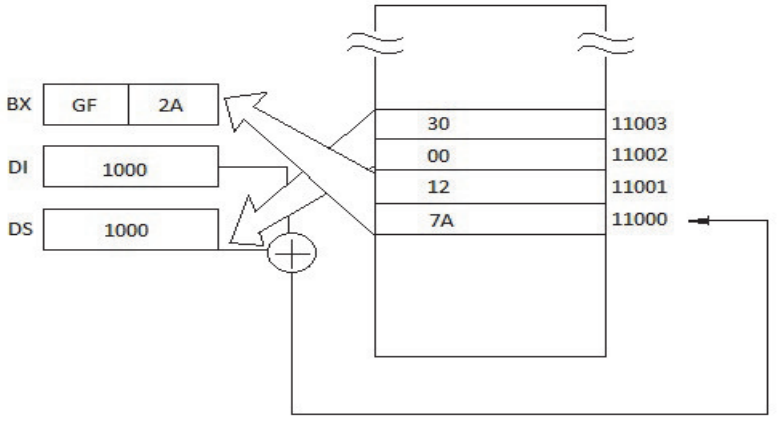
Пример 8.2.

PUSH BX - податокот е во некој општ регистар
PUSH [BX] - податокот е во мемориска локација оддалечена од почетокот на податочниот сегмент за растојание еднакво на вредноста на BX
PUSH 1225H - податокот е даден непосредно
PUSH DS - податокот е во некој сегментен регистар
PUSH A - се заштитуваат податоците во сите 16-битни општи регистри
PUSH F - се заштитува статус-регистарот со неговите знаменца

Кај микропроцесорот 8086 постојат специјални инструкции за пренос на ефективни адреси. Под ефективни адреси се подразбира поместувањето (растојанието), кое ни покажува колку е оддалечен податокот од почетокот на сегментот. Со инструкцијата **LEA** (Load Effective Address) ефективната адреса на податокот се пренесува во некој 16-битен регистар. На пример, со инструкцијата LEA BX, [DI] содржината на регистарот DI се пренесува во

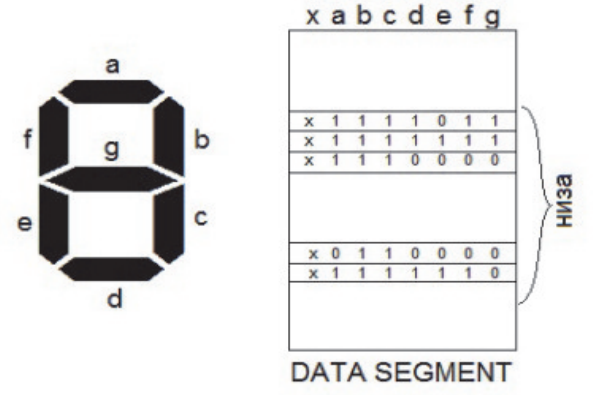
регистарот BX, а со инструкцијата MOV BX, [DI] податокот со ефективна адреса во регистарот DI се пренесува во регистарот BX.

Инструкциите **LDS** и **LES** служат за пренос на 32 бита од меморијата, при што првите два позначајни бајта се пренесуваат во сегментните регистри DS или ES, а вторите два понезначајни бајта во некој 16-битен општ регистар. Со инструкцијата LDS BX, [DI] регистарот BX се полни од локациите со адреси 11000 и 11001 а регистарот DI од локациите со адреси 11002 и 11003. По извршувањето на локацијата, DS ќе го содржи бројот 3000H, а BX бројот 127AH. Ова е прикажано на слика 8.9.



Слика 8.9. Приказ на инструкцијата LDS BX, [DI]

XLAT (Translate) Оваа инструкција се користи при конверзија од еден код во друг, како, на пример, од BCD во седумсегментен код. Во податочниот сегмент е креирана низа од 10 бајти, кои ги содржат седумсегментните кодови за броевите од 0 до 9. На слика 8.10. се прикажани седумсегментниот дисплеј и неговите кодови. Со инструкцијата XLAT содржината на AL се додава на производот BX·10H за да се добие мемориска адреса на локацијата во податочниот сегмент, а потоа содржината од таа мемориска локација се копира во регистарот AL.



Слика 8.10. Податочна низа за активирање на седумсегментен дисплеј

XCHG (Exchange) Оваа инструкција служи за размена на содржините на еден регистар со друг или со некоја мемориска локација. Оваа инструкција не може да се употреби за размена на содржините на две мемориски локации или сегментни регистри.

Инструкциите **IN и OUT** се користат при работа со влезно-излезни порти. Со инструкцијата IN податокот од влезната порта се пренесува во акумулаторот, а со инструкцијата OUT податокот од акумулаторот се пренесува до излезната порта. Постојат два типа инструкции за работа со порти. Кај **првиот тип** портите имаат фиксни адреси. Интерфејс-адресата претставува 8-битна комбинација и таа се проширува до 16 битови со додавање на осум нули од левата страна. На пример, со инструкцијата OUT 19H, AX содржината од регистарот AX се пренесува до излезната порта со фиксна адреса 0019H. Кај **вториот тип** инструкции за работа со порти, портите имаат варијабилна адреса. Адресата на портата се чува во регистарот DX, а податокот што треба да се пренесе или да се прими од портата во регистарот AL или AX. За време на извршување на програмата содржината на DX, односно адресата на портата може да се промени.

Пример 8.3:

```
OUT DX, AL      ;Податокот од регистарот AL се пренесува до
                ; излезната порта со адреса во регистарот DX.
IN AL, DX       ;Податокот од влезната порта со адреса во регистарот
                ; DX се пренесува во регистарот AL микропроцесорот.
```

8.6.2 Операции со низи (стрингови)

Низата претставува множество од податоци сместени во последователни мемориски локации од податочниот или екстра сегментот. Во едниот сегмент се сместени податоците што треба да се обработат, а во другиот сегмент податоците добиени како резултати од извршените операции. Индекс-регистрите DI и SI се регистри што го содржат поместувањето. Регистарот DI се користи во пар со сегментниот регистар ES, а SI во пар со DS. За извршувањето на операциите со низи многу е важно **знаменцето D**. Ако D=0, вредноста на регистрите DI и SI автоматски ќе се намалува за еден, а ако D=1 ќе се зголемува за еден. Инструкцијата **CLD** (Clear D) го ресетира ова знаменце, а инструкцијата **STD** (Set D) го сетира.

LODS и STOS

Со инструкцијата LODS регистарот AX или AL се полни со податок од податочниот сегмент со поместување во индекс регистарот SI. Всушност,

постојат две варијанти на оваа инструкција. Инструкцијата **LODSB** (Load Byte)се користи за пренос на 8-битни податоци, а инструкцијата **LODSW** (Load Word) за пренос на 16-битни податоци. Инструкцијата STOS има обратен ефект од LODS, односно податокот од регистарот AX и AL се пренесува во локација од екстра сегментот со поместување во DI. На инструкцијата **STOS** може да се додаде и префиксот REP кој доаѓа од англискиот збор Repeat што во превод значи повтори. Со префиксот REP инструкцијата STOS се извршува онолку пати колку што изнесува вредноста на регистарот CX. **MOVS** инструкцијата е многу корисна, бидејќи овозможува пренос на податок од една мемориска локација во друга. Да нагласиме дека ова е можно само доколку податоците се подредени во низа. Со инструкцијата MOVS податокот од дата-сегментот со покажувач во регистарот SI се пренесува во екстра сегментот со покажувач DI.

Со инструкцијата **SCAS** (String scan instruction) се врши споредба меѓу содржината на регистарот AL и блокот од бајти во екстра сегментот (SCASB) или меѓу содржината на AX и блокот од зборови во екстра сегментот (SCASW). Пред овие две инструкции може да се стави префиксот REPNE (repeat while not equal) што значи инструкцијата SCAS ќе се повторува сè доденка не се добие успешна компарација (еднакви содржини) или вредноста на CX не стане нула. Друг ваков префикс би бил REPE (repeat while equal).

Со инструкцијата **CMPS** (compare string) се споредуваат два блока од податоци, од кои едниот блок се наоѓа во податочниот сегментот со поместување во SI, а другиот блок во екстра сегментот со поместување во DI. Постојат две варијанти на оваа инструкција CMPSB - за споредба на блокови од бајти и CMPSW - за споредба на блокови од зборови. И пред овие инструкции може да се стават префиксите REPNE и REPE.

8.6.3. Аритметички инструкции

Процесорот 8086 располага со следниве аритметички инструкции: собирање, одземање, множење, компарација, негација, намалување и зголемување за еден.

Собирање и одземање

Постојат неколку видови за собирање. Имено, сите адресни модови применети кај инструкциите се применуваат и кај инструкцијата **ADD**. Ќе наведеме неколку примери:

Пример 8.4.

```
ADD AL, BL          ;Регистарско собирање, AL=AL+BL.

ADD CL, 44H         ;Непосредно собирање CL=CL+44H.
```

```
ADD [BX], AL          ;Содржината од регистарот AL се додава на
                      ;локацијата од дата-сегментот со поместување
                      ;во BX и збирот се сместува во истата локација.

ADD DL , [ BX + DI]   ;Содржината на локацијата од дата-сегментот
                      ;со поместување BX+DI се додава на
                      ;содржината на DL и збирот се запишува
                      ;во истиот регистар DL.

ADD AL, ARRAY [SI]    ;На содржината на регистарот AL се додава
                      ;содржината на еден бајт од низата ARRAY
                      ;оддалечен од почетокот на низата
                      ;за растојание SI. ARRAY е лабела со која се
                      ;означува почетокот на низата.

ADC AL, AH            ;собирање преку Carry, AL=AL+AH+Carry.
```

Аритметичкото собирање влијае врз состојбата на статус-регистарот или поточно врз знаменцата Z, C, A, S, P и D.

Аритметичкото одземање се реализира преку инструкцијата **SUB (Substract).** Сето она што беше речено за инструкцијата ADD важи и за инструкцијата SUB. Содржината на регистарот или мемориската локација по запирката се одзема од содржината на регистарот или локацијата пред запирката и во него се запишува резултатот, разликата. Одземањето со позајмување SBB (Subtraction with borrow) се реализира така што на крај од добиената разлика се одзема вредноста на знаменцето Carry.

CMP (Compare) Компарација

Оваа инструкција влијае врз состојбата на знаменцата, при што вредноста на операндите останува иста. Најчесто по инструкцијата CMP следува инструкцијата за условен скок, како што се инструкциите JA (jump above) и JB (jump below).

Пример 8.5:

```
CMP AL, 10H           ;Од содржината на регистарот AL се
                      ;одзема вредноста 10H. Содржината
                      ;на AL нема да се промени, туку може
                      ;да се промени вредноста на
                      ;знаменцето C. Ако AL>10H, тогаш
                      ;C=1, ако AL<10H тогаш C=0.
```

```
JA   START          ;Скокот ќе се изврши ако знаменцето
                    ;за пренос не е активно, односно ако
                    ;AL>10H.
```

За разлика од микропроцесорот 8085, инструкциското множество на 8086 содржи инструкции за аритметичко **множење** и **делење**. Разликуваме 8-битно и 16-битно множење. Кај 8-битното множење едниот множител секогаш се наоѓа во регистарот AL, а другиот множител во некој друг општ регистар или мемориска локација. Производот е 16-битен и се сместува во регистарот AX.

Кај 16-битното множење едниот множител се наоѓа во регистарот AX, а другиот во некој 16-битен регистар или две последователни мемориски локации. Производот е 32-битен број при што првите 16 понезначајни бита се запишуваат во регистарот AX, а вторите 16 позначајни бита во DX. Асемблерска инструкција за множење е **MUL** што е кратенка од зборот multiplay.

Пример 8.6:

```
MUL CL              ;Содржините на регистрите AL и CL се
                    ;множат и резултатот се запишува во
                    ;AX.

MUL CX              ;Содржините на регистрите AX и CX
                    ;се множат и резултатот се запишува
                    ;во регистарскиот пар AX и DX.

MUL BYTE PTR [BX]   ;Се множи содржината на AL и
                    ;содржината на бајтот оддалечен од
                    ;почетокот на податочниот сегмент
                    ;одалечен од неговиот почеток за
                    ;растојание во BX.
```

Исто како операцијата множење така и операцијата **делење** може да биде 8-битна и 16-битна. Кај 8-битното делење дељеникот е 16-битен број, сместен во регистарот AX, а делителот е 8-битен број сместен во некој 8-битен општ регистар или мемориска локација зададена со самата инструкција. По извршеното делење количникот се сместува во регистарот AL, а остатокот во AH. Во асемблер инструкцијата за делење е **DIV**.

Пример 8.7:

```
DIV CL              ;AX:CL = AL (AH остаток)
```

Кај 16-битното делење дељеникот е 32-битен број сместен во регистарскиот пар DX и AX, а делителот битен број сместен во некој регистар

или мемориска локација зададена со самата инструкција. Количникот се сместува во AX, а остатокот во DX.

8.6.4. Логички инстукции

Инструкциското множество на микропроцесорот 8086 ги содржи следниве логички инструкции: AND, OR, ексклузивно OR, NOT, TEST и NEG.

AND инструкцијата врши логичко множење и се дозволени сите адресни модови.

Пример 8.8:

```
AND AL, BL          ;AL=AL AND BL
AND DI, 4FFFH       ;DI=DI AND 4FFFH
AND AX, [DI]        ;Содржината на регистарот AX логички со множи со
                    ;податокот оддалечен од почетокот на податочниот
                    ;сегментот за растојание во DI.
AND [BX+SI], AL     ;Логичко множење меѓу содржината на AL и
                    ;содржината на локацијата оддалечена од почетокот
                    ;на податочниот сегментот за растојание BX+SI.
                    ;Резултатот се запишува во истата мемориска
                    локација.
```

AND операцијата се користи за **маскирање на** некои **битови** од еден број преку множење на овие битови со нула. Маскираните битови стануваат нула. А немаскираните (оние помножени со единица) си ја задржуваат својата вредност.

Пример 8.9:

```
      x x x x x x x x Непознат број
AND 0 0 0 0 0 0 0 1 Маска
      0 0 0 0 0 0 0 x  Резултат
```

Инструкцијата **OR** врши логичко собирање

Пример 8.10:

```
OR    AX,BX         AX=AX OR BX
OR    AX, [BP]      ;Логички се собираат содржината на
                    ;регистарот AX и зборот оддалечен од
                    ;почетокот на стек сегментот за растојание во
                    ;BP. Резултатот се запишува во AX.
OR    CL, [2555H]   ;Логички се собираат содржината на CL и
                    ;податокот оддалечен од почетокот на
                    ;податочниот сегментот за растојание 2555H.
```

;Резултатот се запишува во регистарот CL.

И со операцијата OR може да се изврши маскирање, при што маскираните битови се собираат со еден и со тоа се сетираат.

Пример 8.11:

```
  x x x x x x x x Непознат број
+ 1 1 1 1 1 1 1 0 Маска
  1 1 1 1 1 1 1 x  Резултат
```

Единствена разлика меѓу операцијата OR и ексклузивно OR, **XOR** е што кај втората операција збирот од две единици е еднаков на нула. Логичкото коло за вршење операција ексклузивно OR се вика и компаратор, бидејќи тоа на својот излез дава нула ако двата влезни бита се истоимени, а единица ако се разноимени.

Оваа операција се користи за маскирање ако треба да се инвертираат некои битови од непознатиот број.

Пример 8.12:

```
  XXXXXXXX  Непознат број
+ 0 0 0 0 1 1 1 1  Маска
  XXXXXXXX  Резултат
```

Пример 8.13:

```
XOR CH, CL     ;Содржината на регистарот CH ексклузивно сесобира
               ;со содржината на регистарот CL и резултатот се
               ;запишува во регистарот CH.
XOR AH, EEH    ;Регистар AH=регистар AH+EEH.
XOR DX, [SI]   ;Содржината на регистарот DX ексклузивно се собира
               ;со  податок оддалечен од почетокот на податочниот
               ;сегментот за растојание еднакво на вредноста на
               ;регистарот SI. Резултатот се запишува во регистарот
               ;DX.
```

Инструкцијата **TEST** е слична со инструкцијата AND со таа разлика што не се менува вредноста на операндите, туку само состојбата на знаменцата.

Инструкцијата **CMP** служи за споредба на содржината на два регистри, а инструкцијата TEST за тестирање на некои битови од некој регистар. На пример, со инструкцијата TEST AL, 01H се тестира битот со најмала тежина (првиот од десна страна). Ако овој бит е еден, тогаш знаменцето Z ќе биде нула бидјќи резултатот од AND операцијата би бил 01H. Ако тестираниот бит е нула, тогаш Z=1 (резултатот=00H). На пример, со инструкцијата TEST AL,80H се тестира битот од AL со најголема тежина (првиот од лево).

Инструкцијата **NOT** служи за пресметка на првиот комплемент на содржината на некој регистар или мемориска локација, а инструкцијата **NEG** служи за пресметка на втор комплемент односно менување на знакот. Овие две инструкции содржат само еден операнд.

Пример 8.14:

```
NOT      CH              ; Прв комплемент на регистарот CH
NEG      AX              ; Втор комплемен на регистарот AX
NOT   BYTE PTR [BGX]     ; Се комплементира бајтот од дата-сегментот
                         ; оддалечен од неговиот почеток за растојание
                         ; еднакво на вредноста BX.
```

8.6.5. Поместување и ротација

Инструкциите за поместување се познати под името **Shift** инструкции (Shift-поместување, пренесување). Со овие инструкции битовите во некој регистар или мемориска локација се поместуваат за неколку места во лево или во десно, а на испразнетите места од десна или од лева страна се внесуваат нули. Разликуваме **четири видови инструкции за поместување**: логичка Shift инструкција во лево (SHL), логичка Shift инструкција во десно (SHR), аритметичка Shift инструкција во лево (SAL), аритметичка Shift инструкција во десно (SAR). Кај последнава е битно да се зачува вредноста на битот за знак.

Пример 8.15:

```
SHL    AX, 1      ;Логичко поместување на битовите во
                  ;регистарот AX за едно место во лево.
SHR    BH, 12     ;Логичко поместување на битовите во BX за 12
                  ;места во десно.
SAR    SI, 2      ;Аритметичко поместување на битовите во  SI за
                  ;две места во десно.
```

Инструкциите за ротација вршат ротација на битовите во еден регистар или мемориска локација од еден во друг крај преку знаменцето CARRY. Постојат **четири видови инструкции за ротација**: ротација во лево (ROL), ротација во десно (ROR), ротација во лево преку Carry и ротација во десно преку Carry.

Пример 8.16:

```
ROL SI, 14      ;Битовите во регистарот се ротираат во лево 14 пати
RRC BL, 6       ; Битовите во регистарот BL ротираат во лево 6 пати
                ; преку  знаменцето Carry
```

8.6.6. Инструкции за скок и потпрограми

Исто како кај микропроцесорот 8085 така и кај микропроцесорот 8086 скоковите можат да бидат условни и безусловни. За разлика од 8085, кај микропроцесорот 8086 постојат повеќе видови условни скокови. На табелата 8.2. се прикажани сите видови условни скокови, потребните состојби на знаменцата и наивната функција.

Инструкција	**Знаменца**	**Функција**
JA (Jump if above)	Z=0, C=0	Скок ако е над (следи по инструкцијата CMP)
JAE (Jump if above or equal)	C=0	Скок ако е над или еднакво
JB (Jump if below)	C=1	Скок ако е под
JBE (Jump if below or equal)	Z=1 or C=1	Скок ако е под или еднакво
JC (Jump if carry)	C=1	Скок ако е сетирано знаменцето за пренос
JE or JZ	Z=1	Скок ако е сетирано знаменцето нула (резултатот = нула)
JG (Jump if greater than)	Z=0, S=0	Скок ако е поголемо
JGE (Jump if greater than or equal)	SF=OF	Скок ако е поголемо или еднакво
JL (Jump if less than)	SF≠OF	Скок ако е помало
JLE (Jump if less or equal)	Z=1 или SF≠OF	Скок ако е помало или еднакво
JNC (Jump if not carry)	C=0	Скок ако не е активирано знаменцето за пренос
JNE or JNZ (Jump if not equal or jump if not zero)	Z=0	Скок ако не е еднакво или не е активирано знаменцето нула
JNO (Jump if no overlow)	OF=0	Скок ако нема пречекорување
JNS (Jump if no sign)	S=0	Скок ако не е активирано знаменцето за знак (позитивен резултат)
JNP or JPO (Jump if not parity or jump if parity odd)	P=0	Непарен резултат
JO (Jump if overlow)	O=1	Скок ако постои пречекорување
JP or JPE (Jump if parity or jump parity even)	P=1	Парен резултат
JS (Jump if sign set)	S=1	Негативен резултат

Табела 8.2. Условни и безусловни скокови за микропроцесор 8086

Да нагласиме дека инструкциите **JG**, **JL, JGE, JLE, JE и JNE** се користат за броеви со предзнак, а инструкциите **JA, JB, JAE, JBE, JE или JNE** се користат за броеви без предзнак. Ако броевите се 8-битни, тогаш броеви со

предзнак ќе бидат од –128 до +127, а без предзнак од 0 до 255. На пример, ако на бројот + 127 (7FH) му се додаде бројот еден (01H) се добива резултат 80H(-128). Се активираат две знаменца OF и SF односно и двете се сетираат. Очигледно дека по извршувањето на оваа операција, може да се употреби инструкцијата JGE (+127>+1).

Постојат три вида инструкции за безусловни скокови: **краток, близок и далечен скок**. Со инструкциите за кратки скокови се вршат скокови не поголеми од +127 и -128 бајти. Со инструкциите за близок скок се вршат скокови до +/- 32 KB, односно скокови во рамките на еден мемориски сегмент. Кај блискиот скок поместувањето е 16-битно. Инструкцијата за далечен скок дозволува скок до која било локација во меморијата на микропроцесорот.

Под **циклус** се подразбира периодично повторување на неколку инструкции сè додека е исполнет бараниот услов. Циклус може да се реализира со помош на инструкциите за условни скокови. За разлика од 8085 кај 8086 постојат специјални инструкции за реализација на циклуси. Инструкцијата **LOOP** го користи регистарот CX како бројач. Со секое извршување на инструкцијата LOOP вредноста на регистарот CX се намалува за еден. Кога CX ќе стане нула, се излегува од циклусот.

Пример 8.17:

```
        MOV CX,50          ;Задавање почетна вредност на бројачот.

CIKL: …….
        …….                ;Инструкции од циклусот
        LOOP CIKL          ;Ако CX не е нула се скока на CIKL, во спротивно, се
                           ;продолжува со следната инструкција по LOOP.
```

Кај инструкциите LOOPE и LOOPNE покрај регистарот CX се анализира и состојбата на знаменцето нула (Z). Кај инструкцијата LOOPE услов за реализација на циклус е CX≠0 и Z=1. Кај инструкцијата LOOPNE услов за циклус е CX≠0 и ZF≠0.

Инструкции за работа со потпрограми се инструкциите **CALL** и **RET**. Со CALL се врши премин од главната програма во потпрограмата, а со инструкцијата RET се врши враќање од потпрограма во главната програма. Инструкцијата CALL претставува комбинација од инструкциите JMP и PUSH. Пред да се почне извршувањето на потпрограмата, содржината на регистарот инструкциски покажувач се запишува во стекот. Инструкцискиот покажувач ја содржи адресата на следната инструкција по извршувањето на потпрограмата. Оваа адреса се вика повратна адреса и е потребна за по извршувањето на потпрограмата да се вратиме кон главната програма на местото каде што сме застанале пред повикувањето на потпрограмата. Ако програмата е блиска

(NEAR), тогаш во стекот се заштитува содржината на регистарот IP, а ако потпрограмата е далечна (FAR), тогаш во стекот се заштитуваат содржините на IP и CS.

8.7. Пишување на програма за микропроцесор 8086

Програмирањето на микропроцесорот 8086 се разликува од програмирањето на микропроцесорот 8085 пред се поради **сегментизацијата**. Потеребно е да **се дефинираат** сите **сегменти** кои ќе се користат во програмата. Во податочниот сегмент ги внесуваме константите и променливите, ја дефинираме големината на стек сегментот, инструкциите се содржат во кодниот сегмент. За дефинирање на сегментите се користат специјални наредби наречени директиви. Со инструкциите вршиме обработка на податоците, а директивите се наредби, насоки со кои се уредува асемблерската програмата. Најчесто користени директиви се следниве: DB (Define Byte), DW (Define Word), DD (Define Doubleword), DUP (Duplicate), EQU (Equate), SEGMENT, ASSUME, PROC, ENDP, USES.

Со **директивите DB, DW и DD** променливите и варијабилите добиваат симболични имиња. Со овие директиви се врши резервирање на простор во податочниот сегмент и тоа за 8 и 16 битни податоци. Наместо да се памти адресата на мемориската локација во која се наоѓа саканиот податок полесно ќе се запамти симболичното име на податокот. Подоле е даден пример за употреба на овие директиви. Покрај директивата, од левата страна се дадени содржините на мемориските локации и нивните адреси. Адресите се во првата колона и тие се всушност 16 битното поместување кое ни покажува колку е одалечена локацијата од почетокот на податочниот сегмент. Гледаме дека кога резервираме простор за 8-битни податоци следната адреса се добива со додавање на еден на тековната адреса , а за 16-битни податоци додаваме 2 на тековната адреса.

Пример 8.18:

```
0000 FE     DATA1 DB 254      ;Во првата локација од податочниот
                              ;сегмент внесуваме децимален податок
                              ;256D=FEH и го именуваме со името
                              ;DATA1
0002 (??)   DATA2 DB (?)      ;Резервирање на мемориски простор од
                              ;еден бајт за променливата DATA2.
```

```
                                    ;Прашалникот означува непозната
                                    ;содржина.
0004  09F0  DATA 3  DW  2544        ;Мемориските локации со адреси 0004H
                                    ;и 0005H го содржат 16-битниот податок
                                    ;2544D=09F0H со симболично име
                                    ;DATA4.
0008  (??)  LIST DB 100 DUP (?)     ;Директивата DUP значи дуплирање на
                                    ;резервираниот мемориски простор.
                                    ;Бројот на дуплирања е даден после
                                    ;директивата DUP. Со оваа конкретна
                                    ;директива се резервира мемориски
                                    ;простор од 100 бајти со непозната
                                    ;содржина. DUP директивата најчесто ;се
                                    користи при работа со низи.
```

Директивата **EQU** се користи за креирање на лабели. Лабелите се симболични имиња на мемориски локации и начесто се користат во инструкциите за скок и потпрограми. Директивата **SEGMENT** го означува почетокот на сегментот, а **ENDS** нејзиниот крај. Директивата **ASSUME** служи за означување на податочниот, кодниот, стек и екстра сегментот кои ќе се употребат во програмата. Да се потсетиме еден ист сегмент може да биде употребен во повеќе програми. За означување на крајот и почетокот на потпрограмите се користат директивите **PROC** и **ENDP**. Со директивата **USES** автоматски се заштитуваат содржините на регистрите на врвот од стек меморијата.

Подоле напишаната потпрограма врши собирање на содржините на општите регистри и бројот 100. На крајот резултатот од собирањето се зачувува во локација од податочниот сегмент наречена ZBIR.

```
TITLE SOBERI                ;Зборот TITLE значи наслов.
                            ;Програмерот го избира името на
                            ;програмата по своја желба.
SSEG SEGMENT STACK          ;Го дефинираме стек сегментот со
     DB 256 (?)             ;големина од 256 бајти.
SSEG ENDS

DSAG SEGMENT                ;Го дефинираме податочниот сегмент.
     BROJ DB 100            ;Тој во себе содржи една константа 100
     ZBIR DB ?              ;и една променлива со непозната
DSEG ENDS                   ;содржина наречена ZBIR.

CSEG SEGMENT                ;Го дефинираме кодниот сегмент.
```

```
ASSUME CS:CSEG, DS:DSAG,        ;Нагласуваме кои сегменти ќе се
SS:SSEG                         ;користат во програмата.

SOBERI PROC FAR                 ;Ја именуваме потпрограмата

        PUSH DS                 ;Го заштитуваме податочниот сегментен
        MOV AX,0                ;регистар и акумулаторот на врвот од
        PUSH AX                 ;стекот.

        MOV BX, DSEG            ;Го внесуваме новиот податочен
        MOV DS, BX              сегмент

      MOV AL, BROJ              ;На содржината на AL прво го додаваме
      ADD AL, BL                ;бројот 100 , а потоа и вредностите на
      ADD AL, CL                ;другите регистри BL, CL и DL.
      ADD AL, DL
      MOV ZBIR, AL              ;Добиениот резултат го запишуваме во
                                ;локацијата со симболично име ZBIR.

RET                             ;Инструкција за враќање од
                                ;потпрограмата во главната програма.
SOBERI ENDP                     ;Крај на потпрограмата.
CSEG ENDS                       ;Крај на кодниот сегмент.
      END SOBERI                ;Крај на програмата
```

8.8. Интегрирани кола за формирање микрокомпјутерски систем 8086

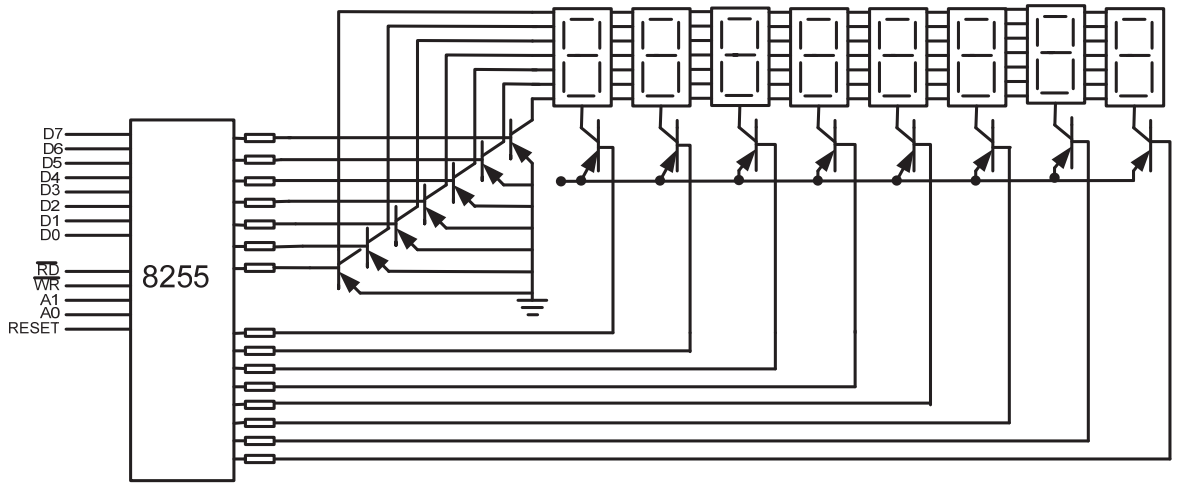
8.8.1. Употреба на интегрирано коло 8255

За поврзување на микропроцесорот 8086 со влезно-излезните уреди се користи програмибилната интерфејс-компонента 8255. Со пин-дијаграмот, програмирањето и режимите на работа на оваа компонента се запознавме уште кога зборувавме за микропроцесорот 8085. Колото **8255 наоѓа примена и во микрокомпјутерските системи на Пентиум микропроцесорите.** За ја разбереме функцијата на ова интегрирано коло ќе разгледаме два конкретни примери: поврзување со седум сегментен дисплеј и поврзување со тастатура

На сликата 8.11. е претставено **коло за поврзување на микропроцесорот 8086 со седумсегментен дисплеј**. Седумсегментниот дисплеј е составен од осум полиња. За испраќање податоци до дисплејот се

користи постапката на мултиплексирање. **Податоците** до сите полиња од дисплејот **се испраќаат преку** пиновите на **портата А**, но не истовремено, туку во различни временски интервали. **Изборот на поле се врши преку портата В.** Пиновите на портата В се поврзани со базите на PNP транзистори, кои работат како прекинувачи.

Само еден пин од портата В ќе биде на ниско ниво, а останатите пинови ќе бидат на високо ниво. Пинот со ниво на логичка нула ќе го вклучи прекинувачот (PNP транзисторот) и ќе го селектира тоа поле од дисплејот. Останатите седум полиња од дисплејот се неактивни. На портата В се испраќа податок составен од една нула и седум единици. Нулата во тој податок постојано се ротира преку инструкциите за ротација во лево или во десно. Полиња во дисплејот се активираат последователно, еден по друг, почнувајќи од лево кон десно или обратно што зависи од применетата инструкција за ротација. Ние не можеме да забележиме дека сите полиња од дисплејот не светат истовремено, бидејќи брзината со која се активираат полињата е многу голема, па имаме впечаток дека сите полиња светат истовремено.

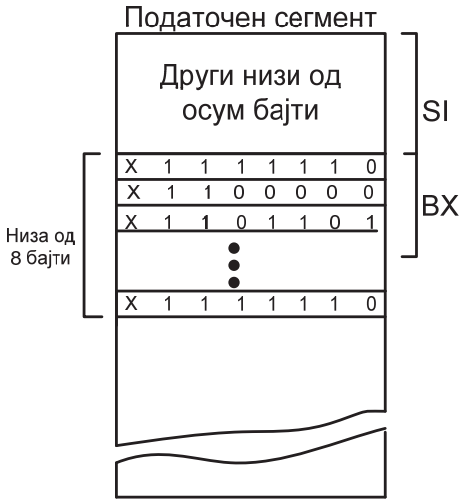


Слика 8.11. Поврзување на микропроцесор 8086 со седумсегментен дисплеј

Пред да започнеме со анализа на потпрограмата која врши мултиплексирање на седумсегментниот дисплеј со осум полиња, да се потсетиме на седумсегментните кодови, бидејќи токму тие се испраќаат преку портата А до дисплејот.

Во податочниот сегмент е креирана низа од 8 бајти кои ги содржат осумте **7-сегментни кодови** што треба да се пренесат до осумте полиња од дисплејот. Оваа низа во податочниот сегмент ја пронаоѓаме со помош на регистарот SI, кој всушност, го дава растојанието од почетокот на податочниот сегментот до почетокот на низата. Првиот бајт, код од низата треба да се внесе во првото поле лево, вториот бајт во полето до него, третиот бајт во полето трето од лево итн. Регистарот BX служи за движење по низата од 8 бајти, и тоа почнувајќи од дното кон врвот. Вредноста на регистарот BX се намалува за

еден сè додека не добие вредност нула. Податочниот сегмент со низата од 8 бајти е прикажана на слика 8.12.



Слика 8.12. Податочен сегмент со седумсегментни кодови

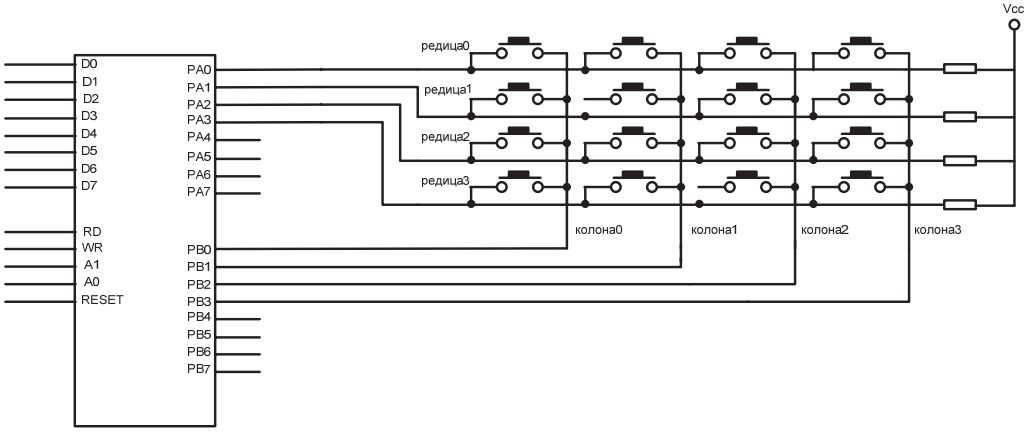
Во потпрограмата за мултиплексирање на дисплејот се повикува нова **потпрограма DELAY**. Со оваа потпрограма се внесува доцнење од 1ms меѓу активирањето на две соседни полиња од дисплејот. Изборот на времето на доцнење е во согласност со препораките на некои производители на лед дисплеи.

Во потпрограмата за мултиплексирање портата А има адреса 700H, а портата В 701H.

```
DISP   PROC
       PUSHF            ;Содржнината на статус-регистарот се
                        сместува ;на врвот од стек=меморијата
       MOV  BX,8        ;Бројачот се полни со вредност 8
       MOV  AH,7FH      ;Се избира бит маската =01111111B за
                        селекција на полињата.
       MOV  DX ,701H    ;Адресирање на портата В
       MOV SI, POCETOK  ;Пронаоѓање на низата од 8 бајти во
                        ;податочниот сегмент
DISP1   MOV  AL, AH     ;Селекција на поле
       OUT  DX,AL
       DEC  DX          ;Адресирање на порта А
       MOV AL,(BX+SI)   ;Се испраќа 7-сегментниот код
       OUT  DX,AL
       CALL  DELAY      ;Се чека 1ms
       ROR  AH,1        ;Пристап до следното поле вдесно
       INC  DX          ;Адресирање на портата В
       DEC  BX          ;Намалување на бројачот за еден
       JNZ  DISP        ;Повтори осум пати
```

```
        POPF            ;Враќање на статус-регистарот
        RET             ;Крај на потпрограмата
DISP    ENDP
```

Сега ќе ја објасниме **примената на 8255 компонентата при поврзување со тастатура**. Тастатурите можат да бидат со најразлични големини. Стандардната тастатура содржи 101 копчиња. Во примерот што е прикажан на слика 8.13. тастатурата содржи 16 копчиња. 16 копчиња се распоредени во четири редици и четири колони. На ваков начин се формира матрица со големина 4×4. **Портата А** е влезна порта и **служи за читање на состојбата на копчињата од редиците**.



Слика 8.13. Поврзување на микропроцесор 8086 со тастатура

Бидејќи има четири редици, ќе бидат употребени четири пина од портата А (PA_0-PA_3). Секоја редица е поврзана на напон Vcc=5V преку pull up отпорници од 10КΩ. Ако ниедно копче од набљудуваната редица не е притиснато, тогаш пинот од портата А ќе биде на ниво на логичка единица. **Портата В е излезна порта и служи за селекција на една колона** . Бидејќи има четири колони, ќе бидат употребени четири пина од портата В (PB_0-PB_3). Селекцијата на колона ќе ја објасниме преку пример. Ако 1110 е излезен податок за четирите пина PB_3-PB_0, тогаш ќе биде селектирана нултата колона. Од четирите пина на портата В само пинот PB0 е на ниво на логичка нула. Ако некое од копчињата од 0 до 3 биде притиснато, тогаш автоматски се носи логичка нула на тој пин од портата А. Копчињата од другите редици можат да бидат притиснати, но ова нема да влијае врз логичката состојба на пиновите од портата А. Ако 1101 е излезен податок за портата В, тогаш ќе бидат селектирани копчињата што припаѓаат на колоната број еден.

Копчињата од тастатурата не се прекинувачи, туку се тастери. Откако копчето ќе се притисне, тоа не останува во таа положба, туку се враќа назад односно се отпушта**. Отпуштањето на копчето трае неколку делови од секундата.** За нас тоа е нереално време, но за микропроцесорот тоа може да

биде долг временски интервал. Може да се случи микропроцесорот да ја испита состојбата на притиснатото копче пред да се ослободи тоа. Микропроцесорот ќе го забележи ова како второ притискање што, се разбира, нема да биде точно. Поради тоа треба да се воведе доцнење меѓу две последователни проверки на копчињата за да не доаѓа до вакви грешки.

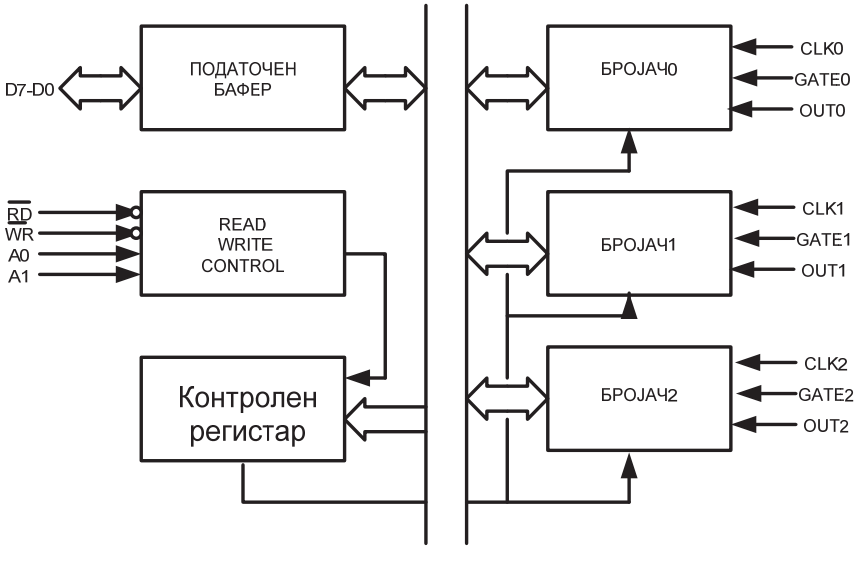
8.8.2. Програмибилен тајмер 8254

Процесното управување бара дигитски такта со многу помали фреквенции од оние што се користат во микропроцесорските системи. Влезната фреквенција е 8MHz. **Програмерот може да ги менува фреквенцијата и обликот на импулсите на дигитскиот такт**. Програмибилниот тајмер 8254 се користи за создавање дигитски такт за проверка на пинот INTR (18.2 Hz) и тактот за освежување на DRAM меморијата. На сликата 8.14. е прикажан блок-дијаграмот на овој тајмер.

Тајмерот 8254 има четири внатрешни регистри: бројач 0, бројач 1, бројач 2 и команден регистар. За адресирање на овие регистри се користат адресните влезови A1 и A0. Адресирањето се врши како што е прикажано на табела 8.3.

A1	A0	регистар
0	0	Бројач0
0	1	Бројач1
1	0	Бројач2
1	1	Контролен регистар

Табела 8.3. Адресирање на внатрешни регистри во програмибилен тајмер 8254

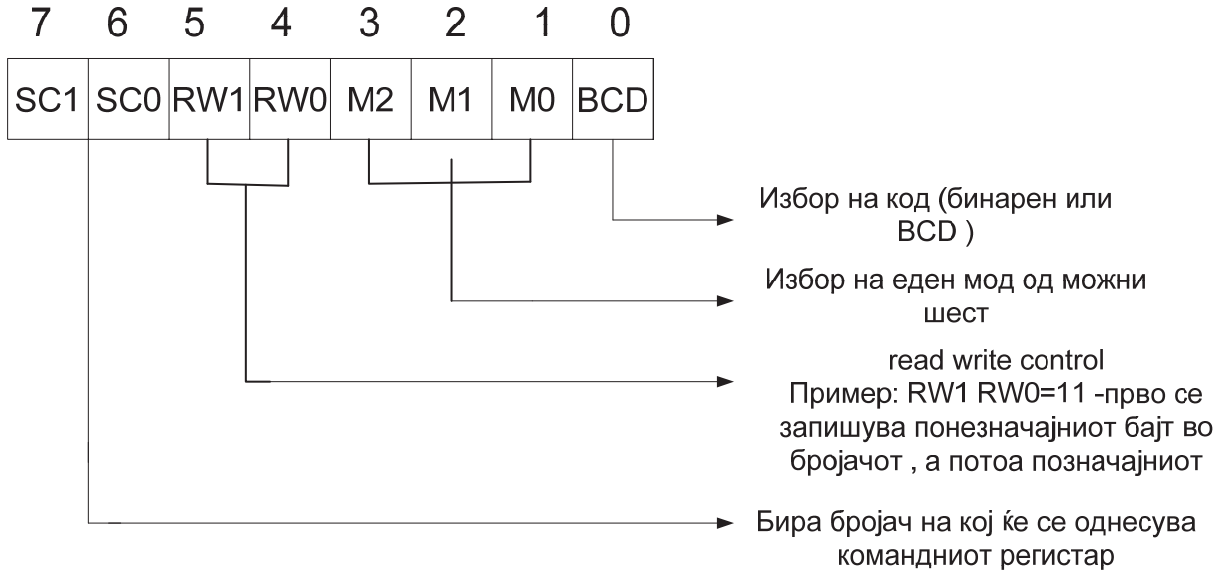


Слика 8.14 Блок шема на програмибилен тајмер 8254

Секој бројач има по 3 пина. CLK е за влезниот дигитски такт, OUT е за излезниот и G (gate) е пин за контрола и оспособување на бројачот. Бројачите

се 16-битни. Максимималната вредност на која може да се испрограмира еден бројач е FFFFH.

Програмирањето се врши преку командниот регистар. Преку командниот регистар се избираат модот и бројачот. Модот го дефинира обликот на излезниот сигнал во зависност од излезниот. На сликата 8.15. е дадено значењето на секој бит од овој регистар.



Слика 8.15. Функциски опис на битовите во командниот регистар на програмибилниот тајмер 8254

Тајмерот 8254 може да работи во 6 различни мода и тие се прикажани на слика 8.16. со временски дијаграми.

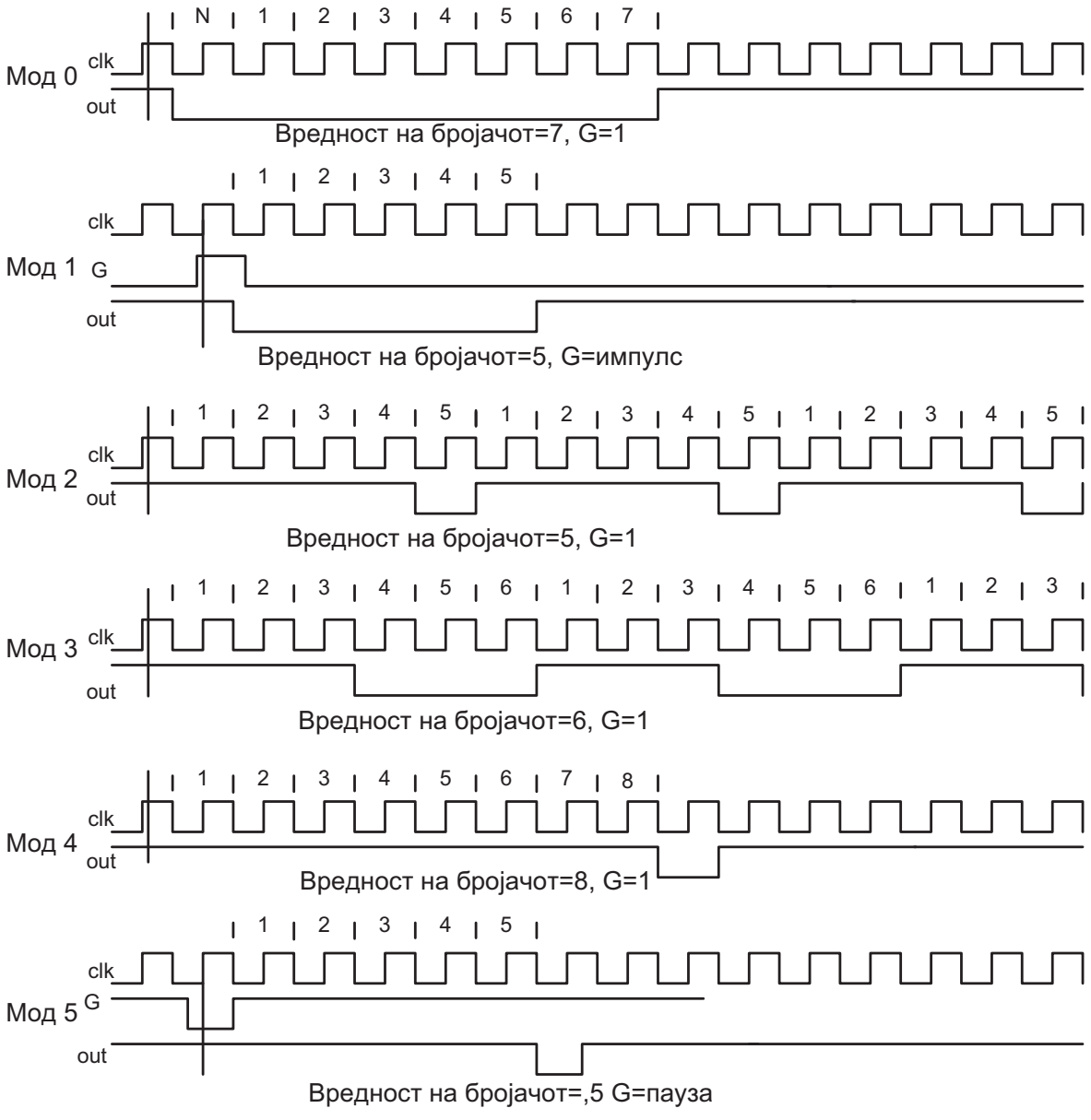
Мод 0 – Во овој мод на работа пинот OUT ќе биде логичка единица сè додека трае броењето. Броењето започнува во моментот кога ќе се запише вредноста на командниот регистар. Додека трае броењето, пинот G треба да биде на високо ниво. Ако додека трае броењето пинот G падне на ниско ниво, тогаш броењето ќе биде прекинато и ќе продолжи кога сигналот на пинот G пак ќе стане логичка единица.

Мод 1 – Бројачот се програмира и чека импулс на пинот G за да почне броењето. Ако се појави уште еден импулс на пинот G додека трае броењето, тогаш штом ќе заврши тековното броење, ќе почне ново .

Мод 2 – Кај овој мод на работа сигналот на пинот G треба да биде на високо ниво цело време. На пример, ако бројачот се испрограмира на вредност 10, тогаш на пинот OUT сигналот ќе биде логичка единица девет дигитски такта, а во десетиот такт ќе биде нула. Поворката од импулси трае сè додека не се изврши репрограмирање или сигналот на

пинот G не стане логичка нула. Ваквата поворка се вика поворка од континуирани импулси.

Мод 3 – Исто како и кај предходниот мод, така и овде е потребно гејтот да биде на високо ниво. На пример, ако COUNT = 4, тогаш сигналот на пинот OUT ќе биде логичка единица два такта, а други два такта ќе биде нула. Овој сигнал се нарекува квадратен сигнал.

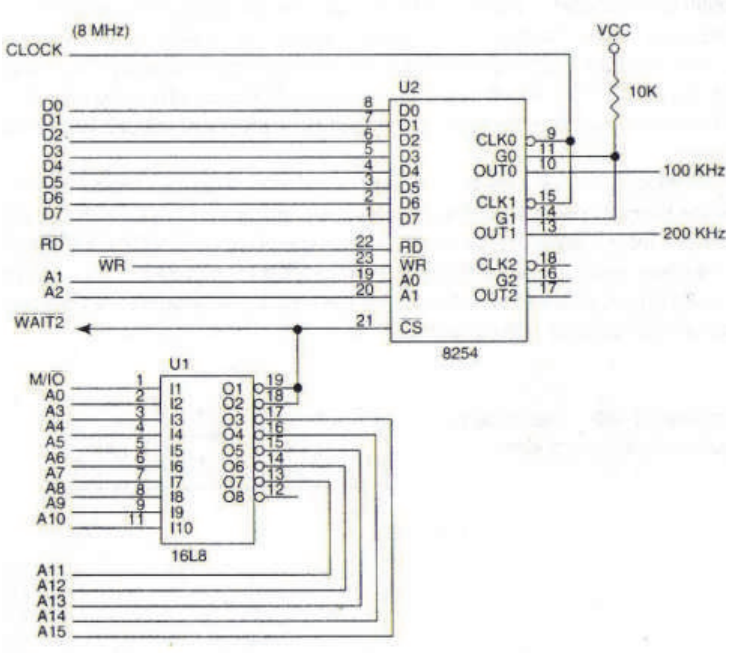


Слика 8.16. Временски дијаграм за зависноста на излезниот сигнал од влезните сигнали кај програмибилен тајмер 8254

Мод 4 – Како кај претходните два мода и овде е потребно гејтот да биде на високо ниво. На пинот OUT ќе се појави само еден импулс откако ќе бидат изброени онолку такта на колку што е програмиран бројачот.

Мод 5 – Овој мод е ист како и мод 4, со таа разлика што за да почне броењето на пинот G мора да се појави импулс односно сигналот од 1 да падне на нула, а потоа повторно да скокне на 1.

На слика 8.17. е прикажана шемата за поврзување на тајмерот 8254 со микропроцесор 8086 со работна фреквенција 8MHz.



Слика 8.17. Поврзување на тајмер 8254 со микропроцесор 8086

Пример 8.20:

Одреди ги содржините на командниот регистар и бројачите во тајмерот 8254 така што на излез од нултиот бројач да се добие квадратен сигнал со фреквенција 100KHz, а на излез од бројачот 1 да се добие поворка од континуирани импулси со фреквенција 200KHz.

Решение:

Содржината на командниот регистар ќе биде различна за секој бројач посебно. За бројачот 0 го користиме мод 3 и бројачот 0 го програмираме на вредност 80 (8MHz : 80 = 100KHz) . За бројачот 1 користиме мод 2 и бројачот го програмираме на вредност 40 (8MHz : 40 = 200KHz).

Потпрограмата за реализација на објаснетата цел се вика TIME. Тајмерот 8254 ги користи интерфејс-адресите 0700H, 07002H, 0704H и 0706H.

```
TIME    PROC    NEAR
        PUSH    AX          ;ја заштитуваме содржината на
        PUSH    DX          регистрите

        MOV     DX, 706H    ;адреса на команден регистар
        MOV     AL,00110110B ;содржина на команден регистар
```

```
        OUT     DX,AL           ; за бројач 0
        MOV     AL,01110100B    ; содржина на команден регистар
        OUT       DX,A L          ; за бројач 0

        MOV     DX,700H         ;адреса на бројач 0
        MOV     AL,80           ;програмирање на бројач 0
        OUT     DX,AL

        MOV     DX,702H         ;адреса на бројач 1
        MOV     AL,40           ;програмирање на бројач 1
        OUT     DX,AL

        POP     DX
        POP     AX
        RET
TIME    ENDP
```

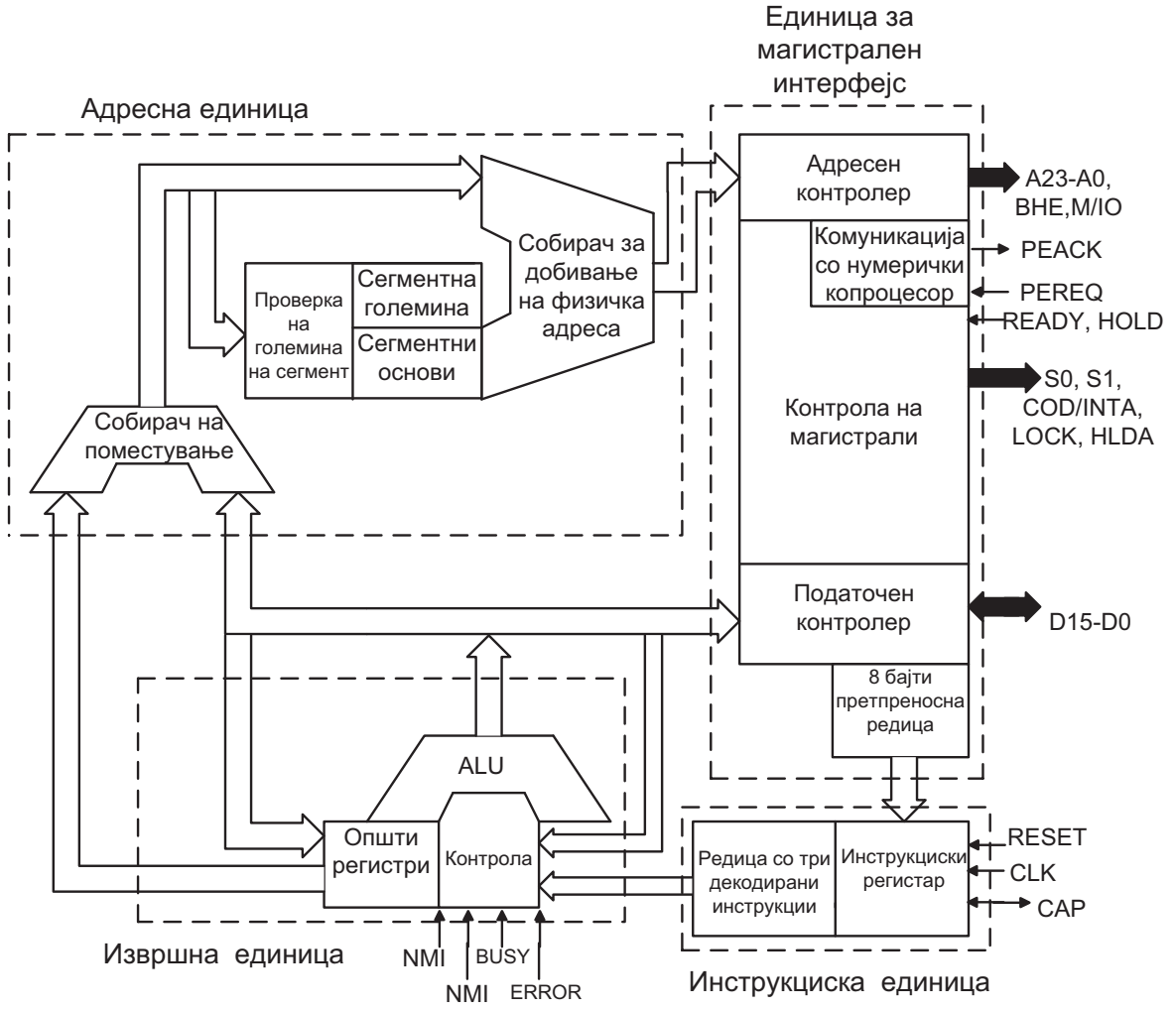
8.9. Основни каратеристики на микропроцесор 80286

Процесорот 80286 е напредна верзија на микропроцесорот 8086, со можност за паралелно извршување на повеќе задачи и кориснички услуги. Микропроцесорот 80286 располага со физичка меморија од 16MB и виртуелна меморија од 1GB, преку употреба на специјална единица за мемориски менаџмент. Бројот на дигитски такта за извршување на инструкциите е намален. 80286 микропроцесорот повеќе не се користи во персоналните компјутери, но тој се уште наоѓа примена во компјутерските системи за процесно управување.

На слика 8.18. е прикажана **архитектурата** на овој микропроцесор.

Единицата за мемориски менаџмент е именувана како адресна единица. Слично како кај микропроцесорот 8086, за пресметка на физичката адреса се користат сегментните регистри и помесувањето (растојание од почетокот на сегментот до саканата локција). Аритметичко логичката единица и општите регистри се 16-битни. Единицата за управување со мегистралите ги генерира статусните сигнали за избор на машински циклус (S0,S1), за управување со DMA контролерот (HOLD, HLDA) и прекините (INTR, COD/INTA). За разлика од 8086 микропроцесорот, микропроцесорот 80286 содржи две инструкциски редици, една пред инструкцискиот декодер и друга со декодирани инструкции. Адресната магистала е 24-битна и се користи за адресирање на реален

мемориски простор од 16MB. Да нагласиме дека микропроцесорот 80286 не користи мултиплексна адресно податочна магистрала.

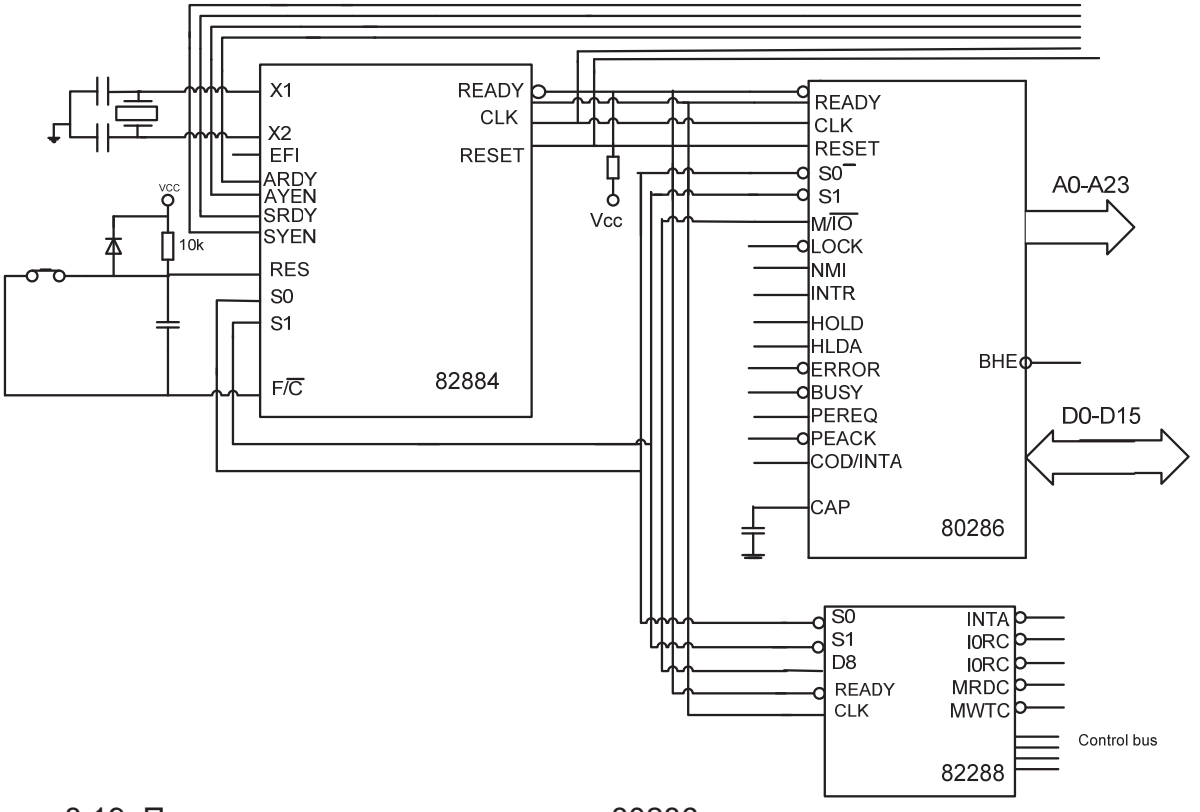


Слика 8.18. Архитектура на микропроцесор 80286

Пиновите BUSY, ERROR, PEREQ и PEACK се нови и тие се користат за комуникација на микропроцесорот 80286 со математичките копроцесори. Значењето на пинот BUSY ја има истата функција како и пинот TEST на микропроцесорот 8086 и се користи за вметнување на цел број тактови за чекање. Пинот ERROR се активира доколку се појави грешка во контролата на парност. Преку пиновите PEREQ и PEACK математичкиот копроцесор поднесува барање и добива дозвола за работа.

На слика 8.19. е прикажан начинот на поврзување на микропроцесорот 80286 со **контролерот на магистрали 82288 и генераторот на такт 82884**. Овие две **интегрирани кола** се дел од микрокомпјутерскиот систем на микропроцесорот 80286. За подобро разбирање на овој систем, истиот треба да го споредиме со системот на микропроцесорот 8086 кога тој работи во максимален режим, слика 8.5. На таа слика е употребен контролер на магистрали 8288 што е многу сличен со контролерот 82288 прикажан на

сликата 8.19. Начинот на генерирање на контролните сигнали е ист. Микропроцесорот ги испраќа статусните битови S0, S1 и IO/$\overline{M}$, тие се декодираат и на излез од контролерот се добиваат сигналите за контрола на меморијата (MRDC, MWTC) и контрола на периферните уреди (IORC, IOWC).



Слика 8.19. Поврзување на микропроцесор 80286 со генератор на дигитски такт 82884 и контролор на магистрали 82288

Генераторот на дигитски такт 82284 е многу сличен со генераторот на такт 8284. Излезни пинови се:

33. CLK (clock)- Овој сигнал е влезен сигнал за микропроцесорот и другите уреди во ситемот
34. READY-Кога овој пин е на ниско ниво микропроцесорот влегува во состојба на чекање. Кога овој пин е на високо ниво тој нема влијание врз аботата на микропроцесорот
35. RESET-Овој пин служи за ресетирање на микропроцесорот.
36. PCLK (Peripheral Clock)-Овој излезен дигитски такт има два пати помала фреквенција од фреквенцијата на дигитскиот такт на пинот EFI

Влезни пинови на генераторот на такт 82284се:

- X1, X2-Овие два пина служат за приклучување на кристалниот осцилатор
- EFI(External Frequency Input)-Овој пин служи за приклучување на надворешен извор на дигитски такт.

- RES (Reset)- Преку овој пин се врши ресетирање на напојувањето и истиот е поврзан со RC коло.
- F/$\overline{C}$ (Frequency Crystal Select)-Преку овој пин се избира изворот на дигитскиот такт: X1, X2 или EFI.
- ARDY, SRDY (Asynchronous Ready Enable, Synchronous Ready Enable)- Овие пинови служат за избор на еден од можните два извори на сигналот READY.
- AYEN, SYEN (Asynchronous Ready, Synchronous Ready)- Кога еден од овие пинови е на ниско ниво се активира излезниот сигнал READY
- S0, S1-Статусните пинови ги испраќа микропроцесорот и служат за дефинирање на вид на машински циклус.

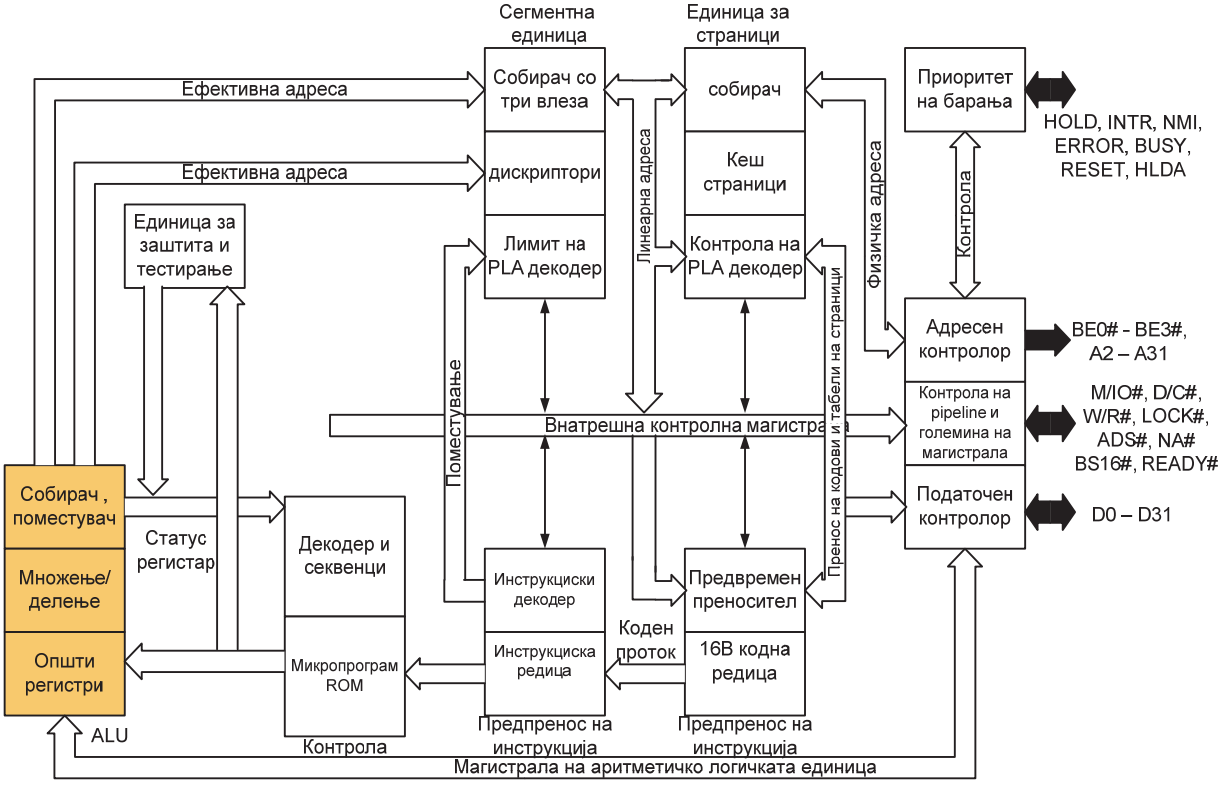
8.10. Основни карактеристики на микропроцесор 80386

Процесорот 80386 е 32-битна верзија на микропроцесорот 8086. Покрај дуплирањето на големината на податоците, кај микропроцесорот 80386 постојат низа други подобрувања во мултипроцесирањето, мемориската организација, виртуелната меморија, заштита на софтверот итн. Со заштитниот и виртуелниот мод на работа ќе се запознаеме во следната тема. **Податочната и адресната магистрала се 32-битни.** Големината на меморискиот простор изнесува 4GB. Микропроцесорот 80386 обезбедува премин од реален во заштитен режим на работа без ресетирање на микропроцесорот, што не беше случај со претходните генерации на микропроцесори.

На слика 8.20. е прикажана архитектурата на микропроцесорот 80386 и неговите **влезни и излезни сигнали**.

Микропроцесорот 80386DX има 30 адресни линии, од A2 до A32 и служат за избор на 32-битна локација од вкупно 1G локации. Наместо адресните линии A0 и A1 се користат четирите контролни сигнали $\overline{BE3}$-$\overline{BE0}$ за избор на еден бајт од четирите бајти во избраната локација. Пинот M/$\overline{IO}$ се користи за избор на уред (1-меморија, 0-периферен уред), а пинот W/$\overline{R}$ за избор на операција (1-пишување, 0-читање). Пинот $\overline{ADS}$ (Address Data Strobe) се активира секогаш кога се испраќа валидна мемориска или интерфејс адреса. Пинот CLK2 дава сигнал со два пати поголема фреквенција од онаа на микропроцесорот. Со активирање на пинот RESET микропроцесорот започнува извршување на софтвер од локација со адреса FFFFFFF0H. Сигналот $\overline{READY}$го контролира бројот на состојби на чекање. Кога пинот D/$\overline{C}$ (data/control) е на високо ниво тогаш се пренесува податок, а ако е на ниско ниво тогаш микропроцесорот извршува машински циклус за обработка на прекин. Пинот $\overline{BS16}$ (Bus Select

16) избира меѓу 16 или 32-битна магистрала. Сигналот $\overline{NA}$ (Next Address) се користи кај pipeline концептот за испраќање на адресата на следната инструкција. Со пиновите HOLD и HLDA се контролира DMA преносот. Аритметичкиот копроцесор бара дозвола за работа од микропроцесорот 80386 преку активирање на пинот $\overline{PEREQ}$ (Coprocessor Request). Пинот $\overline{BUSY}$ е влезен пин за микропроцесорот и е активен кога копроцесорот е зафатен. Доколку се појави грешка во работењето на копроцсорот се активира пинот $\overline{ERROR}$. Пиновите INTR и NMI се активираат во случај на хардверски прекин или немаскирачки прекин.

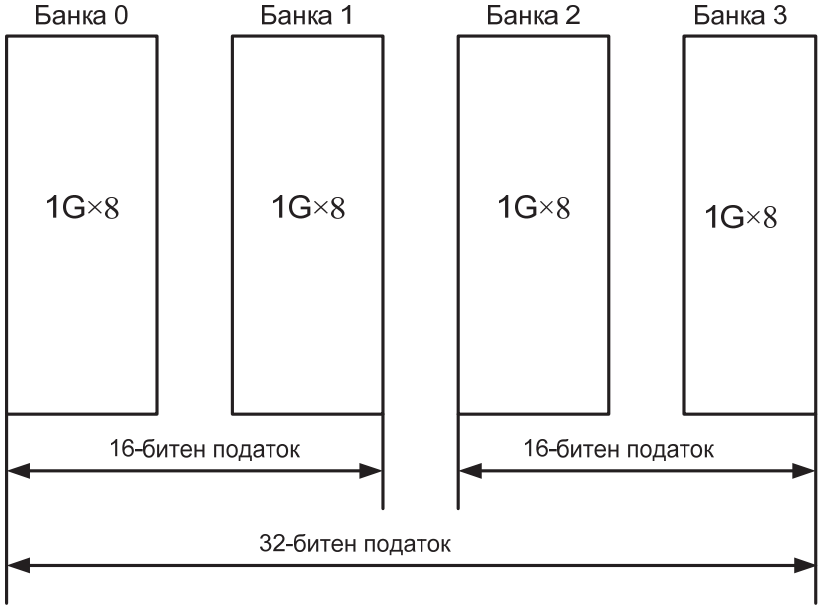


8.20. Архитектура на микропроцесор 80386

Регистрите во микропроцесорот 80386 се поделени во неколку групи: општи регистри, инструкциски покажувач, статус регистар, сегментни регистри, сегментни дискриптор регистри, контролни регистри, систем адресни регистри и регистри за тестирање и отстранување грешки (дебагирње-на англиски зборот debug значи отстранување софтверски грешки). Општите регистри, инструкцискиот покажувач, статус регистарот и сегментните регистри имаат иста функција како кај микропроцесорот 8086. **Општите регистри** можат да бидат 8,16, и 32-битни. 8-битните регистри имаат наставка L(low) и H (high). 16-битните регистри имаат наставка X: AX, BX, CX, DX. 32-битните регистри се означуваат со додавање на буквата E (extended) пред 16-битните регистри: EAX, EBX ,ECX, EDX. Инструкцискиот и статус регистарот се исто така 32 битни, а сегментните регистри се 16-битни. **Дискрипторите на сегментите** се регистри кои даваат информација за почетната физичка адреса на сегментите, големината и други нивни карактеристики. Тие се непрограмибилни, невидливи

за програмерот, но многу важни за да се разбере адресирањето кај микропроцесорот 80386. Постојат **три контролни регистри** CR0, CR2 и CR3. CR1 постои, но не се употребува и тој е резервиран за идните Интелови микропроцесори. Шеснаесете понезначајни битови од регистарот CR0 го содржат таканаречениот машински статус збор, кој содржи информации за видот на машински циклус. Контролниот регистар CR2 ја содржи 32-битната линеарна адреса на последно повиканата страница, а регистарот CR3 ја содржи физичката основна адреса на директориумот на станици. Системските адресни регистри служат за пристап до четирите специјални табели за подршка на сегментите: глобална табела на дискриптори, прекинувачкаа табела на дискриптори, локална табела на дискриптори и сегмент за сосотојбата на моментално извршуваната задача. Со функцијата на овие табели ќе се запознаеме подоцна. Да напоменеме дека истите се користат кај адресирањето во Пентиум процесорите.

Меморијата на микропроцесорот 80386 е поделена во **четири мемориски банки** и секоја банка има капацитет од 1GB. Возможен е директен пристап до 8, 16 и 32-битни податоци. За читање на 32-битен податок на микропроцесорот 80386, потребен му е еден машински циклус.



Слика 8.21. Организација на RAM меморија за микропроцесор 80386

Да се потсетиме дека на микропроцесорот 8086 му беа потребни четири машински циклуси. Секоја локација од меморијата има своја 32-битна адреса, од 00000000H до FFFFFFFFH. На сликата 8.21. се прикажани четирите мемориски банки. За избор на банка се користат контролните сигнали $\overline{BE3}$-$\overline{BE0}$ (Bank Enable). За пристап до еден бајт треба да биде селектирана една банка, а за пристап до 16-битен податок треба да се селектирани две банки. Најчесто зборовите (16-битен податок) се сместени во две соседни банки, банка 0 и 1

или банка 2 и 3. Локацијата со адреса 00000000H е сместена во банката 0, локацијата со адреса 00000001H во банка 1, 00000002H во банка 2 итн.

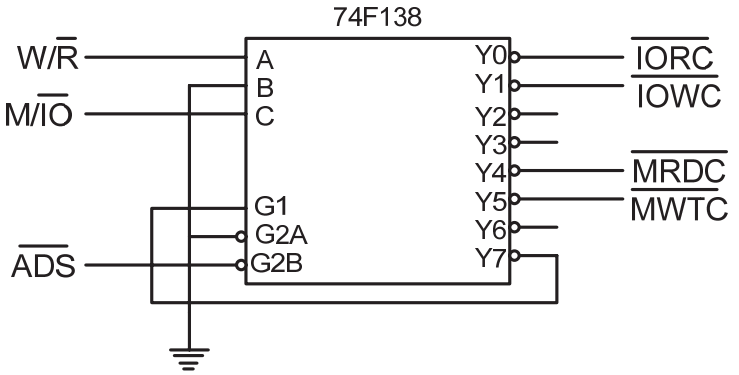
За да микропроцесорот 80386 се синхронизира со меморијата, треба да се воведат тактови на чекање во тајмингот или да се применат нови техники што ќе го забрзаат преносот на податоци. Такви техники се: pipeline, кеш-мемориите и меморија со внатрешна дозвола (interleaved memory).

Кеш-меморијата служи за чување на податоците што ги користи микропроцесорот почесто. Кеш-меморијата е SRAM меморија со време на пристап од 25ns и таа посредува меѓу DRAM меморијата и микропроцесорот. Кај микропроцесорт 80386 кеш-меморијата не е вградена меморија во чипот на микропроцесорот и таа е позната под името кеш-меморија од второ ниво (level two cache). Подоцна ќе видиме дека постојат кеш-мемории од прво ниво (level one cache) кои се интегрирани во самиот микропроцесорски чип.

Меморијата со внатрешна дозвола (interleaved memory) е уште еден метод за подобрување на брзината на микропроцесорските системи. Но, овој метод чини повеќе, бидејќи бара две адресни магистрали и посебен контролер што ќе управува со нив. Кај меморијата со внатрешна дозвола самата меморија е поделена на два посебни дела. Првиот дел ќе ги содржи локациите со адреси 00000000H-00000001H, 00000004H - 00000005H, итн, а вториот дел локациите со адреси 00000002H-00000003H, 00000006H-00000007H итн. Ова овозможува да се пристапи до 16-битниот податок со адреса 00000002H- 00000003H додека микропроцесорот го обработуба 16-битниот податок со адреса 00000000H-00000001H. На ваков начин значително се подобруваат перформансите на меморискиот систем.

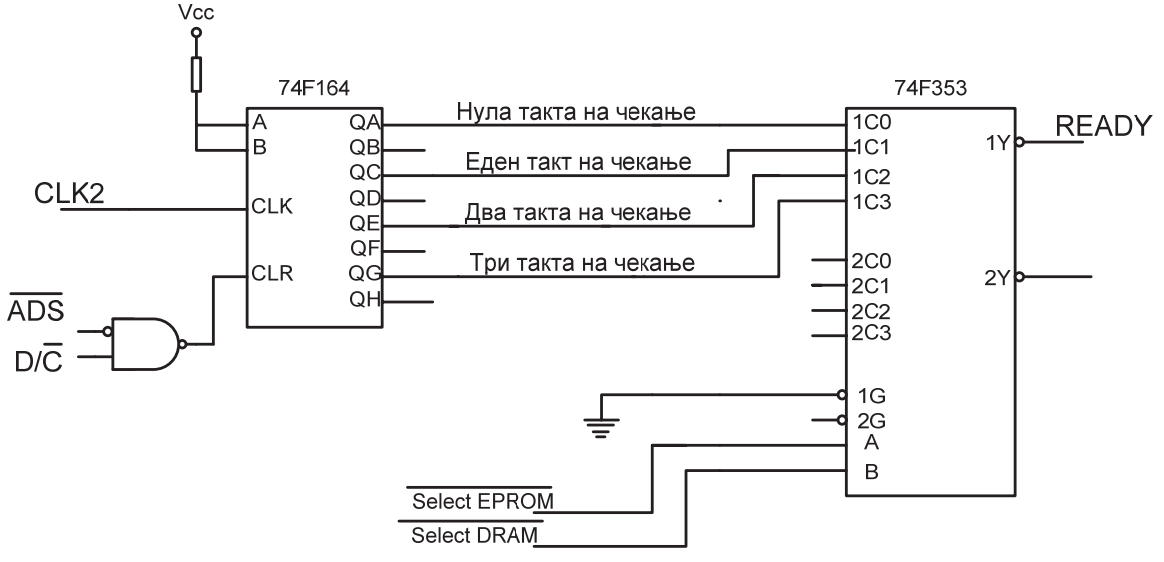
Што се однесува до **влезно излезниот** систем тој е ист со влезно излезниот систем на микропроцесорот 8086. Микропроцесорот 80386 располага со 64КВ мемориски простор наменет за адресирање на влезно излезни единици. Влезно излезниот систем може да биде мемориски изолиран и за пренос на податоци се користат инструкциите IN и OUT. За адресирање на влезно излезните единици се користат адресните линии од A15 до A 2 и контролните сигнали BE3-BE0 за избор на еден бајт, 16 битови или 32 битови од податокот.

На слика 8.22. прикажано е едноставно **интегрирано коло за генерирање на контролните сигнали** за меморијата и периферните уреди. Влезните сигнали од ова коло ги испраќа микропроцесорот, а излезните сигнали се пренесуваат до меморијата и периферните уреди. Обележувањето на контролните сигнали е исто како и кај контролните сигнали на контролерот на магистрали кај микропроцесорот 80286.



Слика 8.22. Коло за генерирање на контролни сигнали за меморија и влезно излезни единици за 80386, 80486 и Пентиум микропроцесори

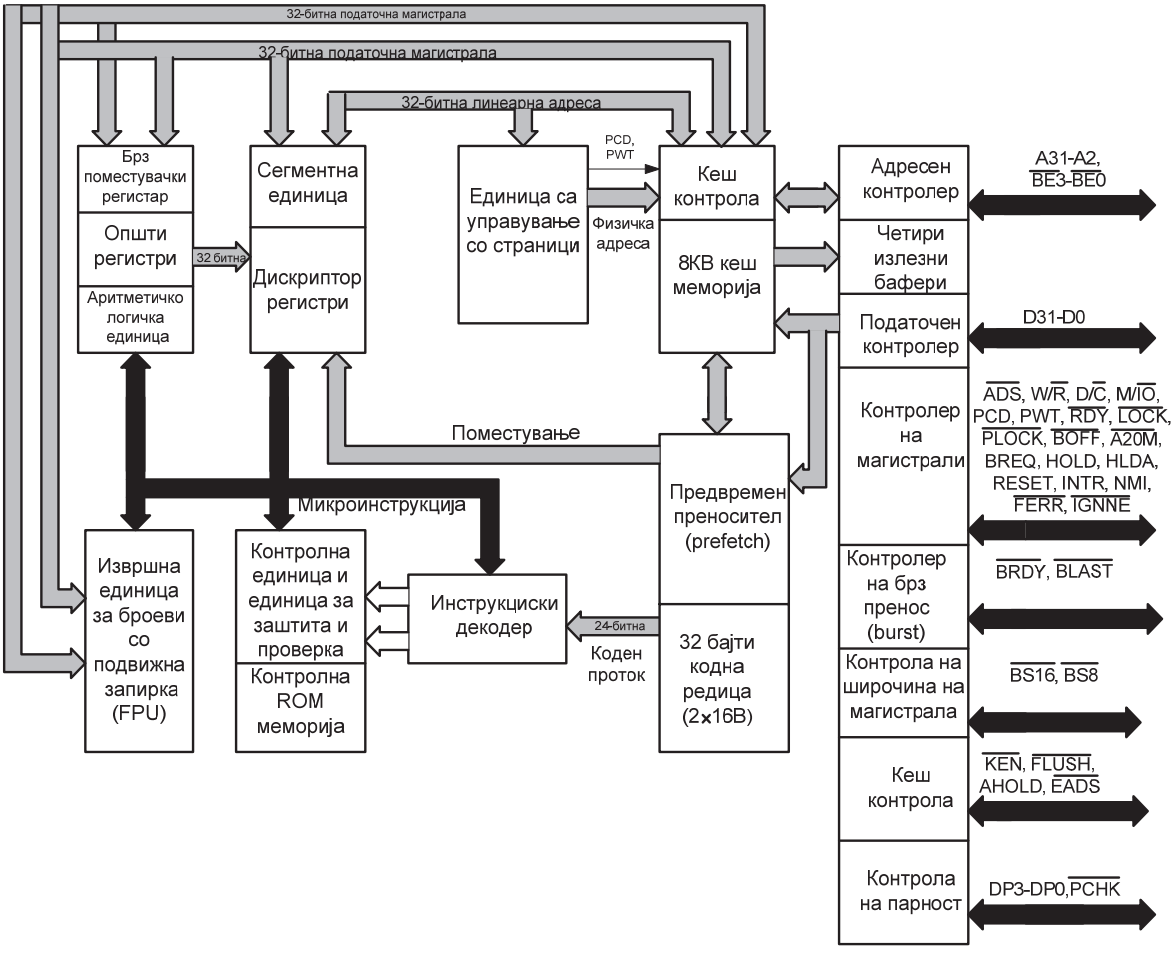
Состојбите на чекање се потребни поради големото време на пристап на мемориите. Кај микропроцесорот 80386 при работна фреквенција од 33MHz времето на пристап треба да изнесува 46ns. Бидејќи DRAM има време на пристап од 60ns тоа значи дека треба да се воведе една состојба на чекање. EPROM меморијата има време на пристап од 100ns што значи воведување на две состојби на чекање. Сигналот READY го контролира бројот на состојби на чекање во тајмингот на микропроцесорот. На слика 8.23. е прикажано **интегрираното коло за генерирање на сигналот READY**. Колото 74164 е поместувачки регистар. Ова коло се ресетира секогаш кога сигналот ADS е на ниско ниво, а сигналот D/C е на високо ниво. Поместувањето започнува кога сигналот ADS ќе стане логичка единица, после што почнуваат да се јавуваат логички единици почнувајќи од излезот QA до QH. Четирите излези од поместувачкиот регисар се поврзани со инвертирачкиот мултиплексер. Активирањето на сигналот READY зависи од сигналите за избор на еден од мемориските чипови (DRAM или EPROM).



Слика 8.23. Интегрирано коло за генерирање на сигналот READY

8.11. Основни карактеристики на микропроцесор 80486

Процесорот 80486 е коло со висок степен на интегрираност и тој е направен од 1,2 милиони транзистори. Брзината на работа на микропроцесорот е различна и изнесува: 25MHz, 33MHz, 50MHz, 66MHz или 100MHz. Постојат две варијанти на овој микропроцесор 80486DX и 80486SX. Разликата меѓу нив е што микропроцесорот 80486SX во својот состав не содржи нумерички копроцесор. На слика 8.24. прикажана е архитектурата на микропроцесорот 80486. Тој во својот состав содржи: единицата за мемориски менаџмент, 8KB кеш меморија од прво ниво, аритметичко логичката единица, **нумеричкиот копроцесор** и контролни единици.



Слика 8.24. Архитектура на микропроцесор 80486

Единицата за мемориски менаџмент MMU (Memory Menagement Unit) е составена од два дела: сегментна единица и единица за работа со страници. Во составот на сегментната единица се наоѓаат сегментните дискриптори кои ја чуваат информацијата за големината и адресата на програмските сегменти.

Кеш меморијата ги ,,храни" со информации сегментната единица и инструкцискиот декодер. Пред инструкцискиот декодер се наоѓа инструкциската редица од 32B која ги чува недекодираните операциски кодови.

Постојат две единици за ефективно извршување, аритметичко логичката единица и нумеричкиот копроцесор. Аритметичко логичката единица служи за работа со цели броеви, а нумеричкиот копроцесор со броеви со подвижна запирка (реални броеви). Кај микропроцесорот 80386 нумеричкиот копроцесор не се наоѓа во внатрешноста на микропроцесорот туку на матичната плоча.

Од десната страна на слика се прикажани контролните единици. Нивната функција ќе ја објасниме преку нивните **контролни сигнали**:

- Адресниот контролор управува со адресните линии, од A31 до A2 (за избор на локација од четирите мемориски банки) и сигналите $\overline{BE3}$-$\overline{BE0}$ за избор на еден, 2 или 4 бајти од 32-битниот податок. Излезните бафери ги испраќа податоците кон меморијата и периферните уреди преку 32-та податочни пина D31-D0.
- Најмногубројни се контролните сигнали за управување со магистралите. Кај микропроцесорот 80486 пиновите $\overline{ADS}$, W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, $\overline{LOCK}$, HOLD, HLDA, RESET, INTR, NMI имаат исто значење како кај микропроцесорите 8086, 80286 и 80486. Сигналот $\overline{A20M}$ (Address bit 20 Mask) предизвикува делење на меморијата од локација со адреса 000FFFFFH до 00000000H и се користи кога микропроцесорот 80486 треба да работи во реален мод со 1MB меморија, исто како и микропроцесорот 8086. Пиновите PCD (Page Cache Disable) и PWT (Page Write Through) се користат при пренос на страници. Пинот $\overline{RDY}$ има исто значење како и пинот READY односно го контролира бројот на состојби на чекање. $\overline{PLOCK}$(Pseudo Lock) е излезен пин и се активира кога нумеричкиот копроцесор бара 64 или 80 битен податок. $\overline{BOFF}$ (back off) е влезен сигнал со кој магистралите на микропроцесорот се стават во состојба на висока импенданса. Сигналот $\overline{BREQ}$ (Bus Request) се активира кога микропроцесорот 80486 ги користи само своите внатрешни магистрали. $\overline{FERR}$ (Floating Point Error) се активира кога нумеричкиот копроцесор ќе открие грешка. Ако откриената грешка се игнорира се активира пинот $\overline{IGNNE}$ (Ignore Numeric Error).
- Сигналите $\overline{BRDY}$ (Burst Ready) и $\overline{BLAST}$ (Burst Last Output) се користат при брз пренос на податоци по магистралата.
- За работа со кеш меморијата се користат пиновите: $\overline{KEN}$, $\overline{FLUSH}$, $\overline{AHOLD}$ и $\overline{EADS}$. $\overline{KEN}$ (Cache Enable) се активира кога податокот кој моментално се пренесува по магистралата треба да се смести во кеш

меморијата. $\overline{\text{FLUSH}}$ (Cache Flush) служи за бришење на содржината на кеш меморијата. Со сигналот $\overline{\text{AHOLD}}$ (Address Hold) адресната магистрала се става во состојба на висока импенданса, а останатите магистрали се активни. Ова се применува кога некој друг мастер на магистралите сака да има пристап до кеш меморијата. $\overline{\text{EADS}}$ (External Address Strobe) се активира истовремено со сигналот $\overline{\text{AHOLD}}$ и сигнализира надворешен пристап до кеш меморијата.

- За контрола на парност се користат четири пина DP3-DO0 (Data Parity) и доколку се открие грешка се активира пинот $\overline{\text{PCHK}}$ (Parity Check).

Заклучоци:

Микропроцесорот 8086 има меморија со капацитет од 1MB и за адресирање на мемориските локации се користат 20-битни адреси. Процесорот 8086 содржи две посебни единици. Извршната единица (execution unit) ги извршува инструкциите, а единицата за магистрален интерфејс (bus interface unit) врши пренос на операциските кодови, операндите и резултатите.

Регистри AX, BX, CX, DX се општи регистри на 8086 процесорот. Доколку сакаме да чуваме 8-битни податоци, 16-битните регистри се делат на половина. Првите помалку заначајни битови од 0 до 7 се бележат со AL, BL, CL, DL (L-low, ниско), а повеќе значајните податочните битови од 8 до 15 се бележат со AH, BH, CH, DH (H-high, високо).

Сегментазицијата претставува поделба на меморијата на повеќе делови наречени сегменти. Постојат четири вида сегменти: податочен , коден, стек и екстра сегмент. Постои разлика меѓу поимите сегмент и сегментен регистар. Сегментот е дел од меморијата, а сегментниот регистар е во микропроцесорот и помага да се пронајде почетокот на саканиот сегмент.

Во реален мод на работа за формирање на физичката адреса потребни се два дела: почетна адреса на сегментот и поместување. Поместувањето што се додава на почетната адреса на сегментот може да се содржи во некој регистар или да биде зададено како 16-битен број. Во асемблерските инструкции поместување како вредност е ставено во средна заграда [ ].

Микропроцесорот 8086 може да работи во два режими, минимален и максимален. Изборот на режим се врши преку пинот MN/$\overline{\text{MX}}$. Во минимален режим микропроцесорот сам ги генерира контролните сигнали. Во максимален

режим управувачката функција ја врши контролерот на магистрали со сериски број 8288.

Почнувајќи од 8086 до 80286 во употреба се следниве начини на адресирање: адресирање на регистри, непосредно, директно, регистарско индиректно, базно индиректно, регистарско релативно и базно релативно индексно. Кај дирекното адресирање поместувањето претставува 16-битен број, а кај регистарското индирекно адресирање поместувањето се содржи во некој општ или индекс регистар.

Со инструкцијата MOV BX, 1234H се пренесува податокот 1234H во регистарот BX. Во инструкцијата MOV BX, [1234H] извор на податок е мемориската локација оддалечена од почетокот на податочниот сегментот за растојание 1234H, а дестинација за податокот е регистарот BX.

Низата претставува множество од податоци сместени во последователни мемориски локации од податочниот или екстра сегментот. Во податочниот сегмент се сместени податоците што треба да се обработат, а во екстра сегментот податоците добиени како резултати од извршените операции. Инструкциите за работа со низи завршуваат на буквата S (string).

Инструкцијата CMP (Compare) служи за споредба на содржината на два регистри и влијае врз состојбата на знаменцата, при што вредноста на операндите останува иста. Најчесто по инструкцијата CMP следува инструкцијата за условен скок, како што се инструкциите JA (jump above) и JB (jump below).

Инструкцијата TEST служи за тестирање на битови од некој регистар. На пример, со инструкцијата TEST AL, 01H се тестира битот со најмала тежина (првиот од десна страна).

Инструкциите JG, JL, JGE, JLE, JE и JNE се условни скокови за броеви со предзнак, а инструкциите JA, JB, JAE, JBE, JE или JNE се користат за броеви без предзнак. Ако броевите се 8-битни, тогаш броеви со предзнак ќе бидат од –128 до +127, а без предзнак од 0 до 255.

Директивите се наредби, насоки со кои се уредува асемблерската програмата. Со директивите DB, DW и DD променливите и варијабилите добиваат симболични имиња. Со овие директиви се врши резервирање на простор во податочниот сегмент и тоа за 8 и 16 битни податоци. Директивата EQU се користи за креирање на лабели. Директивата SEGMENT го означува почетокот на сегментот, а ENDS нејзиниот крај. За означување на крајот и почетокот на потпрограмите се користат директивите PROC и ENDP.

Програмибилната компонента 8255 се користи при поврзување на микропроцесорот 8086 со седумсегментен дисплеј составен од осум полиња. За испраќање податоци до дисплејот се користи постапка на мултиплексирање. Изборот на поле се врши преку осумте пинови на портата B. Само еден пин од портата B ќе биде активен и ќе избере едно поле од дисплејот. Кои сегменти ќе светнат од избраното поле зависи од состојбата на пиновите на портата A.

Програмибилната компонента 8255 се користи при поврзување на микропроцесорот 8086 со тастатура составена од 16 копчиња распоредени во четири редици и четири колони. Портата B е излезна порта и служи за селекција на една колона. Портата A е влезна порта и служи за читање на состојбата на копчињата од редиците.

Тајмерот 8254 има четири внатрешни регистри: бројач 0, бројач 1, бројач 2 и команден регистар. Програмирањето се врши преку командниот регистар. Преку командниот регистар се избираат модот и бројачот. Модот го дефинира обликот на излезниот сигнал во зависност од излезниот.

Микропроцесорот 80286 располага со физичка меморија од 16MB и виртуелна меморија од 1GB. Составен е од четири функционални единици: извршна единица, адресна единица, инструкциска единица и единица за магистрален интерфејс. Микропроцесорот 80286 содржи две инструкциски редици, една пред инструкцискиот декодер и друга со декодирани инструкции.

Пиновите BUSY, ERROR, PEREQ и PEACK користат за комуникација на микропроцесорот 80286 со математичките копроцесори.

Процесорот 80386 е 32-битна верзија на микропроцесорот 8086. Податочната и адресната магистрала се 32-битни. Големината на мемориски простор изнесува 4GB. Меморијата на микропроцесорот 80386 е поделена во четири мемориски банки и секоја банка има капацитет од 1GB. Возможен е директен пристап до 8, 16 и 32-битни податоци. Четирите контролни сигнали $\overline{BE3}$-$\overline{BE0}$ се користат за избор на еден бајт од четирите бајти во избраната локација.

Регистрите во микропроцесорот 80386 се поделени во неколку групи: општи регистри, инструкциски покажувач, статус регистар, сегментни регистри, сегментни дискриптор регистри, контролни регистри, систем адресни регистри и регистри за тестирање и отстранување грешки.

Микропроцесорот 80486 содржи: единицата за мемориски менаџмент, 8KB кеш меморија од прво ниво, аритметичко логичката единица, нумеричкиот копроцесор и контролни единици. Единицата за мемориски менаџмент MMU (Memory Menagement Unit) е составена од два дела: сегментна единица и

единица за работа со страници. Аритметичко логичката единица служи за работа со цели броеви, а нумеричкиот копроцесор со броеви со подвижна запирка (реални броеви). Кај микропроцесорот 80486 нумеричкиот копроцесор се наоѓа во внатрешноста на микропроцесорот , а не на матичната плоча како што беше случај со процесорот 80386.

Прашања и задачи:

1. Која е основната разлика меѓу микропроцесорот 8088 и микропроцесорот 8086?

2. Процесорот 8086 може да работи во минимален и во максимален режим на работа. Објасни!

3. Која е функцијата на извршната единица и единицата за магистрален интерфејс на микропроцесорот 8086?

4. Кои инструкции предизвикуваат ресетирање на инструкциската редица во единицата за магистрален интерфејс?

5. Кој пин од микропроцесорот 8086 се користи за избор на минимален или максимален режим на работа?

6. Каква функција има пинот DT/$\overline{R}$ на микропроцесорот 8086?

7. Објасни го значењето на статусните битови во максимален и минимален режим на работа на микропроцесорот 8086?

8. Објасни како се бележат осумбитните и шеснаесетбитните општи регистри во микропроцесорот 8086!

9. Каква функција имаат индекс-регистрите во микропроцесорот 8086? Наброј ги индекс-регистрите!

10. Што се тоа сегменти? За што служат сегментните регистри?

11. Објасни ја постапката за пресметка на физичката адреса на мемориска локација во реален мод на работа на микропроцесорот 8086!

12. Пресметај ги адресите на мемориските локации на следниве комбинации сегментен регистар – поместување

    а) DS = 3400H BX = 2000H
    б) CS = 4900H IP = 1002H
    в) SS = 6500H SP = 3200H

г) ES = 7600H DI = 3A00H

13. Наброј ги адресните видови што се користат кај микропроцесор 8086!

14. Што ни покажува вредноста во средната заграда во инструкцијата MOV BX, [1234]?

15. Напиши инструкција со која податокот C8H ќе се внесе во регистарот CL?

16. Пополни ја долната табела.

Инструкција	Начин на адресирање	Големина на податок	Извор на податок	Цел на податок
MOV AL,[BP+DI]				
MOV CX,[DI]				
MOV [1000H],DL				
MOV AL,35				
MOV DL,[SP+2345]				
MOV [SI+1000H],AX				

17. Објасни за што служи знаменцето D во статус-регистарот на микропроцесорот 8086!

18. Напиши ги коментарите на следниве инструкции:
    - RET
    - MOVSW
    - IN AL,DX
    - JE ZBIR
    - POP [BX]
    - MOV CX,[SP]

19. Инструкцијата TEST BL,AL е погрешна. Објасни зошто.

20. CL=98H,DL=6AH. Пресметај ја содржината на регистрите по извршувањето на следниве инструкции:

```
AND CH,DL
SAR DL,2
```

21. AX=ABCDH,BX=2288H. Пресметај ја содржината на регистрите по извршувањето на следниве инструкции:

```
XOR  BH,79H
ROL AL,2
SUB AX,BX
```

22. Дали ќе биде извршена инструкцијата за условен скок во следниве примери. Објасни зошто.

А) MOV CL,80H
ADD CL,01H
JO POVTORI

Б) MOV CX,7589H
XOR CL,76H
NOT CL

В) MOV AL,7FH
SUB AL,8AH
JC AJDE

23. Која е разликата меѓу инструкциите CMP и TEST?

24. Објасни ја инструкцијата OUT 0002H,AX.

25. Кој регистар од микропроцесорот 8086 се користи за чување на податокот што го примил микропроцесорот од некој влезен уред?

26. Објасни како е извршено адресирањето на внатрешните регистри на интерфејс-компонентата 8255?

27. Интерфејс-компонентата 8255 се користи при поврзување на микропроцесор 8086 и седумсегментен дисплеј. Објасни каква функција имаат портите А и В?

28. Каков ќе биде визуелниот ефект добиен на седумсегментниот дисплеј ако PA=x1111001B и PB= 10111111B?

29. Контролниот регистар на интерфејс-компонентата 8255 има интерфејс-адреса 011H. Испрограмирај ги портите А и В така што да светне второто поле оддесно на седумсегментниот дисплеј, со бројот 5!

30. Каква функција имаат пиновите на портите А и В при поврзување на интерфејс-компонентата 8255 и тастатура?

31. Зошто е потребно да се воведе доцнење од 1ms меѓу две последователни снимања на копчињата од тастатурата прикажана на слика 8.12.?

32. Колку бројачи содржи програмибилниот тајмер 8254? Објасни ја функцијата на трите пина од секој бројач!

33. За што служи контролниот регистар на тајмерот 8254?

34. Влезната фреквенција на тајмерот 8254 изнесува 8MHz. На која вредност треба да се испрограмира бројачот и контролниот регистар за да на излез од тајмерот 8254 се добие поворка од континуирани импулси со фреквенција 200KHz?

35. Наброј ги составните единици на микропроцесорот 80286 и објасни ја нивната функција?

36. Кои сигнали се влезни, а кои излезни за генераторот на контролни сигнали 82288?

37. Објасни ја функцијата на пиновите F/C и EFI на генераторот на дигитски такт 82884?

38. Колку изнесува широчината на адресната и податочната магистрала кај микропроцесорите 80286, 80386 и 80486?

39. Кои пинови се користат за избор на мемориска банка кај микропроцесорите 80386-Пентиум?

40. Како се бележат општите регистри во микропроцесорот 80386?

41. За што служат дискрипторите на сегменти и контролните регистри кај микропроцесорот 80386?

42. Објасни го концептот на меморија со внатрешна дозвола?

43. Направи споредба меѓу интегрираните кола за генерирање на дигитски такт и контролни сигнали за меморијата и приферните уреди кај микропроцесорите 80286 и 80386?

44. Објасни како се генерира сигналот READY кај микропроцесорот 80386?

45. Кој е прв микропроцесор кој во себе содржи нумерички копроцесор? Кои инструкции ги извршува копроцесорот?

46. Наброј ги пиновите на микропроцесорот 80486 кои се користат за контрола на кеш меморијата. Објасни ја нивната намена!

9. Пентиум микропроцесори

9.1. Пентиум 1 микропроцесор

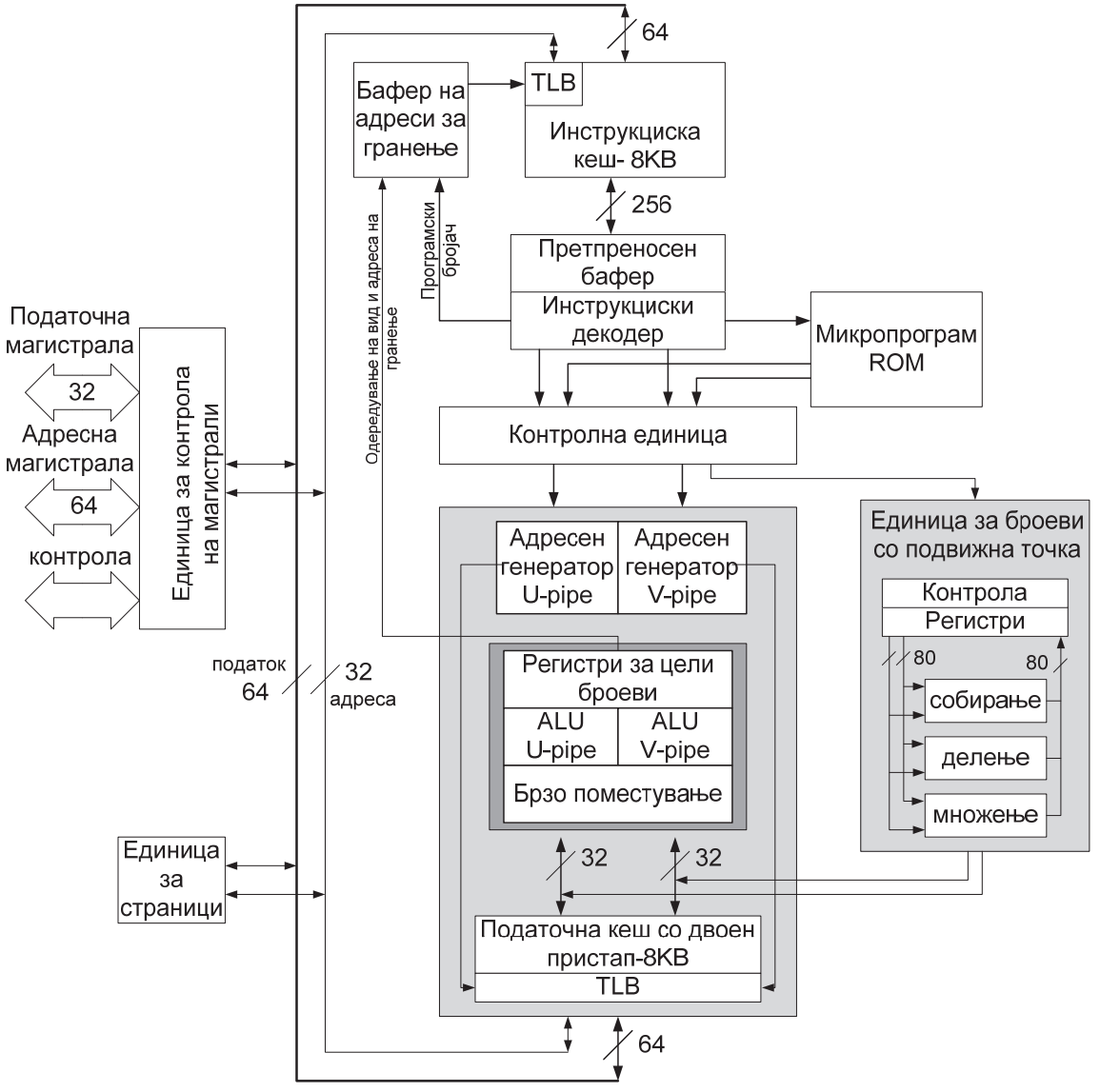
Кај Пентиум микропроцесорите има големи подобрувања во однос на микропроцесорот 80486, како што се: подобрената кеш-меморија, поширокаta 64-битна податочна магистрала, побрзиот нумерички копроцесор, дуплиот микропроцесор за цели броеви и логиката за предвидување на разгранувањата. Архитектурата на Пентиум микропроцесорот е прикажана на слика 9.1.

Кеш-меморијата од прво ниво L1 е поделена на две кеш мемории од по 8KB, од кој првата е инструкциска, а втората е податочна кеш-меморија. 32- битната податочна магистрала кај микропроцесорот 80486 е заменета со 64-битна. **Нумеричкиот копроцесор** служи за обработка на податоци со подвижна запирка и е пет пати побрз од нумеричкиот копроцесор 80486. **Дуплиот микропроцесор за цели броеви** може да извршува две инструкции во еден дигитски такт. Логиката за предвидување ја зголемува ефикасноста во извршувањето на инструкциите за гренење (branch). **Логиката за предвидување на гранењата** придонесува да се намали времето на чекање за извршните единици. Штом ќе се појави скок инструкција микропроцесорот започнува предвремен пренос на следните инструкции од адресата наведена во скок инструкцијата. Инструкциите кои треба да се извршат се чуваат во инструкциската кеш меморија, па штом ќе се дојде на ред скок инструкцијата таа ќе се изврши во еден дигитски такт. Ако условот за скок не биде исполнет тогаш за скок инструкцијата ќе бидат потребни дополнителни три дигитски такта. За среќа предвидувањата се обично точни.

Суперскаларната архитектура на Пентиум микропроцесорите произлегува од постоењето на трите извршни единици: една за броеви со подвижна запирка и две за цели броеви (U pipe, V pipe). Ова значи дека паралелно може да се извршуваат три различни инструкции.

Меморијата на Пентиум микропроцесорот е голема 4GB и таа е поделена во 8 мемориски банки по 512MB. Секоја мемориска локација може да запамти податок од еден бајт и еден бит за контрола на парност. Ако бројот на единици во

мемориранит бајт е парен битот за парност ќе биде еден , а ако бројот на единици е непарен тогаш тој бит е нула.



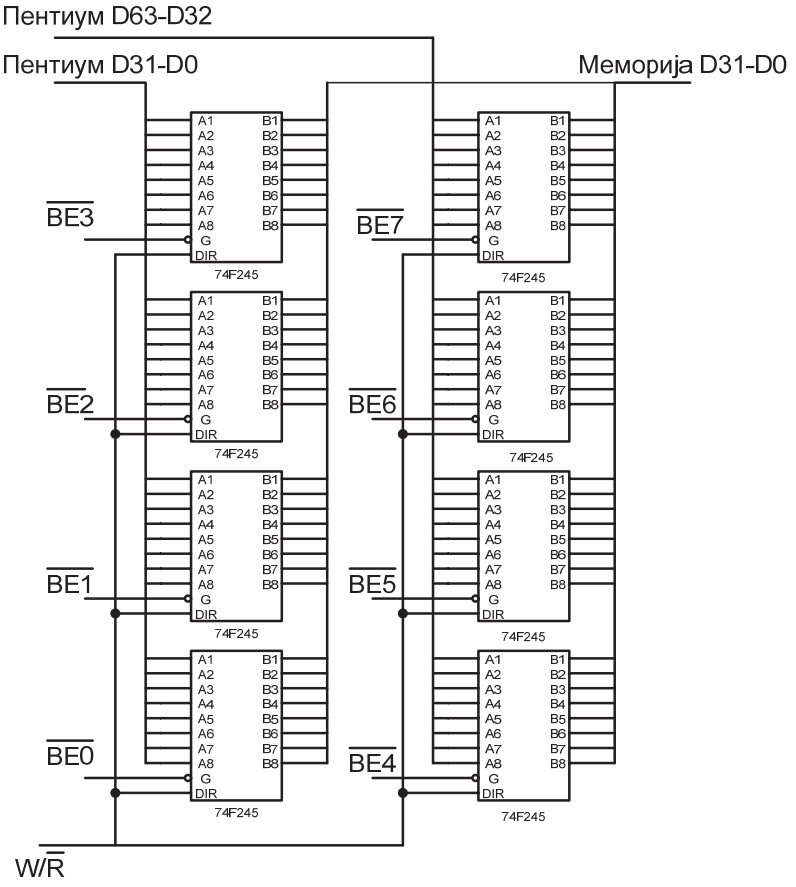
Слика 9.1. Архитектура на Пентиум 1 микропроцесор

Селекцијата на банките се врши преку контролните сигнали $\overline{BE7}$-$\overline{BE0}$ (bank enable). Посебниот пристап до секоја мемориска банка дозволува пристап до 8, 16, 32 и 64-битни податоци. На пример, кога микропроцесорот сака да прочита 64-битен податок од меморијата тогаш треба да бидат селектирани сите осум банки односно сите осум пинови $\overline{BE7}$-$\overline{BE0}$ треба да бидат активни. Ако треба да се прочита само првиот бајт од 64-битниот податок тогаш ќе биде активен само пинот BE0. 64-битната мемориска широчина и 64-битната податочна магистрала се многу важни доколку се обработуваат броеви со подвижна запирка. Бидејќи броевите со подвижна запирка се големи 64 битови, за читање на еден ваков број ќе биде

потребен еден циклус , за разлика од микропроцесорот 80486 каде беа потребни два циклуси.

Покрај контролата на парност на податоците, кај микропроцесорот Пентиум 1 контрола на парност се врши и на адресните битови (A31-A0) и тоа за секоја операција. За оваа цел се користи пинот AP (address parity) и доколку се појави грешка тогаш се активира пинот APCHK (address parity check). Во случај на грешка системот превзема соодветно дејство односно генерира прекин.

Податочната магистрала на меморијата е 32-битна, а на Пентиум микропроцесорот 64-битна. За **поврзување на** овие две **магистрали** се користат мултиплексери кои вршат претварање на 64-битните податоци во 32-битни и обратно. На слика 9.2. е прикажан еден ваков систем составен од осум бидирекционални бафери.



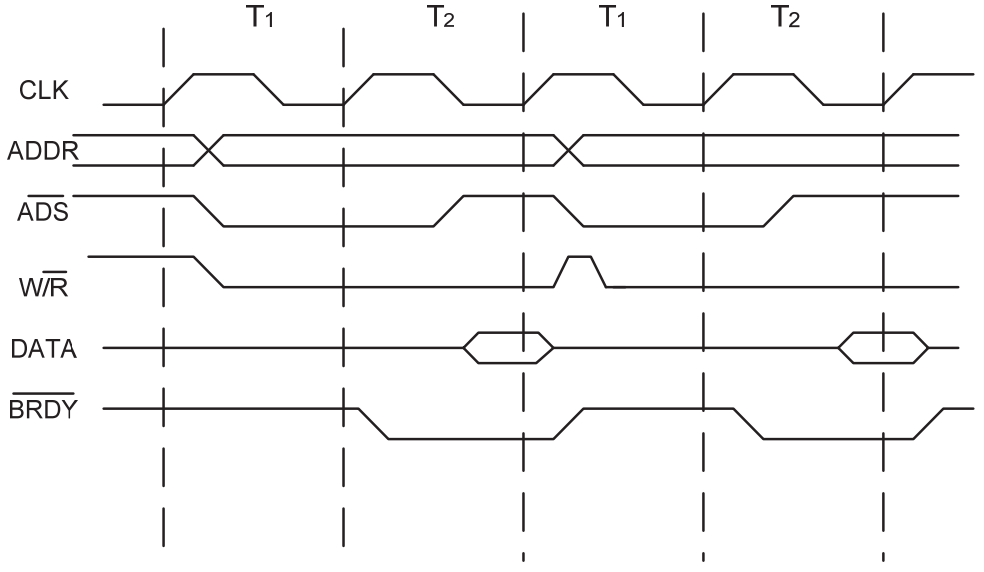
Слика 9.2. Поврзување на 64-битната податочна магистрала на микропроцесорот Пентиум со 32- битната податочна магистрала на меморијата

Насоката на пренос ја одреува сигналот W/$\overline{R}$ кој се носи до пиновите DIR (direction) на сите осум бафери. Кога W/$\overline{R}$ =0 тогаш се чита и податоците се движат од меморијата кон микропроцесорот. Кога W/$\overline{R}$=1 тогаш се пишува и податоците се движат од микропроцесорот кон меморијата. Сигналите за избор на мемориските

банки од $\overline{BE0}$ до $\overline{BE7}$ се користат за оспособување на баферите односно тие се носат до пиновите за селекција на баферите G (Gate). Кога треба да се пренесат четирите позначајни бајти од 64- битниот податок, D63-D32 тогаш се активни сигналите од $\overline{BE4}$ до $\overline{BE7}$, а кога треба да се пренесат четирите понезначајни бајти, D31-D0 тогаш се активни линиите од $\overline{BE0}$ до $\overline{BE3}$. Како се активираат сигналите Bank Enable тека податоците се запишуваат во соодветните мемориски банки

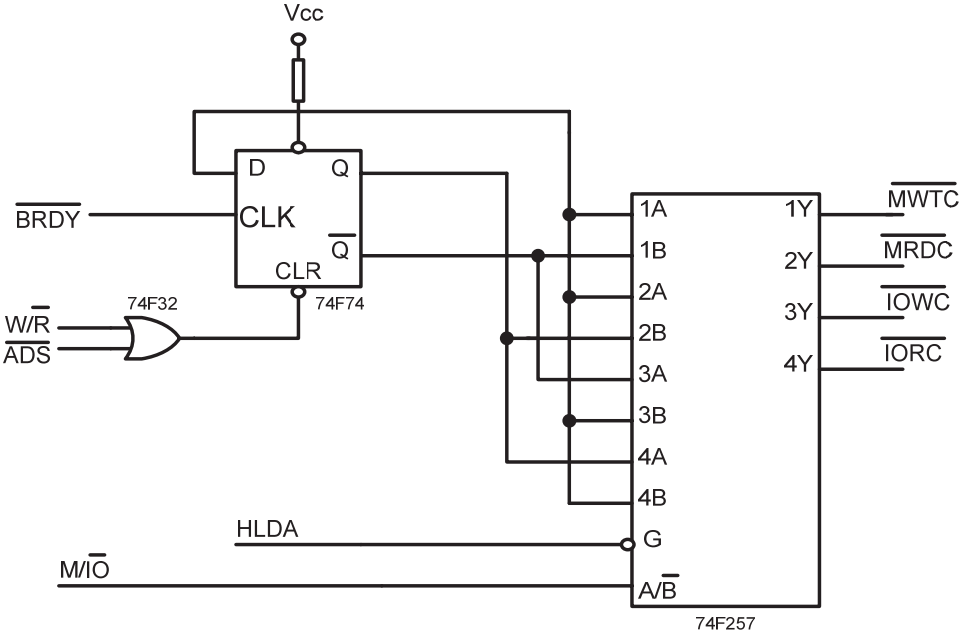
Многу поефективен начин за читање од меморија е преку циклусот **за проток (burst).** Англискиот збор burst во превод значи брз проток, силно струење, наплив, надоаѓање. Со еден циклус за проток може да се пренесат четири 64-битни броеви за времетраење од пет дигитски такта. Без такта за чекање со овој циклус преносот од меморија трае 15,2ns. Ова е посебно згодно кога се чита од кеш меморија од второ ниво.

Машинскиот циклус кај Пентиум микропроцесорите траат два дигитски такта. Работната фреквенција изнесува 66MHz што значи 33 милиони мемориски преноси по секунда. На сликата 9.3. се прикажани два машински циклуси, првиот за читање од меморија (W/$\overline{R}$=0), а вториот за пишување (W/$\overline{R}$ =1) . Пинот $\overline{ADS}$ (Address Data Strobe) се акривира секогаш кога микропроцесорот испраќа адреса. Без овој сигнал адресата нема да биде валидна. Сигналот $\overline{BRDY}$ (Burst Ready) е влезен сигнал за микропроцесорот. На крајот од првиот дигитски такт од двата машински циклуси сигналот $\overline{BRDY}$ е еднаков на еден и ваквата негова состојба е индикатор дека меморијата се уште не е спремна за пренос на податоци, со што автоматски се вметнува цел дигитски такт чекање. За да нема чекање потребно е сигналот $\overline{BRDY}$ да биде на ниско ниво и ова се случува на крајот од вториот дигитски такт од мешинските циклуси. После завршувањето на вториот такт тековниот циклус завршува и може да почне нов.



Слика 9.3. Временски дијаграм на машински циклус за читање и пишување кај Пентиум 1 микропроцесор

Контролните сигнали W/$\overline{R}$, $\overline{ADS}$ и M/$\overline{IO}$ служат за **генерирање на контролните сигнали** за меморијата и периферните уреди. За таа цел се користи тактирачки флип флоп и мултиплексер со осум влезови и четири излези. Ова е прикажано на слика 9.4.



Слика 9.4. Генерирање контролни сигнали за меморија и периферни уреди

Пинот $\overline{BRDY}$ (Burst Ready) се активира кога меморијата или периферниот уред презема или дава податоци на податочната магистрала. За да заврши циклусот на читање, на крајот од вториот дигитски такт сигналот $\overline{BRDY}$ треба да биде еднаков на логичка нула. Во спротивно ќе биде потребно да се воведе такт на чекање. Диколку сигналот $\overline{BRDY}$ не е еднаков на нула може да се воведат најмногу четири такта на чекање.

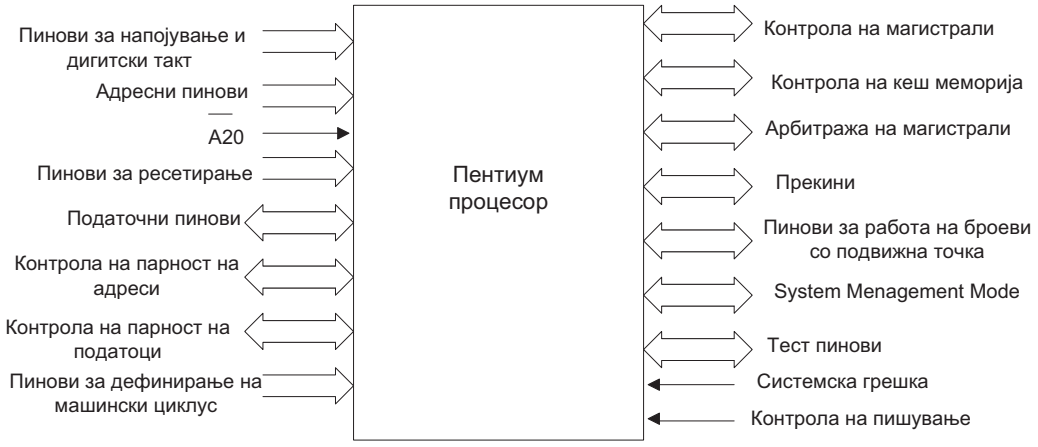
9.2. Пин дијаграм на Пентиум 1 микропроцесор

Пентиум 1 микропроцесорот има **237** пина. На сликата 9.5. е прикажан пин дијаграм во кој пиновите се поделени во групи во зависност од нивната функционалност. Микропроцесорот Пентиум 1 користи напојување од 5V со јачина на електрична струја од 3,3A. Фреквенцијата на работниот такт изнесува 66MHz.

Постојат **29 адресни** пинови A31-A3 за избор на мемориска локација и осум пинови за избор на мемориска банка $\overline{BE7}$-$\overline{BE0}$.

Пинот $\overline{A20}$ се активира кога микропроцесорот треба да работи во реален мод на работа. Ова е потребно за да се постигне компатибилност со постарите генерации микропроцесори.

Постојат два **пина за ресетирање**. Кога ќе се активира пинот RESET сите програми се прекинуваат и микропроцесорот започнува да извршува софтвер од локација со адреса FFFFFFF0H. Влезниот пин INIT (initialization) исто означува ресетирање со исклучок на кеш меморииите, баферите за повратно пишување и регистрите за броеви со подвижни точки.



Слика 9.5. Пин дијаграм на Пентиум микропроцесор

За **контрола на парност на адресите** се користат два пина AP и $\overline{APCHK}$. Преку пинот AP(address parity) се испраќа битот за парност . Ако во испратената адреса има парен број на единици тогаш овој бит е еден , а во спротивно е нула. Доколку се појави грешка во парноста тогаш се активира излезниот пин $\overline{APCHK}$ (address parity check).

Контрола на парност се врши и **на податочните битови** прочитани од меморијата или некој периферен уред . За секоја мемориска банка постои по еден пин за контрола на парност DP7-DP0 (data parity). Доколку се појави грешка во парноста се активира излезниот пин $\overline{PCHK}$ (parity check). Преку пинот $\overline{PEN}$ (parity enable) микропроцесорот ја активира прекинувачката потпрограма за отстранување на настанатата грешка.

За дефинирање на видот на машински циклус се користат пиновите M/$\overline{IO}$ (memory/$\overline{input\ output}$) , D/$\overline{C}$ (data/$\overline{control}$) , W/$\overline{R}$ (write/$\overline{read}$) , $\overline{CASHE}$, $\overline{LOCK}$. Со пинот M/$\overline{IO}$ микропроцесорот бира вид на уред за комуникација. Со пинот W/$\overline{R}$ бира операција. Кога пинот D/$\overline{C}$ е еднаков на еден тогаш се пренесуваат податоци од меморијата или периферниот уред, а кога тој е нула се работи за машински цуклус обработка на прекин . Кога е активен пинот $\overline{CASHE}$ тогаш микропроцесорот чита податоци од кеш меморијата. Пинот $\overline{LOCK}$ најчесто се користи при директен пренос на податоци од меморија (DMA концептот).

Пинови за контрола на магистралите се $\overline{NA}$ (next address) , $\overline{ADS}$ (address data strobe) , $BRDY$ (burst ready). Пинот NA се активира кога надворешната меморија е спремна за следниот машински циклус. $\overline{ADS}$ служи за оспособување на адресната

магистрала и овој пин е секогаш активен кога микропроцесорот испраќа адреса до меморијата или периферниот уред. Пинот $\overline{BRDY}$ служи за внесување на такта на чекање доколку на меморијата и треба повеќе време да пристапи до наведената локација.

Постојат специјални **пинови за контрола на кеш меморијата**. Пинот $\overline{KEN}$ (cache enable) ја оспособува внатрешната кеш меморија. Пинот WB/$\overline{WT}$ (write-back/write-through) бира операција за кеш меморијата. Кога овој пин е еден тогаш кеш меморијата прима податоци од микропроцесорот, акога овој пин е нула тогаш прима податоци од меморијата. Преку активирање на пинот $\overline{FLUSH}$ се поништува внатрешната кеш меморија.

Под **арбитража на магистрали** подразбираме бирање на уред кој ќе ги користи магистралите за пренос на податоци . Пинот BREQ (bus request) се активира кога микропроцесорот испраќа барање за контрола на магистралите. Магистрралите може да ги користи и некој брз периферен уред за директен пренос на податоци (DMA) и тогаш се активираат пиновите HOLD и HLDA.

Нумеричкиот копроцесор обработува реални броеви познати уште како броеви со подвижна точка. Пинот $\overline{FERR}$ (floating-point error) се активира кога ќе се појави грешка во работата на нумеричкиот копроцесор. Но ако е активен пинот I$\overline{GNNE}$ (ignore numeric error) тогаш овие грешки не се забележуваат.

Процесорот Пентиум 1 може да работи во два **специјални мода** . Кога е активен влезниот пин $\overline{SMI}$ (system management interrupt) тогаш микропроцесорот работи во мод на системско управување со меморијата. Ако е активен пинот TMS (test mode select) тогаш микропроцесорот ќе работи во тест мод .

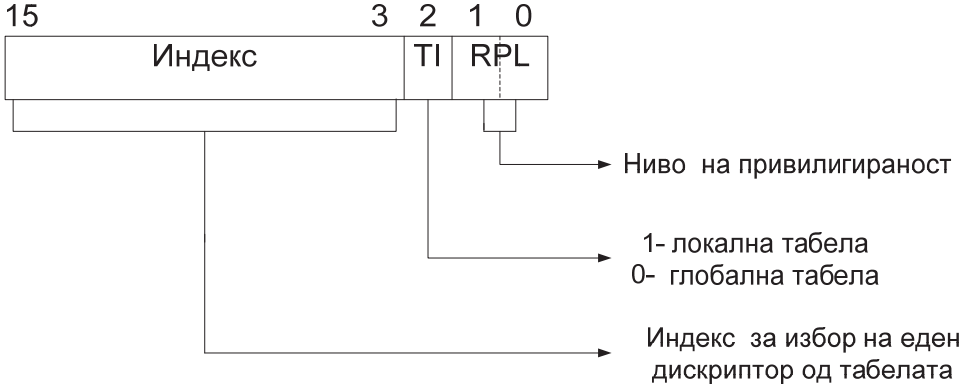
9.3. Пентиум мемориски менаџмент

Пентиум микропроцесорите може да работат во три мода: реален мод, заштитен мод и виртуелен мод на работа. Со реалниот мод на работа се запознавме кога го проучувавме микропроцесорот 8086. Тој е наједноставен и дозволува пристап на микропроцесорот до првите 1MB бајти мемориски простор. Во **заштитен мод** адресниот простор се зголемува до **4GB** (2^{32}), а со **виртуелизација** добиваме екстремно голем адресен простор од **64TB** (2^{46}).

Во заштитен мод на работа сегментниот регистар не ја содржи почетната адреса на сегментот, туку тој содржи таканаречен **селектор**. Селекторот овозможува избор на еден од 8192 дискртиптори. Секоја програма си има своја локална табела на дискриптори (опишувачи), но постои една единствена глобална табела на дискриптори што ја делат сите програми во сметачот. Локалната табела ги опишува сегментите што ги користи една програма (коден, податочен, стек-сегмент), додека

глобалната табела ги опишува системските сегменти, вклучувајќи ги оние што ги користи и самиот оперативен систем.

За да пристапиме до еден сегмент, потребно е неговиот селектор да се смести во еден од сегментните регистри. Секој селектор се состои од 16 бита. Селекторот е така формиран за да обезбеди лесно пронаоѓање на дискрипторот. На слика 9.6. е прикажано значењето на битовите од селекторот.



Слика 9.6. Функционален опис на битовите од селекторот

Прво се избира една од двете табели GTD или LTD во зависност од вредноста на битот 2 од селекторот. Со помош на индексот од селекторот се лоцира еден дискриптор, односно индексот ни покажува колку е оддалечен дискрипторот од почетокот (дното) на соодветната табела на дескриптори.

Преку **дискрипторот с**е пронаоѓа саканиот сегмент во меморијата. На слика 9.7. е прикажан еден дискриптор. Тој е составен од 8 бајти.

Слика 9.7. Функционален опис на битовите од дискриптор на сегмент

Базната адреса, всушност, претставува почетна адреса на сегментот. Лимитот служи за пронаоѓање на крајната адреса на сегментот. Големината на сегментот зависи и од битот G. Ако битот G (granularity) е нула, тогаш **лимитот** е изразен во бајти и сегментот има максимална големина 2^{20}=1MB (20-број на бита предвидени за лимитот). Ако G=1, тогаш лимитот е изразен во страници. Кај микропроцесорот 80836 страниците не се никогаш помали од 4KB=2^{12} . Ова значи дека со 20 бита во лимитот добиваме вкупна големина на сегментот од $2^{32}=2^{20}2^{12}$ = 4GB

Пример 9.1 :Базната адреса на сегментот е 1000 000H и лимитот е 001FFH. Да се пресмета адресата на последната локација, односно крајот на сегментот.

А) G=0

Б) G=1

Решение:

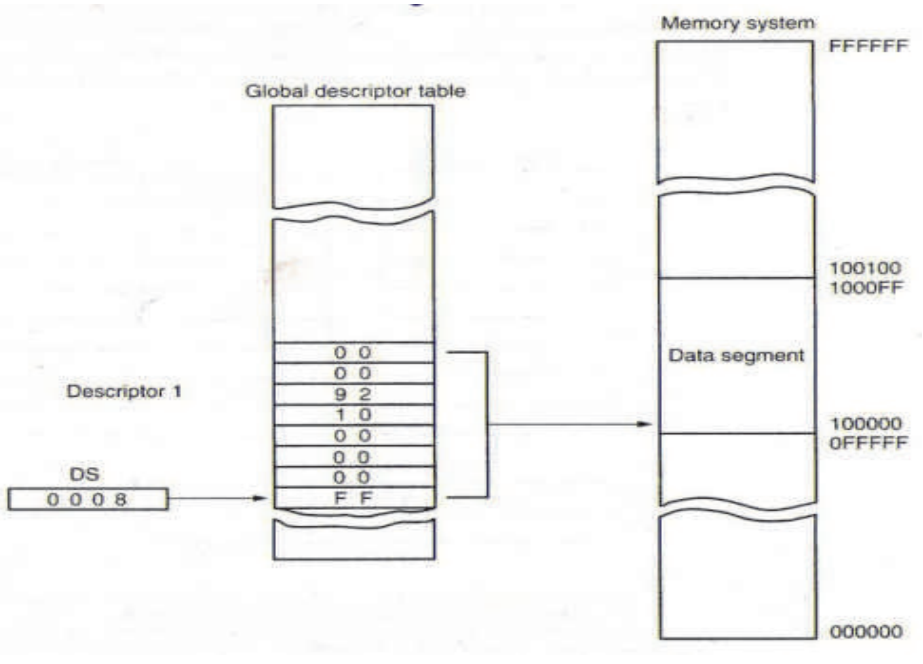
А) Ако G=0, тогаш лимитот е даден во бајти па крајот го добиваме на следниов начин:

Крајна адреса=Почетна адреса + лимит=1000 000H+001FFH=1000 0 1FFH

Б) Ако G=1, тогаш лимитот е даден во страници. Бидејќи сегментот почнува од адреса 1000 000H, тоа значи дека неговата прва страница ќе се простира од адресата 1000 000H ДО 1000 0FFFH. Втората страница ќе се простира од 1000 1000H до 1000 1FFFH, третата од 1000 2000H до 1000 2FFFH. Ова треба да се повтори 01FF пати. Последната страница ќе се простира од 101FF000H до 101FFFFFH. Бидејќи крајот на последната страница се совпаѓа со крајот на сегментот, тоа значи крајната адреса=101FFFFFH.

Бајтот за контрола на пристапот опишува како функционира сегментот во рамките на системот. На пример, ако станува збор за податочен сегмент, може да се нагласи дали податоците во него можат да бидат менувани или тие се заштитени. Исто така, доколку сегментот го надмине својот лимит, програмата ќе биде прекината (ќе се појави прекинот TRAP).

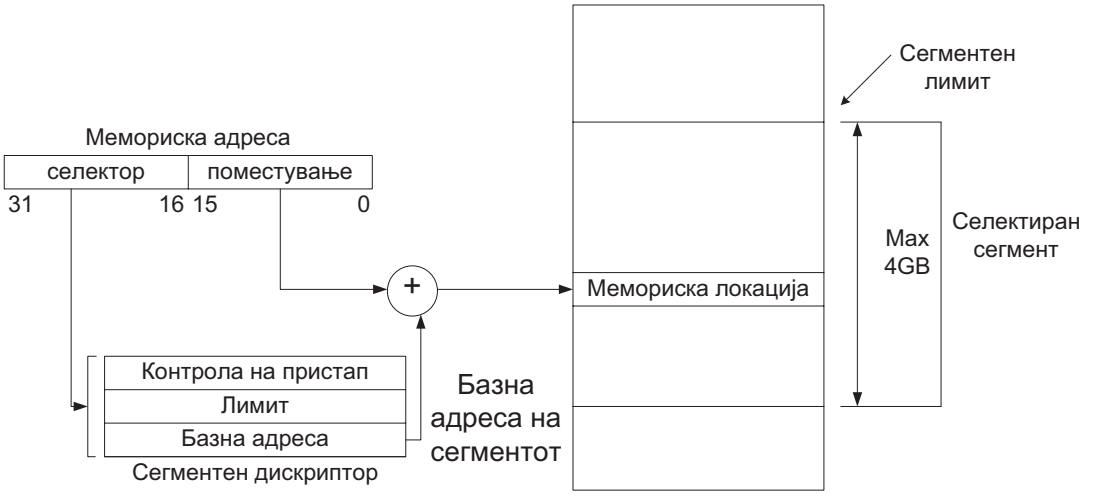
На сликата 9.8. е даден пример за **пронаоѓање на** саканиот податочен **сегмент** од меморијата кога сегментниот регистар содржи селектор чија вредност е 0008H=0000000000001000B.



Слика 9.8. Пристап на Пентиум микропроцесорот до податочен сегмент во меморијата

Според сликата 9.6. битовите од 3 до 15 покажуваат на првиот дискриптор од глобалната табела на дискриптори (битот број 2 е 0). На слика 9.7. се дадени осумте бајти од кои е составен дискрипторот и кои се нумерирани од 0 до 7. Вториот, третиот, четвртиот и седмиот бајт од дискрипторот, всушност, се базната адреса, односно ја даваат почетната адреса на сегментот. Од слика 9.8. гледаме дека овие бајти од дискрипторот заедно ја даваат вредноста 10000000.

Откако ќе се пронајде саканиот сегмент, се пристапува кон **пронаоѓање на** саканиот **податок**. 80386 користи 32-битна адреса. Позначајните шеснаесет бита од адресата ја даваат вредноста на селекторот, а понезначајните шеснаесет бита ја даваат вредноста на поместувањето. Кај микропроцесорот 80386 поместувањето има исто значење како и поместувањето кај микропроцесорот 8086. Поместувањето го дава растојанието од почетокот на сегментот до саканата мемориска локација. Ова е прикажано на сликата 9.9 .



Слика 9.9. Пристап до мемориска локација од избраниот сегмент

На крајот можеме да го донесеме следниов **заклучок**:

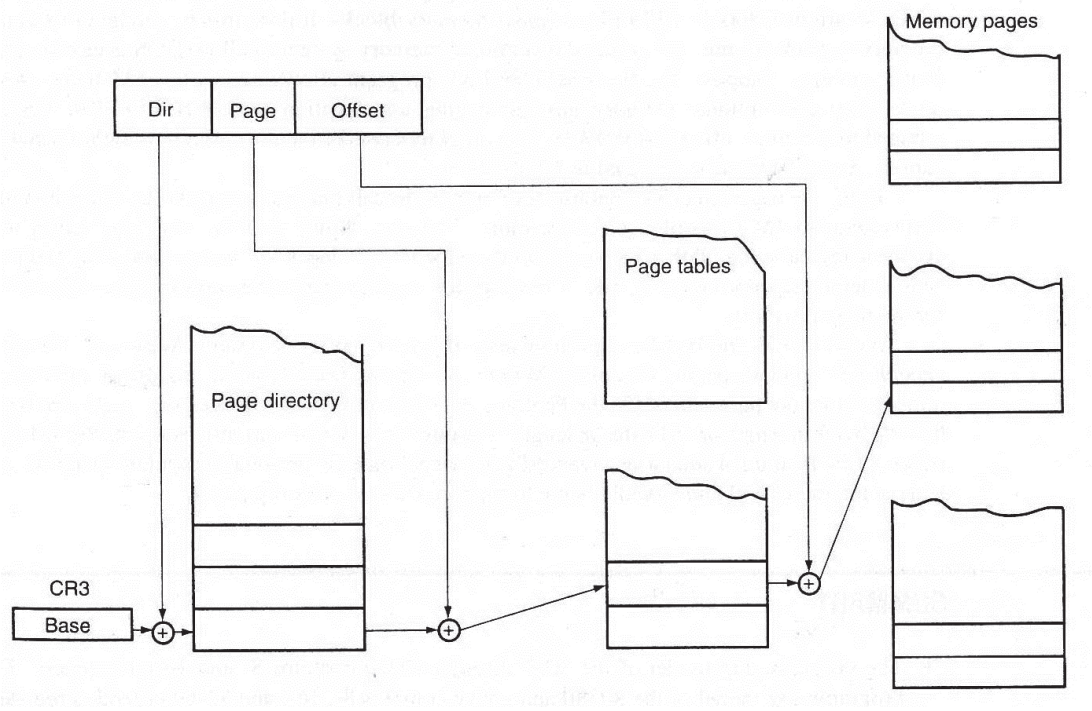
- Шеснаесетте позначајни бита од мемориската адреса ја даваат вредноста на селекторот.
- Селекторот служи за пронаоѓање на дискрипторот на сегментот во табелите на дискриптори.
- Дискрипторот служи за пресметка на почетната и на крајната адреса на сегментот.
- Поместувањето служи за пронаоѓање на мемориската локација од селектираниот сегмент.

9.3.1. Заштитно виртуален мод на работа

Доколку се користи механизмот на страници, тогаш 32-битната мемориска адреса се интерпретира како виртуелна адреса. За добивање на физичката адреса од линеарната (сега вирутуелна) адреса, се користат три компоненти:

1. Директориум на станици.
2. Табела на страници.
3. Самата страница, која е дел од корисничката меморија и е ограничена во таканаречена рамка.

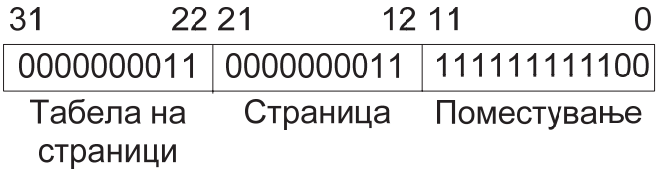
На сликата 9.10. е прикажан механизмот на добивање физичка адреса од виртуелна адреса.



Слика 9.10. Пристап до меморија со употреба на виртуелни страници

Во системот постои само еден директориум, чија почетна адреса се наоѓа во контролниот регистар CR3. **Директориумот** е составен од 1024 локации. Која од овие локации ќе биде избрана, зависи од вредноста на битовите од 22 до 31 од виртуелната адреса. Избраната локација од директориумот нè упатува до една од 1024-те **табели на страници**. Бројот на локации во директориумот е еднаков на бројот на табели на страници. Секоја табела на страници е составена од 1024 локации и која локација ќе биде избрана, зависи од вредноста на битовите од 12 до 21 од виртуелната адреса. Избраната локација од табелата на страници нè упатува до почетокот на една **виртуелна страница**. На крај, која **локација од страницата** ќе биде избрана, зависи од поместувањето, односно битовите од 0 до 11 од виртуелната адреса.

Пример 9.2:
Одреди го редниот број на табела на страница, редниот број на страница и локацијата од избраната страница, ако е позната вредноста на виртуелната адреса 00C03FFCH.
Решение:
Виртуелната адреса во бинарен броен системе прикажана на слика 9.11.



Слика 9.11. Значење на битовите од виртуелната адреса

Битовите од 31 до 22 упатуваат на табелата на страници со реден број 3. Битовите од 21 до 12 упатуваат на третата страница. Бараната локација е оддалечена од почетокот на страницата за растојание FFCH.

Честопати микропроцесорот бара податоци од соседни мемориски локации. За да се скрати времето на проверување на меморијата, микропроцесорот 80386 има хардверска поддршка за чување на последната користена комбинација директориум-табела-страница, а се менува вредноста на поместувањето (битовите од 0 до 11).

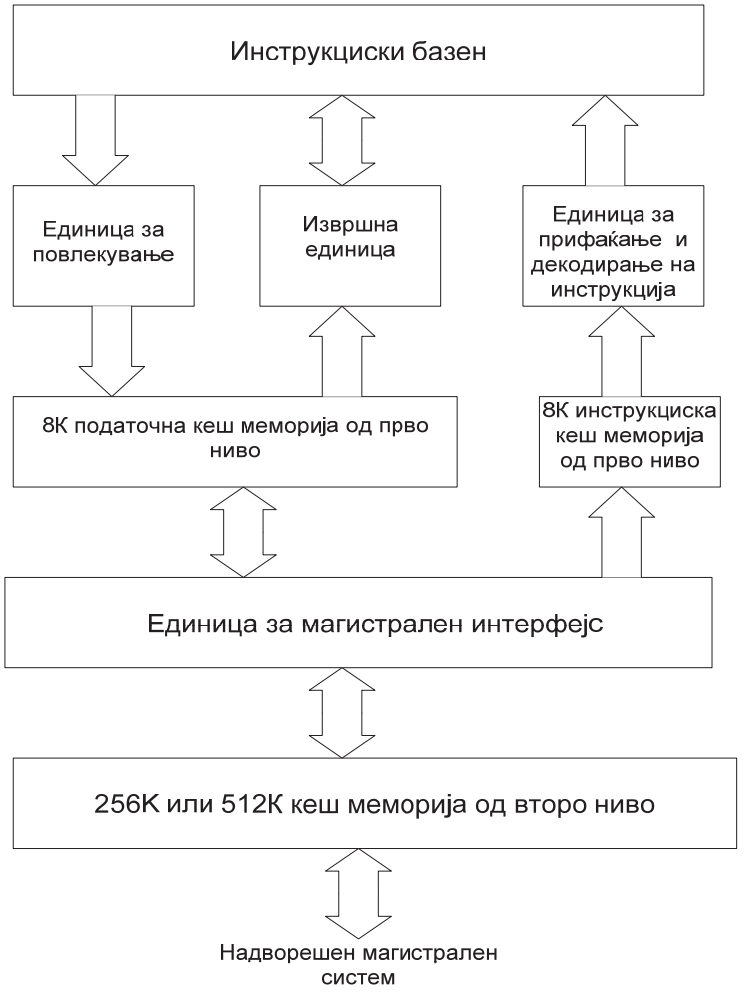
Сега ќе го пресметаме капацитетот на виртуелната меморија

- Големината на една страница изнесува 2^{12} = 4KB (12 е број на бита во поместувањето).
- Табелата на страници има капацитет од 1024 страници × 4KB = 1K× 4KB = 4MB.
- Директориумот на страници содржи 1024 табели на страници што значи неговиот капацитет изнесува 1024×4MB=1K×4MB=4GB.
- Всушност, 4GB е капацитетот на еден сегмент кога лимитот на сегментот е изразен во страници. Да се потсетиме, пронаоѓањето на саканата локација во меморијата започна со внесување на селекторот на сегментот во сегментниот регистар. Селекторот содржи 13-битен индекс за избор на дискриптор и TI бит за избор на една од двете табели на дискриптори. Ова значи дека со селекторот можат да се адресираат 2^{14} =16K сегменти. Ако бројот на сегменти се помножи со капацитетот на еден сегмент, се добива капацитет од 64 TB.

9.4. Пентум Про микропроцесор

Пентиум Про микропроцесорот по својата структура се разликува од сите претходни микропроцесори. На сликата 9.12. е прикажана внатрешната структура на Пентиум Про микропроцесорот.

Пентиум Про микропроцесорот содржи две **кеш-мемории** од различни нивоа. Кеш меморијата од второ ниво е со капацитет од 256К или 512КВ и е вградена во самата матична плоча. Кеш меморијата од прво ниво е во самиот микропроцесорски чип и има капацитет 8КВ податочна кеш-меморија и 8КВ инструкциска кеш-меморија.



Слика 9.12. Архитектура на Пентиум Про микропроцесор

Единицата за магистрален интерфејс ги генерира мемориските адреси и контролните сигнали и доставува податоци или инструкции до двете кеш-мемории од прво ниво.

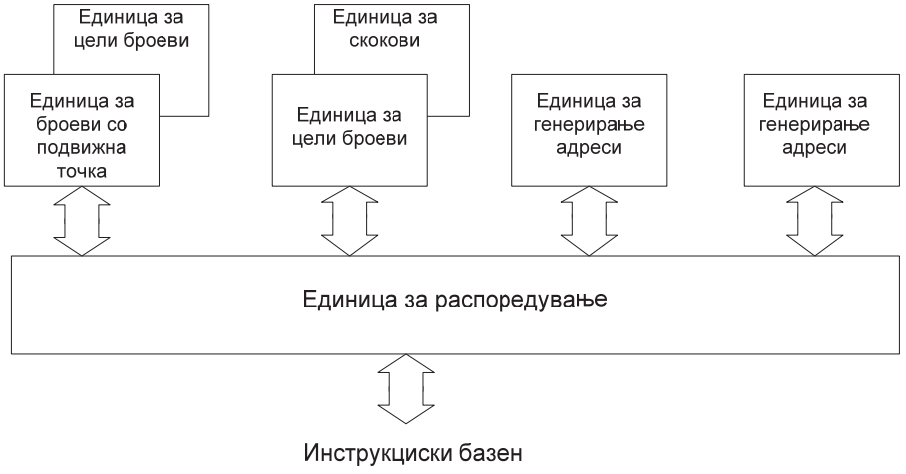
Единицата за прифаќање на инструкцијата и нејзино декодирање се состои од три декодери, кои можат истовремено да декодираат три различни

инструкции. Излезите на трите декодери се поврзани со инструкцискиот базен во кој инструкциите остануваат сè додека не ги прифати извршната единица. Во единицата за прифаќање на инструкциите и нивно декодирање постои посебна логичка секција која го менува текот на програмата доколку се лоцира инструкција за скок. Инструкцискиот базен во внатрешноста на микропроцесорот е мала меморија со адреси на локациите.

Внатрешната структура на **единицата за** прифаќање и за **извршување** на инструкциите е прикажана на слика 9.13. Се забележува дека постојат три извршни единици: две за цели броеви и една за броеви со подвижна точка.

Единицата за распоредување ги подредува, прави листа на инструкциите што треба да се извршат. Таа е поврзана со инструкцискиот базен од каде што ги зема декодираните, но неизвршени инструкции.

Со **инструкцискиот базен** е поврзана и единицата за повлекување. Таа ги зема извршените инструкции и ги предава на податочната кеш-меморија од прво ниво.



Слика 9.13. Единица за прифаќање и извршување на инструкциите

Меморијата на Пентиум Про микропроцесорот е поделена во осум мемориски банки. Секоја локација од банките може да собере информација од еден бајт и секоја банка има вкупен капацитет од 8GB. Ова значи дека капацитетот на меморијата за Пентиум Про микропроцесор изнесува 64GB. Адресирањето е овозможено со воведување дополнителни адресни линии од A_{32} до A_{35}, што значи вкупно 36 адресни линии (2^{36} = 64GB). Пентиум про микропроцесорот користи 64-битна податочна магистрала која служи за пренос на осумте бајти од осумте мемориски банки. Бидејќи адресниот простор е зголемен зголемена е и големината на страниците на 2MB. Исто како 80476 и Пентиум I микропроцесорите, Пентиум Про користи внатрешна контрола на парност, односно покрај осунте битови во секоја мемориска локација додаден е уште еден бит за контрола на парноста.

Новина кај Пентиум Про микропроцесорот е внатрешното **коло за корекција на грешки** (EEC Error Correction Circuit). Ова коло овозможува детекција на две бит грешки и корекција на една бит грешка. За да се постигне ова потребни се осум дополнителни битови, кои се додадени на 64-битните броеви. Овие дополнителни осум битови го содржат кодот за автоматска корекција на грешка. EEC кодовите се многу доверливи и често се користат кај современите компјутери. Единствен недостаток е цената на SDRAM меморијата со широчина од 72 битови (64-битен податок+8-битен код).

9.5. Пентиум 2 микропроцесор

Од претходните микропроцесори, Пентиум 2 микропроцесорот се разликува по тоа што тој не е вграден во матичната плоча, туку постои **специјално дизајнирано пластично подножје** (слот) во кое се поставува микропроцесорот. Работната фреквенција се движи во границите од 233MHz до 450MHz со брзина на магистралата од 66MHz до 100MHz. Капацитетот на кеш-меморијата од второ ниво може да биде 512KB,1MB или 2MB. Пентиум 2 микропроцесорот користи 64-**битна податочна магистрала и 36-битна адресна магистрала.** Кеш-меморијата од прво ниво е со капацитет 32KB, но **кеш-меморијата од второ ниво не е веќе во истото интегрирано коло на микропроцесорот**, иако тие се многу блиску еден до друг, во исто пакување. На ваков начин ефективноста задоволува, а цената на микропроцесорот се намалува. Брзината на кеш-меморијата е двапати помала од брзината на Пентиум 2 микропроцесорот. Ако микропроцесорот има работна фреквенција 400MHz, тогаш кеш-меморијата работи со брзина 200MHz. Микропроцесорт Celeron е, всушност, веерзија на Пентиум 2 микропроцесорот со таа разлика што тој нема кеш-меморија од второ ниво. Имено, оваа меморија е посебно издвоена и вградена во матичната плоча и работи со брзина 66MHz. Xeon микропроцесорот е најдобра верзија на Пентиум 2 микропроцесорот, содржи кеш-меморија од второ ниво при што брзината на кеш-меморијата е иста со онаа на микропроцесорот.

Пентиум 2 микропроцесорот има 242 пина. Накратко ќе ја објасниме **функцијата на** речиси сите групи **пинови** од Пентиум 2 микропроцесорот:

- $\overline{A20}$ ⟶ Ги маскира позначајните адресни битови над A20. На ваков начин се постигнува компатибилност со 8086 микропроцесорот
- $\overline{BE7}$-$\overline{BE0}$ ⟶ Избираат една од осумте мемориски банки.
- $\overline{A35}$-$\overline{A3}$ ⟶ Избираат локација од мемориската банка.
- $\overline{ADS}$ ⟶ Означува испраќање валидна адреса.
- $\overline{AERR}$ ⟶ Активира контрола на парност на адресата.
- $\overline{AP1}$-$\overline{AP0}$ ⟶ Сигнализира појава на грешка во адресата.

$\overline{BCLK}$ ⟶ Одредува фреквенција на магистрала, 66MHz или 100MHz

$\overline{DERR}$ ⟶ Сигнализира грешка во магистралниот систем.

$\overline{BINT}$ ⟶ Го ресетира магистралниот систем.

$\overline{BNR}$ ⟶ (Bus Not Ready) Вметнува дигитски такта на чекање.

BP[3:2] ⟶ (Breakpoints) Сигнализираат појава на совпаѓање во работата на регистрите за дебагирање.

$\overline{BPRI}$ ⟶ (Bus Prioriy Reques) Се користи за доставување на барање за користење на системката магистрала.

$\overline{BP0}$-$\overline{BP1}$ ⟶ (Bus Request) Сигнализира дека Пентуим 2 испратил барање за магистрала.

$\overline{D63}$-$\overline{D0}$ ⟶ Податочни пинови

$\overline{DEFER}$ ⟶ Сигнализира дека надворешниот систем не може да го искомлетира машинскиот циклус.

$\overline{DEP7}$-$\overline{DEP0}$ ⟶ Сигнализираат појава на грешка во преносот на податоци.

$\overline{DRDY}$ ⟶ (Data Ready) Сигнализира испраќање на валиден податок.

$\overline{FERR}$ ⟶ Сигнализира грешка во работата на нумеричкиот копроцесор.

$\overline{FLUSH}$ ⟶ Ги онеспособува кеш линиите за читање и пишување.

$\overline{HIT}$ ⟶ Сигнализира успешно пронајден податок во кеш меморијата.

$\overline{HITM}$ ⟶ (Hit Modified) Се активира со цел да се одбегне другите единици да го користат кешот додека меморијата запишува во него.

$\overline{IERR}$ ⟶ Сигнализира внатрешна грешка во системот после што се иницира прекон.

$\overline{IGNNE}$ ⟶ Предизвикува игнорирање на грешката во работата на нумеричкиот копроцесор.

$\overline{INIT}$ ⟶ Се рестира системот со исклучок на кеш меморијата, влезно излезните бафери и регистрите за броеви со подвижна запирка.

$\overline{INTR}$ ⟶ Се активира кога некој надворешен уред бара прекин.

$\overline{LOCK}$ ⟶ Станува логичка нула кога се извршува инструкција со ваков префикс. Се употребува најчесто при директен пристап до меморија.

NMI ⟶ Немаскирачки прекин

PICCLK ⟶ Генерира дигитски такт со фреквенција ¼ од фреквенцијата на пинот BCLK.

$\overline{PM1}$-$\overline{PM0}$ ⟶ (Performance Monitor) Се користат при тестирање на перформансите на микропроцесорот.

PRDY ⟶ (Probe Ready) Сигнализира вклучено дебагирање.

$\overline{PREQ}$ ⟶ (Probe Request) Се активира при барање за дебагирање.

$\overline{REQ4}$-$\overline{REQ0}$ ⟶ (Request Signal) Служат за комуникација меѓу контролерот на магистрали и микропроцесорот.

$\overline{RESET}$ ⟶ Иницира извршување на софтвер со почетна адреса

FFFFFFF0H или 000FFFFF0H.

$\overline{RP}$ ⟶ (Request Parity) Се активира при барање на контрола на парност.

$\overline{RS2}$-$\overline{RS0}$ ⟶ (Reguest Status) Се активираат при барање за прикажување на состојбата на микропроцесорот.

SLP ⟶ Процесорот влегува во режим на мирување (Sleep).

$\overline{SMI}$ ⟶ (System Management Interrupt) Микропроцесорот влегува во мод на системски менаџмент.

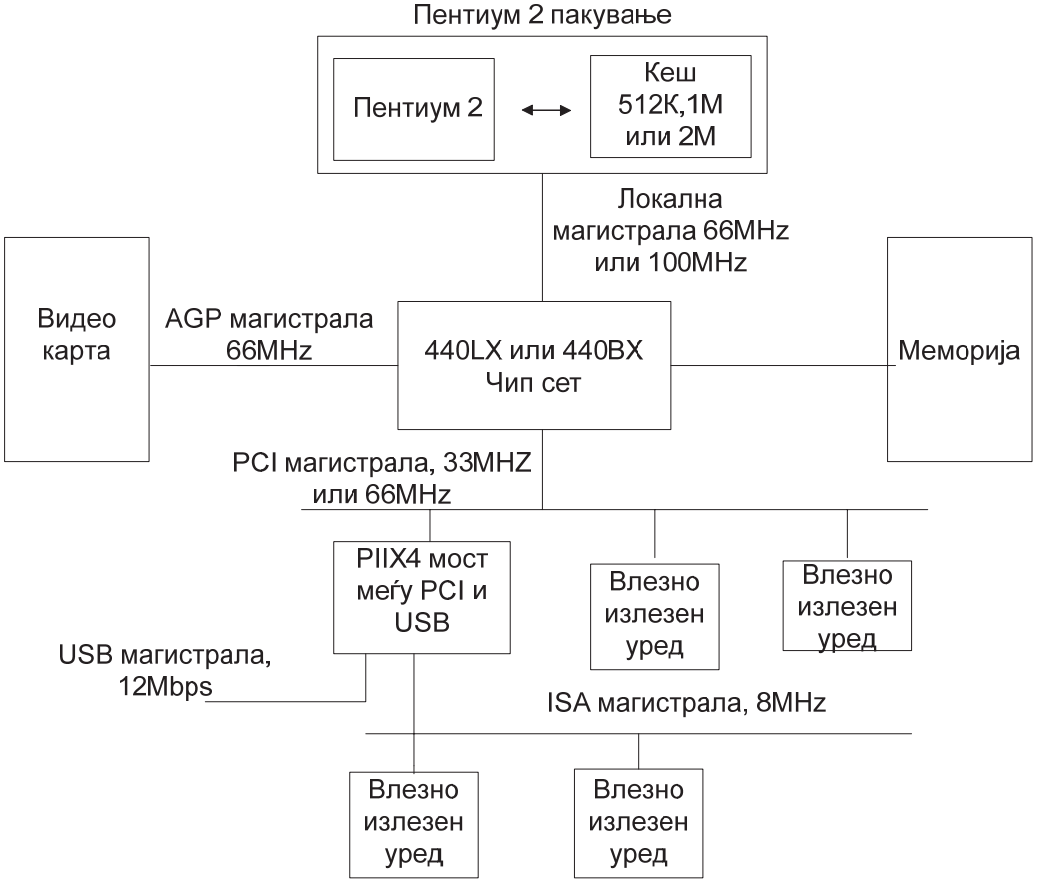
$\overline{THERMTRIP}$ ⟶ (Thermal Sensor Trip) Се активира ако температурата на микропроцесорот се зголеми над 130^{o}C.

TMS ⟶ (Test Mode Select) Избира вид на режим за тестирање.

$\overline{TRDY}$ ⟶ (Target Ready) Предизвикува микропроцесорот да започне операција на повратно пишување.

$\overline{VID4}$-$\overline{VID0}$ ⟶ (Voltage Data) Пинови за напојување и заземјување.

За контрола на преносот на податоци меѓу Пентиум 2 микропроцесорот и меморијата се користи **чип сет** 440LX или 440BX. Чип сетот е посредник во комуникацијата, за разлика од традиционалниот начин на комуникација каде микропроцесорот е директно поврзан за меморијата. На слика 9.14. е прикажана структурата на **Пентиум 2 компјутерски систем** заедно со сите видови на магистрали.



Слика 9.14. Структура на Пентиум 2 компјутерски систем

ISA (Industrial Standard Architecture) е најстар вид на магистрала, со широчина од 8 или16 битови и се користи за поврзување на бавни периферни уреди или мемории со работна фреквенција до 8MHz. Во иднина оваа магистрала ќе биде заменета со USB (Universal Serial Bus) магистралата, која може да биде со два различни пропусни опсези: 1.5Mbps или 12Mbps (bit per second-бит во секунда). PCI (Peripheral Component Interconnect) подржува 32 и 64-битни преноси меѓу микропроцесорот и периферните уреди или меморијата, со работна фреквенција до 33MHz. AGP (Accelerated Graphics Port) магистралата има иста работна фреквенција како и микропроцесорот, пропусен опсег од 528M бајти во секунда и се користи исклучиво за поврзување на видео картата со меморискиот систем.

9.6. Пентиум 3 микропроцесор

Основни карактеристики на Пентиум 3 микропроцесорот се: пренос на четири 64 битни податоци за време од еден дигитски такт при работна фреквенција од 133MHz, две кеш мемории од прво ниво со капацитет од по 16KB и една кеш меморија од второ ниво со капацитет од 512 KB, подршка на SSE инструкции и предвидување на гранења во програмите.

Кај микропроцесорот Пентиум 3, извршувањето на инструкцијата е поделено на **единаесет фази.** Во понатамошниот текст се објаснети сите фази од извршувањето на инструкцијата.

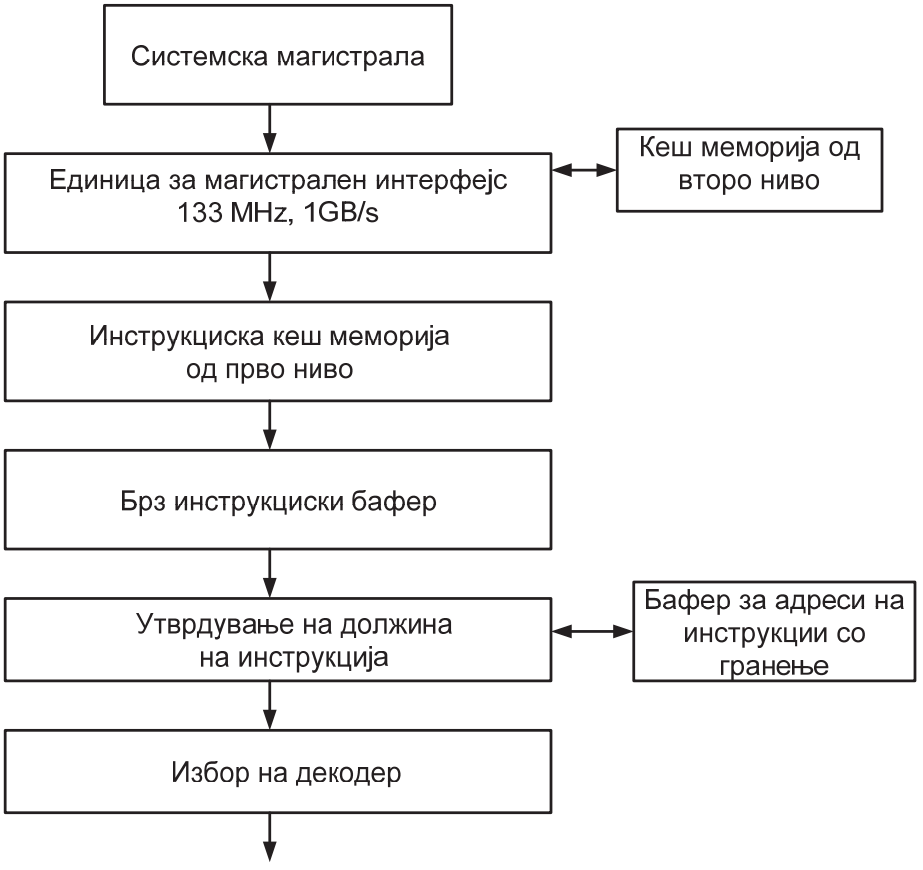
Првите три фази служат за **пренос и превземање на инструкцијата**. На слика 9.15. е прикажан блок дијаграмот на единицата за пренос на инструкцијата.

- Фаза 1: Во брзиот инструкциски бафер се внесува една кеш линија од инструкцискатa кеш меморија L1. Кеш линијата е голема 256 битови.
- Фаза 2:Од кеш линијата се издвојува инструкцијата. Бидејќи инструкциите се со променлива должина се издвојуваат првите 128 битови. Доколку се работи за инструкција за гранење (branch) адресата се сместува во баферот за инструкции за гранење од каде микропроцесорот подоцна ќе ја повика.
- Фаза 3: за тековната инструкција се бира еден од трите декодери.

После преносот на инструкцијата истата треба да се декодира и за таа цел се потребни две фази.

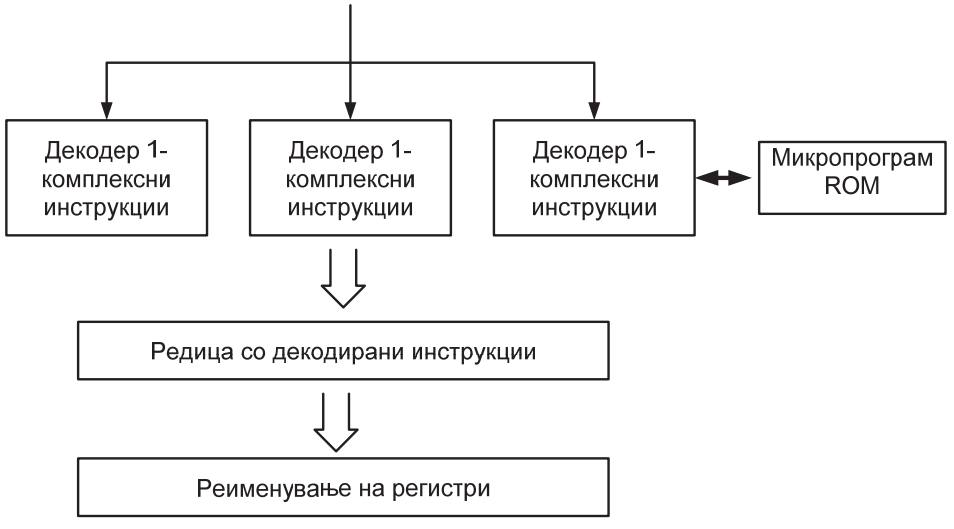
- Фаза 4: За декодирање на инструкцијата се користи еден од трите декодери што значи дека истовремено можат да се декодираат три инструкции. Како резултат од декодирањето се добиваат неколку микроинструкции.

- Фаза 5: Микроинструкциите се внесуваат во инструкциската редица. Ако од една инструкција се добијат повеќе од шест микроинструкции оваа фаза треба да се повтори.



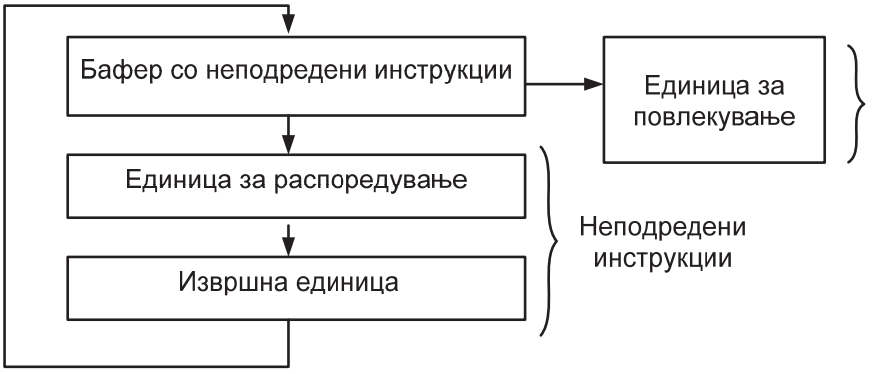
Слика 9.15. Единица за пренос на инструкција кај Пентиум 3 микропроцесор

На слика 9.16. е прикажан блок дијаграмот на **единицата за декодирање и единицата за реименување.**



Слика 9.16. Единица за декодирање и реименување кај Пентиум 3 микропроцесор

- Фаза 6: Бидејќи инструкциите не се извршуваат по истиот редослед по кои се напишани во програмата тешко е да се прати состојбата на регистрите. Поради тоа на секоја микроинструкција и се доделува по еден од 40 внатрешни регистри.
- Фаза 7: Микроинструкциите се внесуваат во баферот **(reoder buffer**) неподредени, без да се води сметка за нивниот редослед. Од овој бафер микроинструкциите се внесуваат во единицата за распоредување, под услов да се достапни сите податоци за извршување на инструкцијата и да има слободен влез во единицата за распоредување.
- Фаза 8: Во оваа фаза микроинструкцијата мора да биде во единицата за распоредување и таа се испраќа до соодветната извршна единица. Единицата за распоредување е поврзана со извршните единици преку четири порти.
- Фаза 9: Микропроцесорот Пентиум 3 има **пет извршни единици**: две единици за цели броеви, две единици за броеви со подвижна запирка и две единици за комуникација со меморија. За извршување на секоја инструкција е потребно време од еден дигитски такт.
- Фаза 10: **Единицата за повлекување** проверува кои микроинструкции од баферот без редослед се извршени
- Фаза 11: Единицата за повлекување ги трга извршените микроинструкции од баферот без редослед и ги подредува онака како што боле напишани во програмата. Ова е прикажано на сликата 9.17.

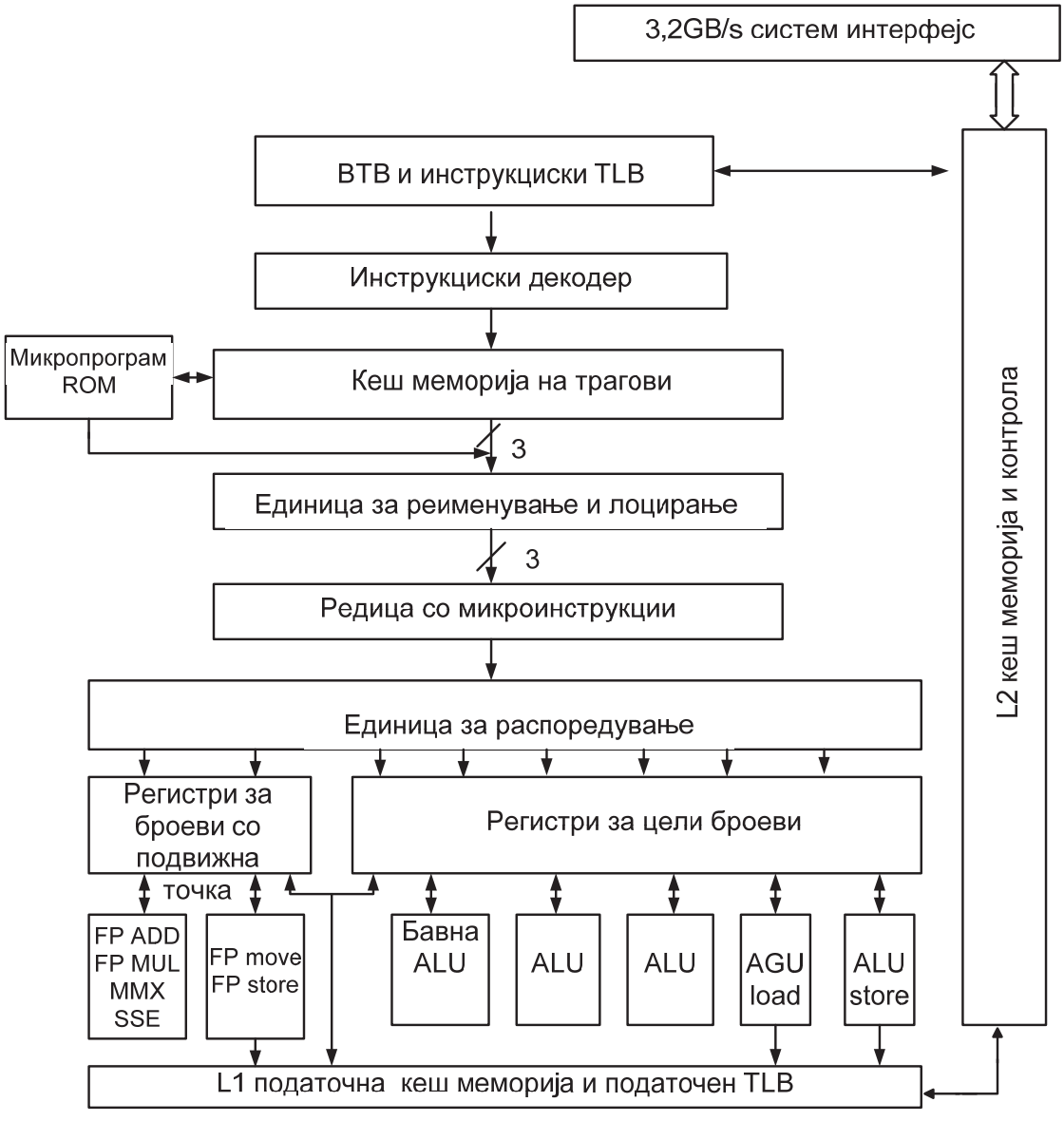


Слика 9.17. Функција на единицата за повлекување

9.7. Пентиум 4 микропроцесор

По архитектурата микропроцесорот Пентиум 4 е многу сличен со Пентиум Про микропроцесорот. Првиот Пентиум 4 микропроцесор имал работна фреквенција од 1,3GHz, а денешниве микропроцесори се со 3,2,GHz работна фреквенција. Постојат две пакувања од: 423-пинско PGA и 478-пинско FC-PGA2.

На слика 9.18. е прикажана **архитектурата на Пентиум 4** микропроцесор. Системската магистрала е со пропусен опсег 3,2GB/s. Широчината на магистралата изнесува 64 бита и работната фреквенција е 100MHz. Но нејзината брзина е четири пати поголема од фреквенцијата бидејќи за време од еден дигитски такт се пренесуваат четири податоци, па реално магистралата ќе пренесува како да има фреквенција од 400MHz. Ова техника е позната под името Quad Data Rate (QDR) што во превод значи **квадрантет податочен пропусен опсег**. Кеш меморијата од второ ниво може да биде со капацитет 256KB, 512KB, 1MB или 2MB, а податочната кеш меморија од прво ниво 8KB или 16KB.



Слика 9.18. Архитектура на Пентиум 4 микропроцесор

Можеби некој ќе помисли дека Пентиум 4 микропроцесорот нема инструкциска кеш меморија. Таква меморија постои, но таа се наоѓа после инструкцискиот декодер и е позната под името **кеш меморија на трагови**. Таа во себе може да чува 12K

микроинструкции. Бидејќи една микроинструкција е 100 битови, капацитетот на кеш меморијата со трагови изнесува 150KB. Поставувањето на инструкцискатa кеш меморија после декодерот го ,,штеди" декодерот од постојано декодирање на инструкции кој постојано се повторуваат како што се инструкциите за циклуси. Пред кеш меморијата со трагови поставени се два бафери: BTB (Branch Target Buffer) и TLB (Translation Lookside Buffer). **BTB баферот** преставува мала меморија која ги содржи адресите повиканите скок или инструкции за гранење.

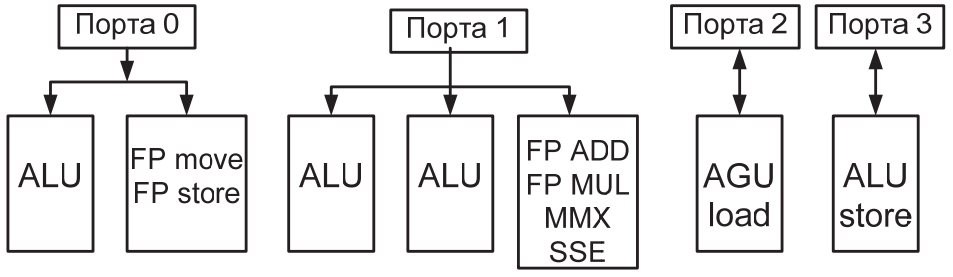
Декодерот може да обработи една инструкција за време од еден дигитски такт. Ако декодираната инструкција е сложена може да се добијат повеќе од четири микроинструкции. Микроинструкциите се препраќаат до ROM меморијата која всушност ги дава контролните сигнали потребни да се изврши инструкција. Кеш меморијата може да испрати по три микроинструкции за време на еден дигитски такт до единицата за реименување и релокација.

Единицата за лоцирање врши подготовка на ресурсите за да се биде спремно кога микроинструкцијата ќе пристигне со извршната единица. Единицата за лоцирање резервира еден од 126 бафери за секоја микроинструкција, преку која ја прати нејзината состојба. Микроинструкциите може да се извршуваат без да се води сметка за редоследот бидејќи откако ќе се извршат микропроцесорот ќе ги подреди користејќи ги токму овие бафери. Единицата за лоцирање резервира внатрешни регистри за сместување на резултатот добиен од извршувањето на микроинструкцијата. **Регистрите се поделени во неколку групи**: мемориски, регистри за работа со цели броеви и регистри за броеви со подвижна запира. На слика 9.18. регистрите за работа со подвижна запирка се бележат со буквите FP што е кратенка од англиските зборови Flating Point (превод-подвижна запирка). Доколку се работи за LOAD или STORE инструкција тогаш единицата за лоцирање ќе резевира еден од 48-те бафери за полнење (load) или еден од 24-те бафери за складирање (store), во зависност од тоа дали ќе се чита или пишува во меморијата.

Интересна функција има и **единицата за релокација**. Пентиум микропроцесорот има 32 регистри кои се видливи за програмерот: EAX, EBX, ECX, EDX, EDI, ESI, EBP и ESP. Овој број е премал посебно ако се земе во предвид дека не постои последователност во извршувањето на инструкциите. Поради тоа овие регистри се реименуваат во еден од 256 внатрешни регистри, 128 регистри за цели броеви и 128 регистри за броеви со подвижна запирка. Единицата за реименување може да обработува три микроинструкции за еден дигитски такт. Реименуваните и лоцирани микроинструкции влегуваат во редицата за микроинструкции каде што чекаат да ги превземе единицата за распоредување.

Единицата за распоредување ги подредува микроинструкциите по редоследот по кои биле декодирани. Во кеш меморијата на трагови и единицата за лоцирање овие инструкции беа измешани со цел да се потикне секој дел од микропроцесорот постојано да работи. Единицата за распоредување ги дели микроинструкциите во неколку групи: мемориски инструкции, брзи (едноставни)

инструкции, поедноставни инструкции за работа со броеви со подвижна запирка и сложени (бавни) инструкции. Ваквата распределба е многу важна за брз пренос на податоците до извршната единица. Единицата за распоредување и извршната единица се поврзани меѓу себе со четири порти. Од нив две порти пренесуваат по две микроинструкции во еден дигитски такт, а две порти по една микроинструкција. Портите и иршната единица се прикажани на слика 9.19. Пентиум 4 микропроцесорот има 5 извршни едииници, од кои две за работа со цели броеви и две за работа со броеви со подвижна запирка и две единици за пренос на податоци од RAM меморијата (load, store). На слика 9.19. последниве две единици се обележани со ознаката AGU(address Generator Unit) што во превод значи **генератор на адреси**. Гледаме дека портите 2 и 3 се поврзани само со единицата за комуникација со меморијата се со цел да се подобри брзината на преносот на потребните податоци.



Слика 9.19. Порти за поврзување на единицата за распоредување и извршната единица

Петте извршни единици работат паралелно. На пример, на единицата за обработка на броеви со подвижна запирка и требаат неколку дигитски такта по една микроинструкција. На додека таа работи портите 0 и 1 ќе испраќаат податоци до другите поедноставни аритметичко логички единици се со цел извршната единица да работи нон стоп со полн капацитет. Две од едноставните аритметичко логички единици можат да обработуваат по две микроинструкции во еден дигитски такт па на тој начин добиваме вкупен капацитет на сите извршни единици од седум инструкции по еден дигитски такт.

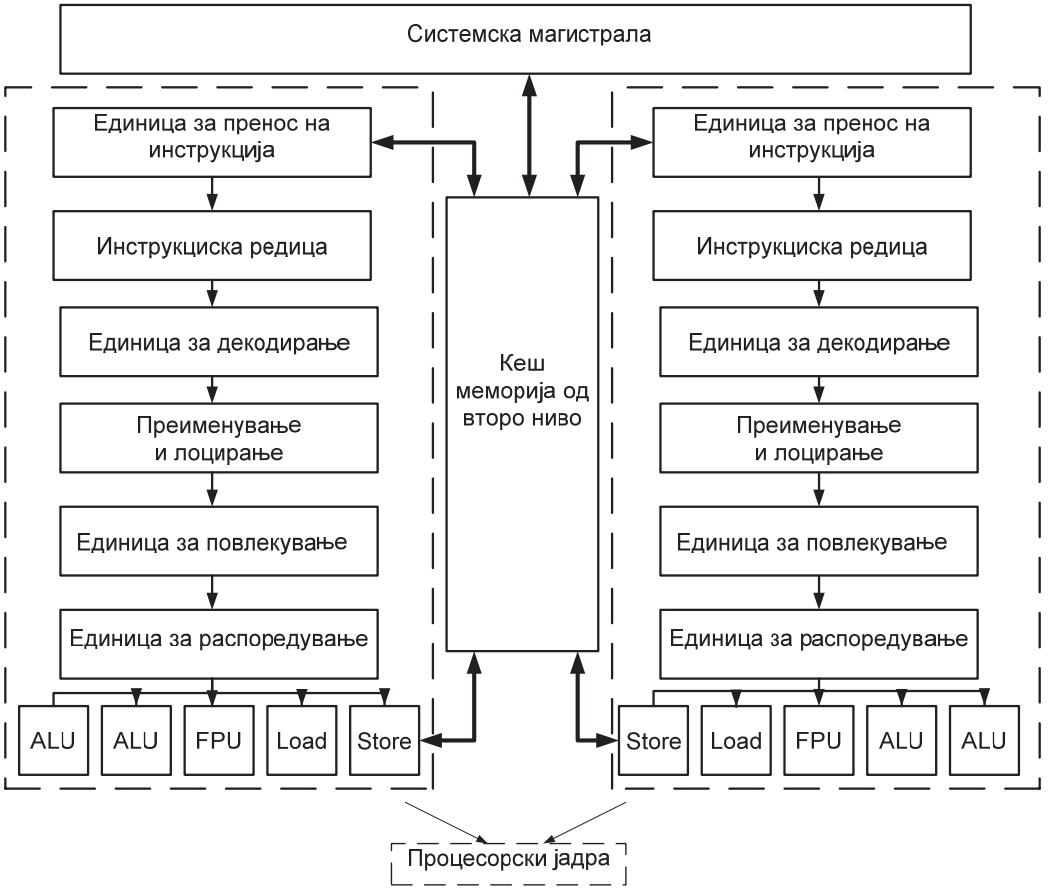
9.8. Пентиум Dual Core микропроцесор

Кај традиционалниот микропроцесор со едно јадро централната микропроцесорска единица се грижи за преносот, извршувањето и складирањето на податоците. Но проблем е брзината на RAM-от и магистралите која е многу помала од брзината на микропроцесорот и тоа преставува кочница во неговото работење. Проблемот уште повеќе се зголемува со мултипроцесирањето, односно извршување на повеќе апликации истовремено. На почетокот производителите на микропроцесорски системи пробале да го решат проблемот со мултипроцесирањето

преку зголемување на работната фреквенција на микропроцесорот. Но тоа било многу тешка и скапа работа бидејќи поголемата фреквенција значи поголемо греење што може да доведе до грешки во работата и оштетување. Поради тоа почнале да размислуваат за вградување на уште еден микропроцесор во самото интегрирано коло. Едноставно кажано две глави размислуваат подобро од една, четири раце се повредни од две. Двете јадра ги делат апликациите кои треба да се извршат. Дури и кога се извршува една програма едното микропроцесорско јадро ќе ги извршува инструкциите, а другото ќе пристапува до меморијата и ќе ги декодира пренесените инструкции. Бидејќи податоците кои ги разменуваат двете микропроцесорски јадра не го напуштаат микропроцесорскиот чип, нема потреба од засилување на сигналите, што секако доведува до помала потрошувачка на енергија и помало греење. Ова е посебно важно кај преносливите компјутери-лаптопови.

За да системот ги препознае двете јадра потребен е специјален софтвер познат под кратенката SMT (Simultaneous Multi-threading Technology), кој всушност преставува дел од оперативниот систем.

На слика 9.20. е прикажана архитектурата на двојадрен микропроцесор.



Слика 9.20. Архитектура на двојадрен микропроцесор

Процесорските јадра си имаат сопствени податочни и инструкциски кеш мемории од прво ниво, посебни декодери и извршни единици, но користат ист

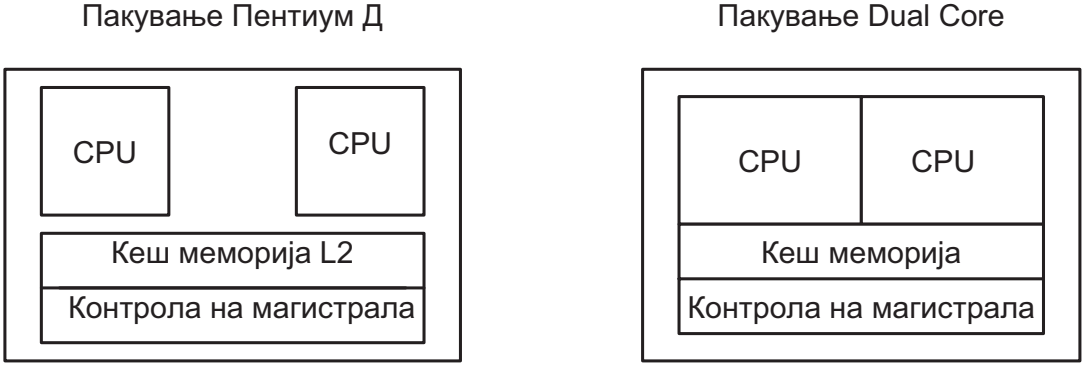
контролер на магистрали и иста кеш-меморија од второ ниво. Двојадрените микропроцесори бараат поголема кеш меморија од еднојадрените за да се намали времето на чекање. Денеска постојат и микропроцесори со четири јадра, но за да истите функционираат потребен е специјален мемориски контролер.

Двојадрените микропроцесори се разликуваат од мултипроцесорските системи. Кај последниве постојат два посебни микропроцесори со сопствени ресурси, а кај двојадрените ресурсите се заеднички. Мултипроцесорските системи се побрзи од системите со двојадрени микропроцесори, но и потрошувачката е поголема.

Денес постојат голем број на двојадрени микропроцесори кои се разликуваат по својата архитектура. **Пентиум Д** микропроцесорот е двојадрен, но двете централно микропроцесорски единици не се во еден чип, тие се посебни чипови, но поставени многу блиску еден до друг, во исто пакување. По својата архитектура Пентиум Д микропроцесорот е многу сличен со Пентиум 4 микропроцесорот. Имено сликата 9.20. речиси целосно одговара на архитектурата на Пентиум Д микропроцесорот. Покрај двата двојадрени микропроцесори во пакувањето е сместена кеш меморијата од второ ниво и единицата за магистрален интерфејс. Можеби извршната моќ на овој микропроцесор е поголема од онаа на еднојадрените микропроцесори, но греењето е многу големо.

Овој недостаток е надминат кај микропроцесорот **Пентиум Dual Core**. Кај овој микропроцесор сите единици се имлементирани во исто интегрирано коло (не пакување). Овие два вида двојадрени микропроцесори се прикажани на слика 9.21.

Пентиум Core 2 Duo е наследник на Пентиум Dual Core микропроцесорот. Но за разлика од него кај Core 2 Duo микропроцесорот двете микропроцесорски јадра имаат посебни кеш мемории од второ ниво и пристап до магистралата (FSB-Front Side Bus). Се разбира ова значи двапати поголема брзина и подобри перформанси.



Слика 9.21. Споредба меѓу Пентиум Д и Dual Core микропроцесор

Подоле се прикажани перформансите на неколку вида двојадрени микропроцесори:

- Пентиум Dual Core ⟶ фреквенција 2GHz, кеш од второ ниво 1MB, брзина на магистрала 200MHz
- Пентиум Core 2 Duo ⟶ фреквенција 1,86GHz, кеш од второ ниво 2MB, брзина на магистрала 266MHz
- Пентиум Core 2 Extreme ⟶ фреквенција 3,2GHz, кеш од второ ниво 2×48MB, брзина на магистрала 333MHz

Со секој нов двојадрен микропроцесор големината на кеш меморијата се зголемува, а исто и брзината на системската магистрала.

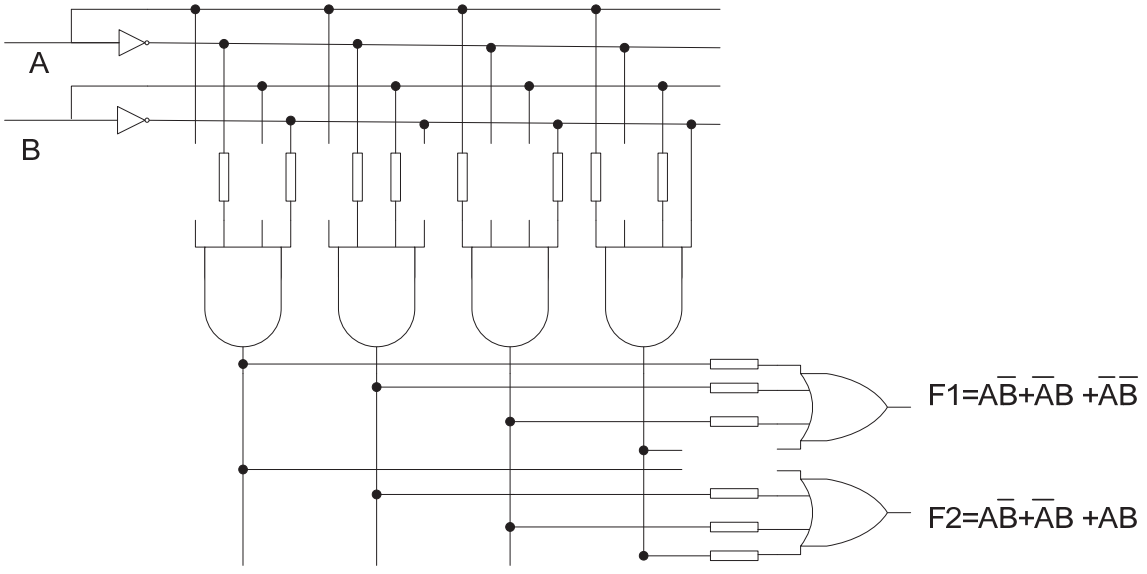
9.10. Интегрирани кола за формирање на Пентиум микрокомпјутерски системи

Пентиум микропроцесорите наоѓаат огромна примена. Најчесто кога ќе кажеме Пентиум микропроцесор мислиме на персоналните компјутери, но тие се користат и во мобилните телефони, преносните компјутери, аудио и видео опремата итн. На почетокот од втората тема, 2.1. Вовед во микрокомпјутерски систем истакнавме дека микрокомпјутерскиот систем е составен од централна процесорска единица, работна (привремена) и трајна меморија. Со додавање на периферни уреди се добиваат компјутерски системи. На пример, со додавање на тастатура и монитор може да се добие минимален компјутерски систем. Ние ќе се задржиме на поврзување на Пентиум микропроцесорите со меморијата и периферните кола. Ќе го објасниме адресното декодирање и употребата на програмибилни декодери. Исто така детално ќе ја објасниме архитектурата и пиновите на чип сетот, кои се специјално дизајнирани интегрирани кола за поврзување на Пентиум микропроцесорот со меморијата и периферните уреди

9.10.1. Интегрирани кола за поврзување на Пентиум микропроцесор со меморија

Програмибилните декодери се познати под кратенката **PLD** (Programmable Logic Device). Постојат повеќе видови PLD декодери кои функционираат на ист начин, но имаат поинакви имиња: PLA (Programmable Logic Array), (Programmable Array Logic) GAL (Gated Array Logic). Програмирањето на декодерите се врши на ист начин како и програмирањето на PROM мемориите, со кои се запознавме во првата тема, 1.7.3. ROM мемории. Содржната на некои PLD декодери може да се менува, односно да се брише и повторно репрограмира, исто како и EPROM мемориите.

На слика 9.22. е прикажана структурата на еден општ PLD декодер. Тој е состваен од два реда: ред од И кола и ред од ИЛИ кола. Секој влез преставува логичка варијабила. Секој излез од декодерот зависи од состојбата на влезовите. Бројот на влезови на едно ИЛИ коло е еднаков на бројот на И кола. Каква ќе биде зависноста на излезите од влезовите зависи од тоа кои осигурувачи се прегорени, а кои не. Прегорените осигурувачи на влез од И колата значат состојба на логичка нула и тие не влијаат врз сосотојбата на излезот од И колото, а прегорените осигурувачи на влез од ИЛИ колата значат состојба на логичка нула. Сите осигурувачи во фабрички произведените декодери се непрегорени. Тие се прегоруваат според потреба преку пуштање на голема струја низ нив.



Слика 9.22. Структура на PAL програмибилен декодер

Колото 16L8 е PAL декодер со 10 фиксни влеза, два фиксни излези и шест пинови кои може да се програмираат како влезни или излезни. На влез од декодерот PAL16L8 доаѓаат адресните сигнали од микропроцесорот, а излезните сигнали од декодерот се носат на влезните пинови $\overline{CE}$ (Chip Enable) од ROM или RAM чиповите и служат за селекција на мемориските компоненети.

На слика 9.23. е прикажан мал Пентиум мемориски систем. Овој систем се состои од осум 27512 EPROM мемориски чипови (64K×8), со вкупен капацитет од 512K и адресен простор од FFF80000H до FFFFFFFFH. За декодирање се употребени два PAL16L8 декодери. Битовите од A19 до A28 се влезови на вториот декодер. Зависноста на излезот U2 од адресните битови е дадена со равенката:

$$\overline{U2} = A19 \cdot A20 \cdot A21 \cdot A22 \cdot A23 \cdot A24 \cdot A25 \cdot A26 \cdot A27 \cdot A28$$

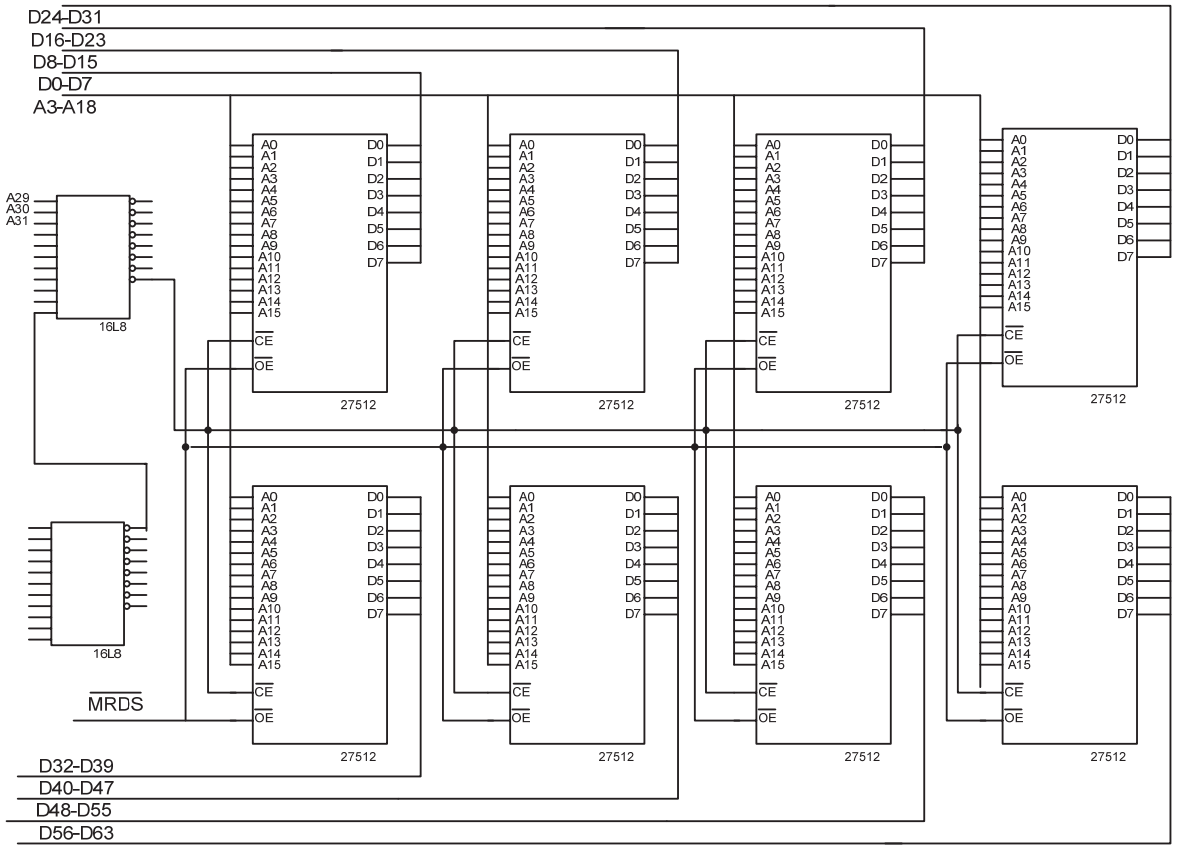
Влезови на првиот декодер се адресните битови A31, A30, A29 и излезот од вториот декодер. Првиот декодер го генерира сигналот за пинот $\overline{CE}$. Кога овој пин ќе

се активира со оспособуваат сите осум мемориски чипови. Зависноста на излезот $\overline{CE}$ од влезовите кај првиот PAL16L8 е дадена со равенката:

$$\overline{CE} = \overline{U2} \cdot A29 \cdot A30 \cdot A29$$

За да се активира сигналот CE потребно е едресните битови од A31 до A19 да бидат логички единици. Вториот пин за селекција на мемориските чипови $\overline{OE}$ го активира контролниот сигнал $\overline{MRDC}$ и начинот на негово генерирање беше прикажан на сликата 9.4. Битовите од A3 до A18 служат за избор на мемориска локација од селектираните мемориски модули.

Податочната магистрала на Пентиум микропроцесорот е 64 битна. Кога ќе се активираат сите осум мемориски чипови 27512 секој од нив ќе даде по еден бајт за да се добие 64-битен податок. Податочните излези од секој мемориски чип се поврзани со посебна група од осум водови од податочната магистрала: D7-D0, D8-D15,D16-D23, D24-D31,D32-D39, D40-D47, D48-D55, D56-D63.



Слика 9.23. Поврзување на Пентиум микропроцесор со 64 битна меморија

Интегрирано коло Интел 865G чип сет

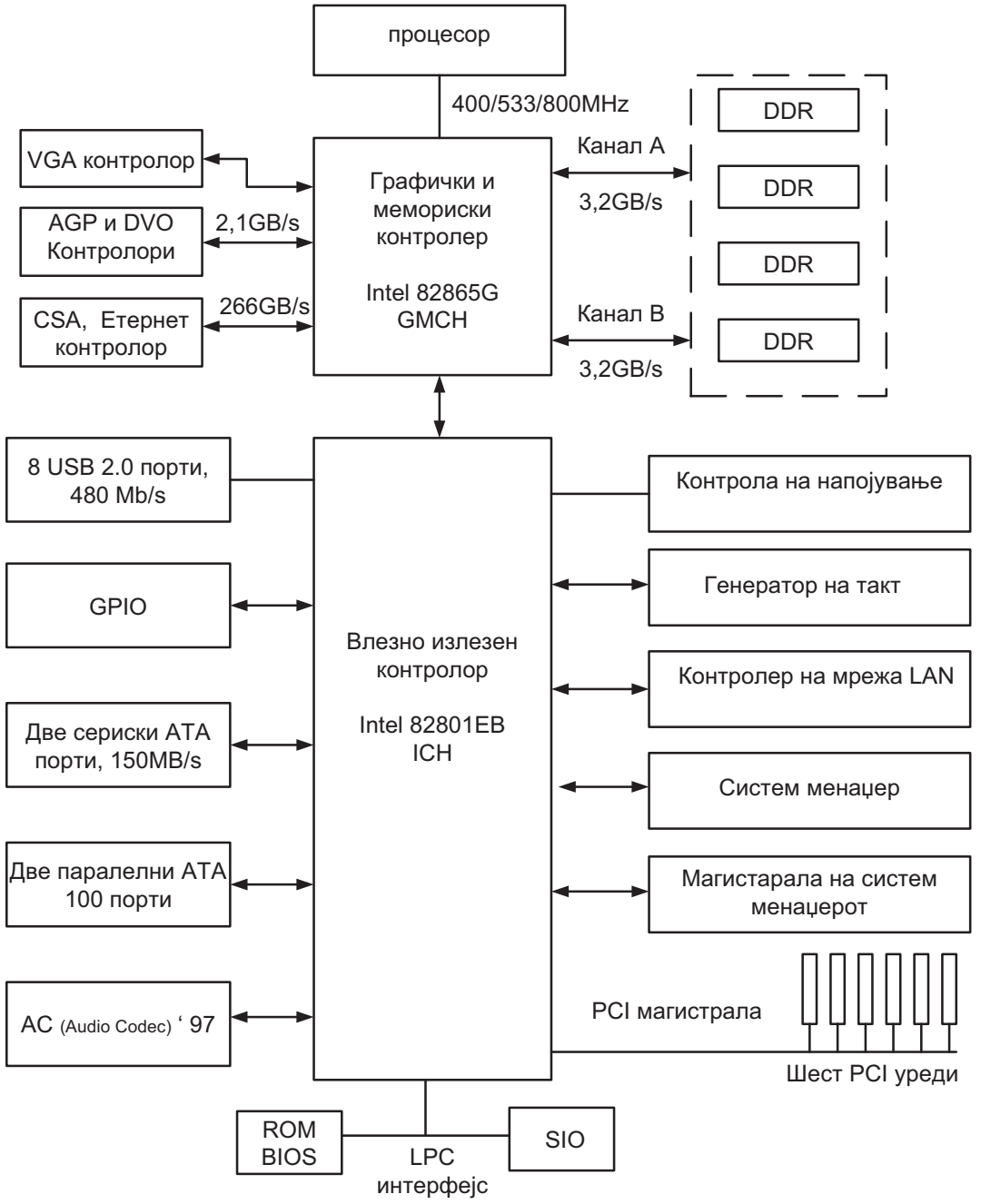
Кај Пентиум компјутерските ситеми микропроцесорите имаат повеќе извршна функција, а **контролната функција ја извршуваат множество од специјални интегрирани кола познати под името чип сет** (Chip Set). Англискиот збор set во превод значи множество, а зборот chip значи чип или интегрирано коло. Овие интегрирани кола ги генерираат контролните сигнали за управување со меморијата и периферните уреди. Чип сетот е посредник во комуникацијата меѓу микропроцесорот и останатите компоненети на матичната плоча. Дури и самата матична плоча се именува според видот на чип сетот вграден во неа. Анализирајќи ја функцијата на посебните единици од блок дијаграмот на чип сетот ние ќе се запознаеме со сите поважни делови на матичната плоча.

Ќе се задржиме на еден конкретен чип сет Интел 865G. Тој се користи во десктоп компјутерските системи кои имаат во себе Пентиум 4 микропроцесор со 512KB кеш меморија од второ ниво и фреквенција на системската магистрала 400MHz, 533MHz или 800MHz. Секој чип сет е составен од **две основни компоненти**: графички и мемориски контролор (82865G CMCH-Graphics Memory Controller Hub) и влезно излезен контролор (82801EB ICH-Input/output Controller Hub). Англискиот збор hub во превод значи средиште, јазол. Тоа се места во системот каде се врши проширување на истиот, разгранување, на пример од еден излез се добиваат повеќе други излези. Графичкиот и мемориските контролор е поврзан со микропроцесорот, системската меморија, единицата за брза комуникација, влезно-излезното средиште (hub) итн. На слика 9.24. е прикажан блок дијаграмот на чип сетот 865G. Сега накратко ќе ги објасниме функциите и англиските термини за сите компоненти од чип сетот 865G.

- **VGA** (Video Graphics Array) е многу популарен стандард за поврзување на мониториот и различните видови аналогни дисплеи. Овој стандард обезбедува резолуција од 640×480 пискели, фреквенција на обновување од 60Hz и истовремено прикажување на 16 бои. При помала резолуција од 320×200 пискели, можат да се прикажат 256 бои. Денес овој стандард е заменет со SVGA (Super VGA) стандардот кој овозможува прикажување на 16 милиони бои при резолуција од 800×600 на 14- инчен монитор или резолуција од 1200×1600 на 20-инчен монитор. VGA конекторот на матичната плоча има 15 пинови.

- **AGP** (Accelerated Graphics Port) е порта за приклучување на графички картички и таканаречени 3D акселатори (графички забрзувачи). AGP портата има директен пристап до системската меморија. AGP магистралата е 32-битна, со работна фреквенција од 66MHz, што дава вкупен пропусен опсег од 2.1GBps. Пред појавата на AGP портите за приклучување на видео картичките се користеле PCI портите со двапати помал опсег.

- Портите DVO (Digital Video Output) служат за поврзување на едноканални и двоканални дигитални видео уреди со поголеми резолуции и фреквенции на обновување (refresh) како што се на пример LCD екраните.



Слика 9.24. Блок дијаграм на Интел 865G чип сет

- **CSA** (Communication Streaming Architecture) е специјално дизајниран концепт за поврзување на графичкиот и мемориските контролор (GMCH) со Етернет контролорот. Етернет е протокол за формирање на компјутерски мрежи. Контролорот за брза комуникација (CSA) обезбедува пренос од 266 милиони трансфери во една секунда при работна фреквенција од 66 MHz.

- За поврзување на чип сетот 865G со системската меморија се користат два канала со пропусен опсег од 3,2GB/s. Капацитетот на меморијата изнесува 4GB, организирана е во четири банки и се употребени DDR SDRAM чипови. Кратенката DDR (Data Dual Rate) значи пренос на два податока за време од еден дигитски такт.

Влезно излезниот контролер 82801EB во себе ги интегрира:

- **ATA** 100 **контролорот** –паралелната ATA порта е замена за 40-пинската IDE (Integrated Drive Electronic) порта за поврзување на хард-диск меморијата. Да напоменеме дека IDE кабелот е долг само неколку сантиметри што е уште еден недостаток во однос на ATA каблите за поврзување.
- Два сериски ATA контролери – сериската ATA порта има поголема фреквенција од паралелната, но помал шум во работењето. Таа е седум пинска и има пропусен опсег од 150MB/s
- EHCI (Enhanced Host Controllor Interface) е контролор за USB 2.0. портите со пропусен опсег од 480MB/s
- UHCI (Universal Host Controllor Interface) е контролор за USB 1,1 и 1,0, портите со пропусен опсег од 12MB/s. Да напоменеме дека чип сетот 865G нема USB 1,1 порти туку само осум USB 2.0 порти, но тие се коматибилни со USB 1.1. портите.
- Контролерот за LPC (Low Pin Count) магистралата која е замена за старата и бавна ISA магистрала. Оваа магистрала се користи за поврзување на: EEPROM чипот кој во себе ја содржи BIOS програмата и едноставните влезно излезни уреди (SIO-Simple Input Output) како што се сериската, паралелната порта, глувчето и тастатурата.
- **Контролорот на PCI** (Peripheral Component Interconnect) магистралата. На пример уред за PCI слотот е звучната картичка.
- AC'97 (Audio Codec) контролерот кој се користи и за аналогно дигитално претварање на звучните сигнали .
- **Контролерот за поврзување со компутерски мрежи** (LAN-Local Area Network)
- GPIO (General Purpose Input Output) контролерот ги контролира 8 пинските GPIO порти кои се осум пински и се многу згодни за поврзување на едноставни уреди како што се лед диодите или прекинувачите. GPIO портите можат да се програмираат како влезни или излезни порти.

Откако се запознавме со поважните единици од чип сетот 865G ќе се обидеме да ја објасниме функцијата на покарактеристичните, поопштите пинови на ова интегрирано коло. Ќе видиме дека голем број од пиновите на чип сетот имаат слична функција со пиновите на Пентиум микропроцесорите. Поради тоа некои од пиновите нема да ги објаснуваме бидејќи со нив се запознавме во 9.2. поглавјето. Пин дијаграм на Пентиум 1 микропроцесор и 9.5. Пентиум 2 микропроцесор. На слика

9.25. се прикажани групите пинови на интегрираното коло 865G, поделени според нивната функција.

За комуникација меѓу графичкиот и меморискиот контролер (82865G) и микропроцесорот се користат следниве сигнали:

HD ⟶ (Host Data)-Податочна магистрала на контролерот
HA ⟶ (Host Address)- Адресна магистрала на контролерот
HADSB ⟶ (Host Address Strobe)- Сигнал за најава на испраќање на адреса
DBSY ⟶ (Data Bus Busy)- Податочната магистрала не завршила со преносот
DINV ⟶ (Dynamic Bus Inversion)- Одредува кој збор од 64-битниот податок ќе се пренесува
DRDY ⟶ (Data Ready)-Податокот пристигнал до саканата дестинација
CPURESET ⟶ Ресетирање на микропроцесорот
HIT,HITM ⟶ Пинови за пристап да кеш меморијата
HTRDY ⟶ (Host Target Ready)- уредот е спремен да ги прифати податоците
PROCHOT ⟶ Зголемување на температурата на микропроцесорот
AP ⟶ (Address Parity)-Контрола на парност на адресата
DP ⟶ (Data Parity)-Контрола на парност на податоците
IERR ⟶ (Internal Error)- Внатрешна грешка во системот
BINIT ⟶ (Bus Initialization) –Ресетирање на магистралите

За комуникацијата со системската меморија DDR SDRAM се користат следниве сигнали:

SCMDCLOK ⟶ (Differential DDR Clock)-За превземање на адресните и податочните сигнали потребен е растечки раб на овој сигнал
SCS ⟶ (Chip select)- Се избира еден SDRAM чип
SMAA ⟶ (Memory Address)-Мемориска адреса
SRAS, SCAS ⟶ (Row Address Strobe, Column Address Strobe)-го контролираат мултиплексот на адресната магистрала

SBA ⟶ (Select Bank)-Избор на мемориска банка
SWE ⟶ (Write Enable)-Оспособување за читање
SDQ ⟶ (Data Lines)-Податочни пинови
SDQS ⟶ (Data line Strobe)-Сигнал за најава за секој бајт од 64-битниот податок

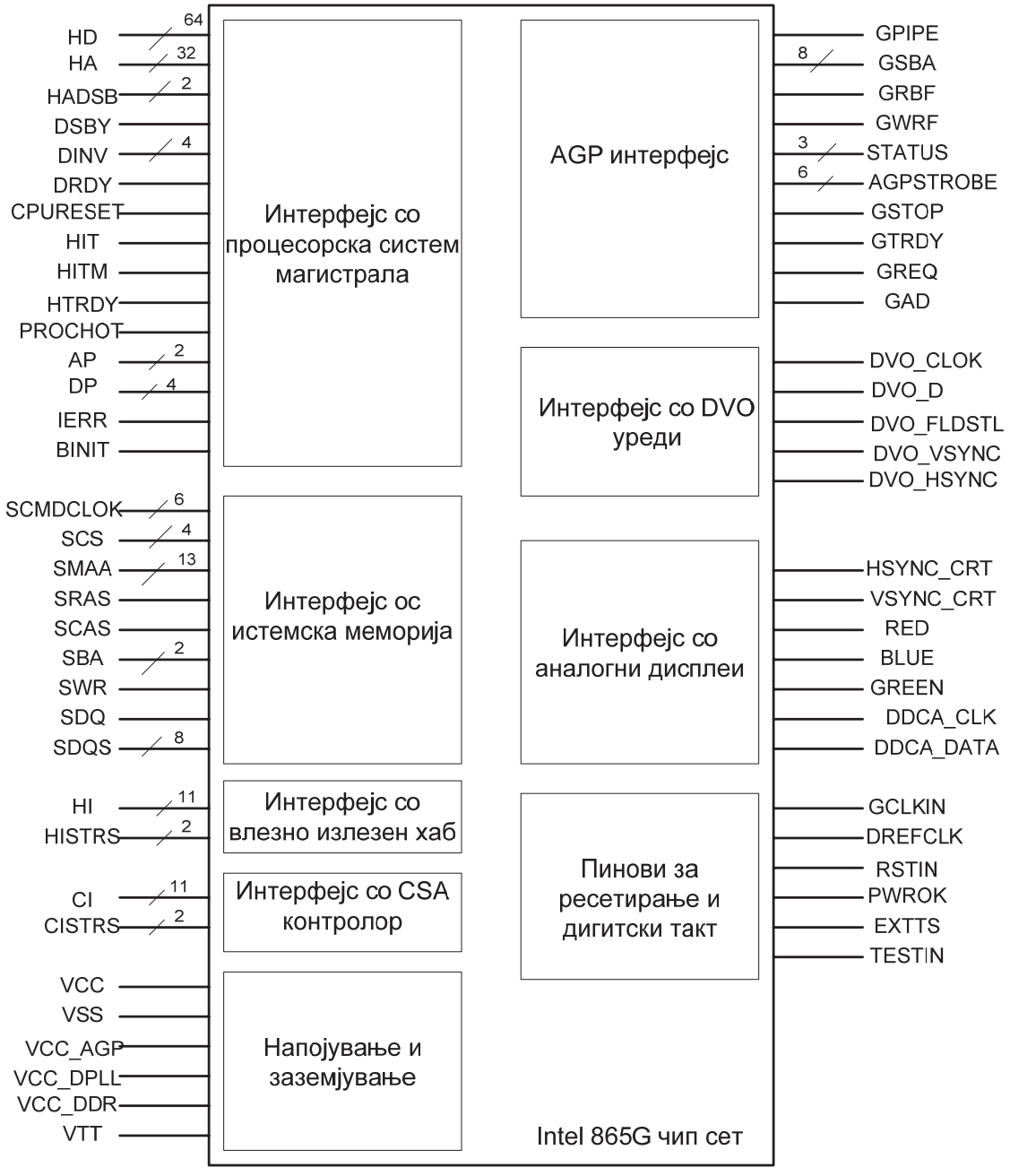
За комуникација на чип сетот со контролорот за брза комуникација и влезно излезниот хаб се користи стробиран режим при што се пренесуваат само податочни

сигнали и сигнали за најава. Сигналите за комуникација со влезно излезниот хаб се следни:

HI ⟶ Податочни сигнали
HISTRS ⟶ Сигнали за најава (Strobe) на преносот на податоци

Сигнали за комуникација со контролорот за брза комуникација (CSA) се:

CI ⟶ Податочни сигнали
CISTRS ⟶ Сигнали за најава (Strobe) на преносот на податоци



Слика 9.25. Пин дијаграм на чип сет Интел 865G

Сигналите за комуникација со AGP контролерот може да се поделат во неколку групи: адресни, контролни, статусни и сигнали за најава. Подолу е дадена нивната функција.

GPIPE → (Pipe read)-Кај pipeline концептот најавува пренос на податоци
GSBA → (Sideband Address)- Се воведуваат нови дополнителни адресни линии
GAD → (Address Data)-Адресно податочни сигнали
GRBF → (Read Buffer Full)- Иницира читање
GWRB → (Write Buffer Full)- Иницира пишување
Status → Статусните битови дефинираат состојба на пренос
Strobe → Сигналите за најава се користат за реализација на тајмингот за пренос на податоци кој трае четири дигитски такта
GSTP → (AGP Stop)-AGP контролорот го прекинува преносот
GTRDY → (AGP Target Ready)-Уредот е спремен да ги прифати подтоците
GREQ → (AGP Request)-AGP доставува барање за користење на магистралите

Сигнали за комуникација меѓу чип сетот и контролорот на DVO портите се:

DVO_CLOK → Дигитски такт за DVO контролорот
DVO_D → Податочна магистрала за пренос на 12-битен пиксел податок за секој дигитски такт
DVO_VSYNC → Вертикална синхронизација
DVO_HSYNC → Хоризонтална синхронизација
DVO_FLDSTL → (TV Field and Flat Panel Stall Signal)–сигнал за избор на дигитален видео уред

При користење на аналогни видео уреди се користат следниве контролни сигнали:

HSYNC, VSYNC → Вертикална и хоризонтална синхронизација
RED, GREEN, BLUE → Аналогни сигнали за боите
DDCA_CLK → Дигитски такт за контролорот за аналогни видео уреди
DDCA_DATA → Податочни сигнали

Други поопшти контролни сигнали се:

GCLKIN → (AGP Clock)-Дигитски такт со фреквенција 68MHz
DREFCLK → (Display Clock Input)-Дигитски такт со фреквенција 48MHz

RSTIN ⟶ (Reset In) Влезен пин за ресетирање на чип сетот
TESTIN ⟶ Чип сетот влегува во тест мод
VCC_AGP ⟶ Напојување за AGP портата
VCC_DDR ⟶ Напојување за системската меморија
VTT ⟶ Напојување за системската магистрала

Можеме да заклучиме дека по својата функционалност и сложеност во работата на интегрираните кола за формирање на Пентиум копјутерски системи се многу слични самите микропроцесори.

Заклучоци:

Кај микропроцесорот Пентиум 1 кеш-меморијата од прво ниво L1 е поделена на две кеш мемории од по 8KB, од кој првата е инструкциска, а втората е податочна кеш-меморија.

Суперскаларната архитектура на Пентиум микропроцесорите произлегува од постоењето на трите извршни единици: една за броеви со подвижна запирка и две за цели броеви (U pipe, V pipe). Ова значи дека паралелно може да се извршуваат три различни инструкции.

Меморијата на Пентиум микропроцесорот е голема 4GB и таа е поделена во 8 мемориски банки по 512MB. Посебниот пристап до секоја мемориска банка дозволува пристап до 8, 16, 32 и 64-битни податоци. За адресирање на меморискиот простор се користи 64 битна адресна магистрала.

Кај микропроцесорот Пентиум 1 читањето од меморија се врши преку циклусот за проток (burst). Со еден циклус за проток може да се пренесат четири 64-битни броеви за времетраење од пет дигитски такта.

Пинот $\overline{BRDY}$ (Burst Ready) се активира кога меморијата или периферниот уред презема или дава податоци на податочната магистрала.

Пинот $\overline{A20}$ се активира кога микропроцесорот треба да работи во реален мод на работа.

За контрола на парност на адресите се користат два пина AP и $\overline{APCHK}$. Контрола на парност се врши и на податочните битови прочитани од меморијата или некој периферен уред . За секоја мемориска банка постои по еден пин за контрола на парност DP7-DP0 (data parity).

Постојат специјални пинови за контрола на кеш меморијата. Пинот $\overline{KEN}$ (cache enable) ја оспособува внатрешната кеш меморија. Пинот WB/$\overline{WT}$ (write-

back/write-through) бира операција за кеш меморијата. Преку активирање на пинот $\overline{FLUSH}$ се поништува внатрешната кеш меморија.

Пентиум микропроцесорите може да работат во три мода: реален мод, заштитен мод и виртуелен мод на работа. Во заштитен мод адресниот простор се зголемува до 4GB (2^{32}), а со виртуелизација добиваме екстремно голем адресен простор од 64TB (2^{46}).

Во заштитен мод на работа сегментниот регистар содржи таканаречен селектор. Селекторот овозможува избор на еден од 8192 дискртиптори. Преку дискрипторот се пронаоѓа саканиот сегмент во меморијата. Откако ќе се пронајде саканиот сегмент, се пристапува кон пронаоѓање на саканиот податок. Позначајните шеснаесет бита од адресата ја даваат вредноста на селекторот, а понезначајните шеснаесет бита ја даваат вредноста на поместувањето.

Во заштитно виртуелен мод постои само еден директориум, чија почетна адреса се наоѓа во контролниот регистар CR3. Директориумот е составен од 1024 локации. Која од овие локации ќе биде избрана, зависи од вредноста на битовите од 22 до 31 од виртуелната адреса. Избраната локација од директориумот нè упатува до една од 1024-те табели на страници. Која локација од табелата на страници ќе биде избрана зависи од вредноста на битовите од 12 до 21 од виртуелната адреса. Избраната локација од табелата на страници нè упатува до почетокот на една виртуелна страница. На крај, која локација од страницата ќе биде избрана, зависи од поместувањето, односно битовите од 0 до 11 од виртуелната адреса.

Пентиум Про микропроцесорот по својата структура се разликува од сите претходни микропроцесори. Новост се двете кеш мемории од различно ниво, инструкцискиот базен за чување на декодираните инструкции, единицата за распоредување на инструкциите кои треба да се извршат и единицата за повлекување на веќе извршените инструкции.

Пентиум 2 микропроцесорот се разликува по тоа што тој не е вграден во матичната плоча, туку постои специјално дизајнирано пластично подножје (слот) во кое се поставува микропроцесорот. Кеш-меморијата од прво ниво е со капацитет 32KB, но кеш-меморијата од второ ниво не е веќе во истото интегрирано коло на микропроцесорот, иако тие се многу блиску еден до друг, во исто пакување.

Микропроцесорите Пентиум 3 и Пентиум 4 процесорите се слични по својата архитектура. Единицата за лоцирање врши подготовка на ресурсите за да се биде спремно кога микроинструкцијата ќе пристигне со извршната единица. Единицата за релокација ги реименува 32-та регистри кои се видливи за програмерот 9EAX, EBX, ECX, EDX, EDI, ESI, EBP и ESP) во еден од 256 внатрешни регистри, 128 регистри за цели броеви и 128 регистри за броеви со подвижна запирка. Единицата за распоредување ги подредува микроинструкциите по редоследот по кои биле

декодирани. Во кеш меморијата на трагови и единицата за лоцирање овие инструкции беа измешани со цел да се потикне секој дел од микропроцесорот постојано да работи.

Пентиум 4 микропроцесорот има 6 извршни едииници, од кои две за работа со цели броеви и две за работа со броеви со подвижна запирка и две единици за пренос на податоци од RAM меморијата.

Пентиум Д микропроцесорот е двојадрен, но двете централно микропроцесорски единици не се во еден чип, тие се посебни чипови, но поставени многу блиску еден до друг, во исто пакување. Покрај двата двојадрени микропроцесори во пакувањето е сместена кеш меморијата од второ ниво и единицата за магистрален интерфејс. Можеби извршната моќ на овој микропроцесор е поголема од онаа на еднојадрените микропроцесори, но греењето е многу големо. Овој недостаток е надминат кај микропроцесорот Пентиум Dual Core. Кај овој микропроцесор сите единици се имлементирани во исто интегрирано коло.

Секој чип сет е составен од две основни компоненти: графички и мемориски контролор (82865G CMCH-Graphics Memory Controller Hub) и влезно излезен контролор (82801EB ICH-Input/output Controller Hub).

Влезно излезниот контролер 82801EB во себе ги интегрира: контролорот за поврзување со хард диск меморијата, ATA контролорите за USB портите, контролорите за ISA и PCI магистралите, контролорот за поврзување со LAN мрежи, GPIO контролорот за поврзување со лед диоди или прекинувачи.

Прашања и задачи:

1. Како логиката за предвидување на гранењата ја подобруваат брзината на работа на микропроцесорот?
2. Објасни што значи терминот Суперскаларна архитектура?
3. Кои пинови се користат за контрола на парност кај Пентиум микропроцесорите?
4. Кои сигнали се активираат при прз пренос на податоци од меморија во микропроцесор?
5. Какво коло се користи за генерирање на сигнали за контрола на

меморијата и периферните уреди кај Пентиум микропроцесорите?

6. Наброј ги пиновите за контрола на кеш меморијата?

7. Кои пинови од микропроцесорот Пентиум 2 се користат за контреола на нумеричкиот копроцесор?

8. Колку изнесува адресниот простор на микропроцесорот Пентиум 1 кога тој работи во:
   А) реален мод
   Б) заштитен мод
   В) виртуелен мод

9. Објасни ја постапката за избор на сегмент од меморијата кога микропроцесорот работи во заштитен мод на работа!

10. За што служат дискрипторите?

11. Колку изнесува големината на еден сегмент ако неговиот лимит изнесува 01FFFH и битот G е еднаков на еден?

12. Од колку делови е составена виртуелната адреса на микропроцесор 80386? За што служи секој од овие делови?

13. Одреди го:
    - редниот број на табела на страници
    - редниот број на страница
    - растојанието од почетокот на страницата до саканата локација

    ако е позната виртуелната адреса 01405FFFH за микропроцесорот 80386?

14. Објасни како е добиен адресен простор од 64TB кога микропроцесорот 80386 работи во виртуелен мод?

15. Колку изнесува капацитетот на меморијата на Пентиум 1 микропроцесор?

16. Колку изнесува капацитетот на меморијата на Пентиум Про микропроцесор?

17. Колкава е широчината на податочната магистрала за Пентиум 1 микропроцесор?

18. За што служи пинот AD на Пентиум микропроцесорот?

19. Колку изнесува работната фреквенција на Пентиум 1 микропроцесор?

20. Колку видови кеш-мемории има микропроцесорот Пентиум Про?

21. Објасни како се внесува и изнесува инструкција во инструкциски базен во микропроцесор Пентиум Про!

22. Како е организирана меморијата на микропроцесорот Пентиум Про?

23. Колкава е широчината на адресната и податочната магистрала за микропроцесор Пентиум-2?

24. Кои се двата најголеми новитети кај микропроцесорот Пентиум 2?

25. Опиши ги трите најпопуларни верзии на микропроцесорот Пентиум 2!

26. Опиши ја внатрешната структура на микропроцесор со две јадра?

27. Објасни каде се наоѓа инструкциската кеш меморија од прво ниво кај Пентиум 4 микропроцесорот?

28. Објасни ја функцијата на единицата за лоцирање и реименување?

29. Колку извршни единици имаат микропроцесорите Пентиум 3 и Пентиум 4?

30. Која е разликата меѓу микропроцесорите Пентиум Д и Интел Dual Core?

31. Објасни каква функција има чип сетот за работата на компјутерскиот систем?

32. Набројкои порти ги контролира чип сетот Интел 865G во Пентиум компјутерските системи!

Користена литература

- Structured Computer Organization-Andrew S. Tannenbaum-1998
- Fundamentals of Computer Organization and Design-Sivarama P. Dandamudi-2002
- The Intel Microprocessors 8086/8088, 80186/80188, 80286, 80386, 80486,Pentium and Pentium Pro Processor Architecture, Programming and Inter-facing-2003
- The Essentials of Computer Organization and Architecture-Linda Null and Julia Lobur-2003
- PIC in Practice-D.W.Smith-2002
- Intel 8085 processor –Data Sheet
- Intel 8086 processor –Data Sheet
- Intel 80386 processor –Data Sheet
- Intel 865G Chipset Family –Data Sheet
- Microchip PIC16f84-Data Sheet