Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма: Машинско инжењерство

Ниво студија: Основне академске студије

Модул: Информатика у инжењерству

Предмет: Архитектура рачунарских система

Број индекса: 710/2012

Милош Д. Николић

Основи организације и архитектуре процесора

завршни рад

Комисија за преглед и одбрану:

1. **Др Јасна Радуловић - ментор**
2. _____
3. _____

Датум одбране:_____

Оцена:_____

У оквиру овог завршног рада кандидат треба да:

Истакне улогу процесора у архитектури рачунарских система, начин рада и поступак обраде података као и меморисање обрађеног дела податка.

Такође, потребно је описати делове процесора, начине помоћу којег се добијају боље карактеристике рада и извршења података, локације и адресе података који се из меморије допремају процесору на извршење.

Од израде првих процесора до данас, направио се велики помак у равоју структуре процесора који је описан у другом делу рада, односно, описани су поједини процесори који су имали веома значајну улогу у развоју и архитектури рачунарских система.

Препоручена литература:

[1] **Радуловић Ј.**: *Архитектура рачунарских система*, предавања, Факултет инжењерских наука Крагујевац, 2013.

[2] **Стојчев М.**: *Архитектура и програмирање микрорачунарских система заснованих на фамилији процесора 80x86; I издање*, Универзитет у Нишу, Електронски факултет, 1999.

Крагујевац, ментор:

04.09.2013. **Др Јасна Радуловић**

Резиме: У раду се описује процесор као важан део рачунара и разматрају поступци који убрзавају начин извршења инструкција, односно чине рачунар ефикаснијим и бржим за рад. Такође, описани су и делови који чине склоп процесора уз помоћ којих процесор извршава команде које му се задају. Саставни делови процесора су: управљачка јединица и аритметичко–логичка јединица. Поред ова два главна дела, ту убрајамо регистре и кеш меморију, који убрзавају рад и дају боље карактеристике процесору.

Кључне речи: хардвер, софтвер, инструкција, операнди, податак, *Pipeline*, директан приступ меморији *DMA*, регистри, адресирање, механизам прекида.

Abstract: This paper describes the processor as an important part of the computer and examine the actions that accelerate the manner of execution of instructions and make your computer faster and more efficient to operate. Also described are the parts that make up part of the process by which the processor executes the commands that are given to him. The components of the central processing unit (CPU) are the control unit and the arithmetic logic unit. In addition to these two main parts, include the registers and cache, which accelerate the work and provide better performance of the processor.

Key words: hardware, softvare, instruction, operands, information, Pipeline, Direct Memory Access *DMA*, registers, addressing, mechanism breaks.

САДРЖАЈ:

1. УВОД

1.1. Архитектура рачунарских система

Под архитектуром рачунарског система подразумева се његово представљање помоћу основних функционалних јединица. Другим речима, термин **архитектура рачунара** се односи на атрибуте и понашање рачунара како их види програмер на најнижем софтверском нивоу, тј. на нивоу машинског језика. То су атрибути они који имају директни утицај на логичко извршење програма. У надлежности архитектуре рачунара су описи: скупа инструкција, формата инструкција, типова и формата података, регистара рачунара, адресног простора, свих меморијских локација, као и техника за адресирање меморије.

Архитектура рачунара, се још назива и архитектура скупа инструкција, *ISA (Instruction Set Architecture).*

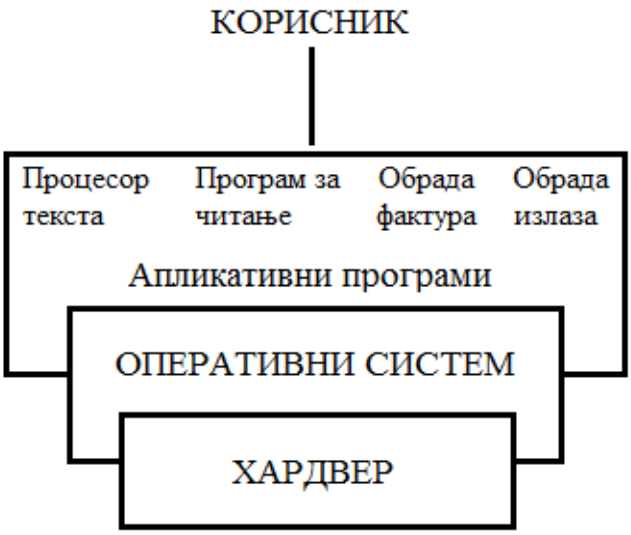
Материјализација архитектуре рачунара одговарајућим хардвером назива се имплементација рачунара. Архитектура и софтвер написан за ту архитектуру предвиђени су да опстану неколико деценија, док поједине имплементације имају животни век много краћи [1].

1.2. Хардвер и софтвер рачунара

Улазно–излазне јединице, оперативна меморија, процесор и екстерне меморије чине главне физичке делове рачунара који се једним именом називају хардвер *(eng. hardware).* Сам хардвер није довољан за рад рачунара. Да би рачунар могао да изврши неку обраду података, неопходно је дефинисати програм рада рачунара у коме ће прецизно бити задате све активности у оквиру обраде, како по питању садржаја, тако и по питању редоследа њиховог извођења. С тога програм представља низ појединачних инструкција или наредби који се чува у меморији рачунара. У рачунару постоји већи број различитих програма који се једним именом називају софтвер *(softvare).* Треба уочити да се избором програма мења начин обраде података и ако хардвер рачунара остаје неизмењен.

Програми се у зависности од порекла, могу сврстати у две категорије: системски и апликативни софтвер.

Системским софтвером, се обично називају програми који потичу од произвођача рачунара и испоручују се заједно са рачунаром. То су најчешће програми без којих рачунар не може да ради (на пример оперативни систем), као и неки услужни програми који корисници примењују независно од проблема којим се баве.



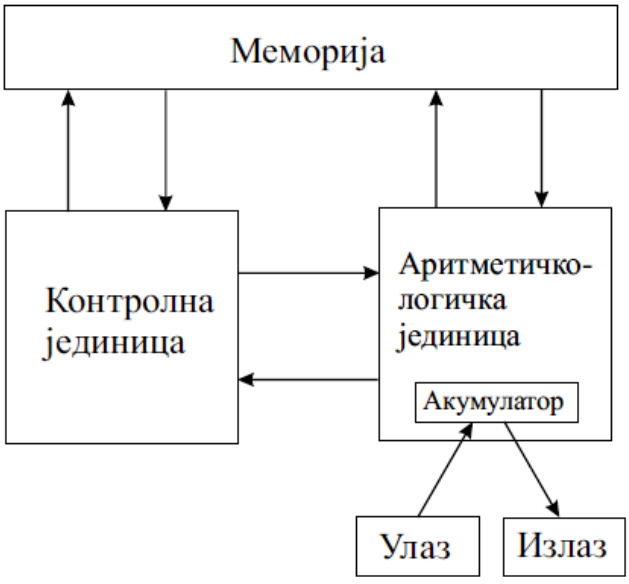
Слика 1. *Хардвер и софтвер*

Апликативни софтвер чине програми које пишу разне категорије корисника за потребе решавања својих проблема. Строгу границу између системског и апликативног софтвера понекад је тешко дефинисати. На пример, још увек не постоји сагласност око тога у коју категорију спада програм *Internet Explorer*, тј. да ли је он део оперативног система *Windows* или није [1, 6].

2. ОРГАНИЗАЦИЈА РАЧУНАРА

Рачунари су дигитални системи који служе за обраду података. На основу улазних података које задаје који представљају величине битне за решавање неког проблема, рачунар генерише излазне податке који представљају решење тог проблема. Иако подаци које корисник уноси, као и резултати који му се презентују могу бити у различитим форматима (текст, одмерци неког сигнала итд.), они у се конвертују у рачунар у бинарни облик, јер само тако могу да буду обрађивани. Бинарне речи које представљају податке су низови бинарних цифара 0 и 1 са дефинисаним значењем. То значи да се за сваку бинарну реч у систему зна шта она представља (податак, инструкцију, и сл.).

Обрада података заснива се на извршавању релативно малог броја операција дефинисаних над бинарним речима. Операције које се извршавају над једном бинарном речи називају **унарним**, за разлику од операција од операција које се обављају над двема бинарним речима које се зову **бинарним** операцијама. Операције се могу сврстати у четири класе: аритметичке, логичке, операције померања и преносе. У аритметичким операцијама бинарне речи се посматрају као бинарни бројеви (неозначени, у покретном зарезу, у комплементу двојке итд.). Аритметичке операције се реализују преко сабирања и одузимања. Логичке операције се изводе над појединачним разредима бинарних речи. Операције померања испод унарне дефинишу се као померања бинарне речи за одређен број разреда улево или удесно уз одговарајуће попуњавање празних разреда добијених услед померања. Операције преноса служе за пренос бинарних речи са једне локације на другу, при чему се реч која се преноси не мења. У рачунару, операције се реализују помоћу комбинационих и секвенцијалних мрежа као што су: регистри, меморије, декодери, мултиплексери итд.



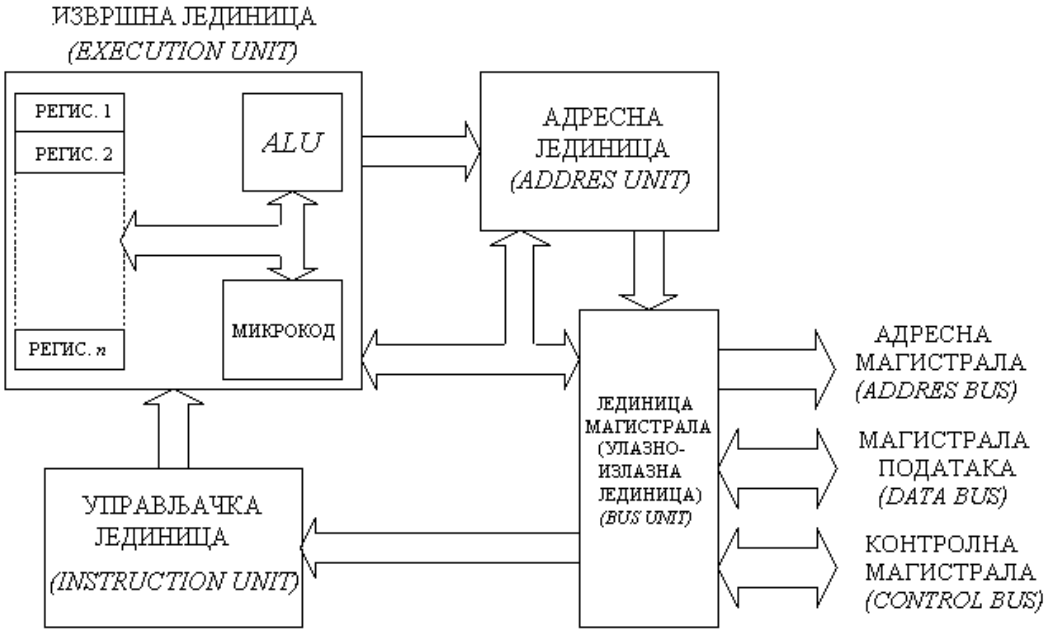
Слика 2. *Подсистем рачунара*

Организација рачунара се бави принципијелним остварењем функционалности рачунара, односно утврђивањем врста организационих компоненти рачунара, технологије израде, њихових особина и начина њиховог комбиновања. Као организационе компоненте рачунара јављају се: процесор (централни процесор, централна процесорска јединица), оперативна (главна) меморија, контролери, улазни и излазни уређаји, сабирница али и системски програми попут оперативног система или компајлера [1].

3. ОРГАНИЗАЦИЈА ПРОЦЕСОРА

Процесор извршава програм који је претходно смештен у оперативну меморију инструкцију по инструкцију и то оним редоследом којим су инструкције наведене у програму, све док не дође до краја програма. Овај поступак се може представити циклусима који садрже следећа четири корака:

- узимање инструкције из меморије,
- узимање операнада, уколико се то захтева у инструкцији,
- извршавање инсрукције,
- упис резултата у меморију или слање на неки излазни уређај ако се то у инструкцији захтева.



Слика 3. *Схема процесора*

Скуп инструкција представља једну од кључних особина архитектуре рачунара јер описује учинак рачунарског система, а у суштини га чини специфицан скуп операција које дати систем може да обави. Основни захтеви које скуп инструкција у рачунару треба да испуни су да треба да буде:

- *комплетна*. То значи да је коришћењем инструкција из овог скупа могуће написати програм на машинском језику за израчунавање произвољне функције, уз коришћење расположивог меморијског простора.
- *ефикасна*. Ово указује на то да функције које се често захтевају треба да се изврше брзо, што представља временску ефикасност и имплементирају што краћим низовима инструкција, што представља просторну ефикасност,

- *регуларна*. Ово подразумева да скуп инструкција треба да садржи очекиване опкодове и адресне начине рада, чиме себи обезбеђује манипулација над свим типовима података,
- *компатибилна*. То значи да скуп инструкција треба да буде усаглашен са раније дефинисаним скупом инструкција, тј. оних са архитектурама процесора које поседују,
- *симетрична*. Овај захтев подразумева да се исти скуп адресних начина рада може применити за приступ операндима у свим инструкцијама, итд.

Код већине рачунара, инструкције се могу класификовати у следеће три категорије:

- **инструкције за пренос података**. Помоћу њих се врши пренос података (копирање) из једне локације у другу, без измене садржаја бинарне информације,
- **инструкције за манипулисање подацима**. Ово су инструкције помоћу којих се обављају: аритметицке операције, логичке операције и операције померања. Поред аритметичких операција, логичке операције извршавају бинарне операције над низовима битова сачуваних у регистрима. Корисне су за манипулисање појединачним битовима, или групом битова која представља бинарно кодирану информацију. Логичке инструкције разматрају сваки бит операнада посебно и третирају га као [1]Булову променљиву. Већина рачунара у свом склопу инструкција поседује инструкције померања садржаја операнда, које се може обавити на више начина. Група инструкција померања садржи инструкције логичког померања, аритметичког померања и инструкције ротације,
- **Инструкције програмског управљања**. Ове инструкције мењају секвенцу програмског извршења. Ове инструкције су веома значајне, јер обезбеђују управљање током извршења програма, уз могућност гранања на разлиите програмске сегменте. Да би инструкција могла да се изврши неопходно је претходно обезбедити расположивост операнада битних за њено извршење. Стога, инструкција у свом формату мора да специфицира, осим операције коју процесор треба да обави и податке над којима се операција обавља.

Операнди су подаци у бинарном облику над у којима се врше жељене операције. На пример, бројеви које треба сабрати су операнди за операцију сабирања. Операнди се обично налазе у оперативној меморији (као и програми), мада се могу добити и од неке периферне јединице [1, 3].

[1] Булова алгебра се заснива на бинарним законима мишљења, где један исказ може бити истинит (тачан) или неистинит (нетачан), никада не може бити делимично тачан или делимично нетачан.

3.1 *CISC* и *RISC* архитектуре процесора

Реализовани скупови инструкација садрже више стотина различитих инструкација. Основна карактеристика већине инструкција јесте извршење операција над операндима од којих се један најчешће налази у акумулатору а други се позива из оперативне меморије. Рачунари са овако реализованим скупом инструкација се називају ***CISC*** *(od Complex Instruction Set Computers)* или рачунари са комплексни скупом инструкација.

Почетком 80–тих година при реализацији нових процесора тежило се повећању укупног броја инструкција и њиховом слагању. То је за последицу имало повећање потребног броја такт–интервала, а захтевало је и дужи такт од такта потребног за реализацију простих инструкација.

Анализа највише коришћених програма показала је да највећи део времена за извршавање је трошен на врло мали број релативно простих инструкација. Утврђено је следеће правило да преко 80% времена извршења било ког програма се троши на извршење инструкација чији број не пролази 20% укупног броја инструкција процесора.

Резулатат ових истраживања била је нова ***RISC*** *(od Reduced Instruction Set Computers)* архитектура. Идеја је да се смањи број инструкација и повећа ефикасност. Многим *CISC* микропроцесорима се замера да поседују могућности које се скоро никада не употребљавају, што значи да представљају вишак [3].

Принципи на којима се заснива градња *RISC* процесора су:

- Смањити број тактова за извршење једне инструкације
- Користити фиксни формат инструкције да би се упростило декодирање
- Смањити број инструкција помоћу којих се приступа основној меморији
- Смањити број кодова које процесор распознаје

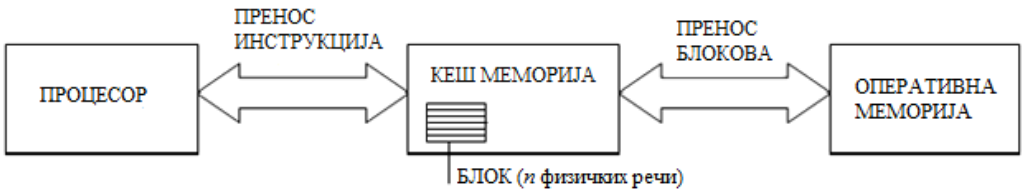
4. ИНТЕРНА МЕМОРИЈА ПРОЦЕСОРА

За рад процесора, као и рачунара неопходна је меморија пошто се у њој током рада смештају програми који се извршавају, као и инструкције који се тим програмима обрађују. Основна јединица за величину меморије је бајт.

4.1. Кеш меморија

Кеш меморија (*Cashe*) је нешто значајнија процесору управо због тога што је као и регостарска меморија саставни део процесора.

Кеш меморија је меморија малог капацитета реда свега неколико мегабајта (*MB*). У кеш меморију, се чувају подаци и инструкције из оперативне меморије које процесор тренутно користи. Она омогућује увећање брзине обраде, јер се у њој налазе текући подаци и текуће инструкције програма којима процесор приступа знатно брже, чиме се повећава продуктивност рада процесора, односно време извршења програма. Циљ ефикасног меморијског система је да ефективно време приступа процесора подацима буде врло блиско времену приступа кеш меморији.



Слика 4: Кеш меморија процесора

Кеш меморија се користи на следећи начин: Оперативна и кеш меморија су подељене јединице које се називају блокови. Блок (понекад, се назива и линија) представља скуп од *n* сукцесивних меморијских локација који се увек као целина преноси између оперативне и кеш меморије. То значи да се сви подаци или инструкције у неком блоку истовремено налазе или не налазе у кеш меморији. Јединица преноса између централног процесора и кеш меморије је физичка реч. Јединица преноса између кеш меморије и оперативне меморије је блок. Величина блока обично износи између 4 и 128 бајтова. Капацитет кеш меморије је у опсегу од 1 до 1024 *KB*. Број блокова оперативне меморије знатно је већи од броја блокова кеш меморије, тако да се у кеш меморији у истом тренутку налазе копије само малог броја блокова оперативне меморије [6, 7].

Када централни процесор генерише адресу меморијске локације, формира се управљачки сигнал за приступ кеш меморији. Уколико, се податак са траженом адресом налази у кеш меморији, он се преноси у процесор ради обраде или се замењује новом вредношћу из процесора која представља резултат обраде. Уколико у кеш меморији нема блокова са траженом адресом, активира се процедура којом се из кеш

меморије један блок шаље у оперативну меморију, а на његово место се из оперативне меморије позива тражени блок који се преноси у кеш меморију, истовремено се тражени податак преноси у оперативну меморију.

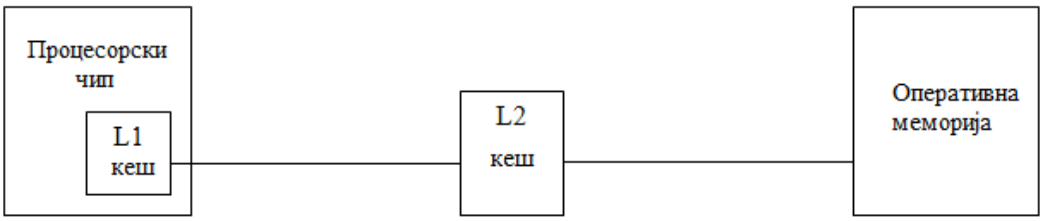
Појавом *Pentium* процесора, појавила су се и два нивоа кеш меморије.

Када су кеш меморије првобитно уведене, обичан систем имао је једну кеш меморију. Касније је коришћење више кеш меморија постала норма. Два аспекта овог проблема односе се на број нивоа кеш меморије и коришћење јединствене или подељене кеш меморије.

Кеш меморија првог нивоа (*L1* ниво) је смештена у само језгро микропроцесора. Ова веома брза меморија има релативно мали капацитет и подељна је на два блока, један служи за инструкције а други за податке. Она обично ради на истом такту као и сам процесор.

Кеш меморија другог нивоа (*L2* ниво) има знатно већи капацитет и код данашњих микропроцесора је такође смештена унутар самог процесорског чипа. Зависно од типа процесора, ова кеш меморија може да ради на пуном такту процесора или на нижем такту (обично половина учестаности такта процесора).

Увођењем кеш меморије у сам микопроцесор добило се даље повећање брзине рада рачунара, са тим да та брзина доста зависи од квалитета кеш контролера, односно од тога Колико добро она предвиђа следеће податке које ће бити потребни микропроцесору да би могао унапред да их припреми.



Слилка 5: *Кеш меморија другог нивоа*

Данас су заступљени процесори са кеш меморијом и трећег нивоа (*L3* ниво) који могу имати меморију и до 10 *MB*. Фреквенција *L3* кеша се може разликовати од фреквенције језгра процесора. Брзина трансфера из *L3* кеша ка језгрима износи 32 бајта по циклусу, односно 256–бита, колико је уједно и ширина магистрале [6, 8].

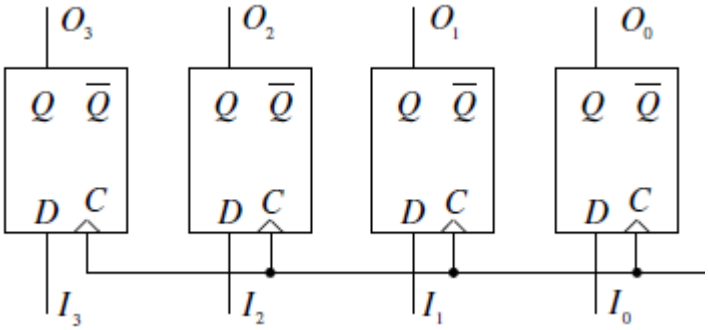
4.2. Регистри процесора

Регистар је дигитални систем који служи за привремено меморисање, односно памћење вишебитних информација и представља један од главних елемената процесора. Регистар се састоји од више [2]флип–флопова (*Flip–flop)* од којих сваки служи за смештање једног бита. Тако се регистар састављен од *n* флип флопова се користи за смештање бинарног податка од *n* бита. За реализацију регистра се најчешће користи *D* флип–флоп, односна флип–флоп са два стабилна стања. Скуп регистара представља део стазе података процесора и такође представља веома брзу меморијску јединицу малог капацитета за привремено смештање улазних и излазних података, као и појединих међрезултата.

Начин уноса података у регистар се може извести на два начина: стационарним и померачким путем уноса података.

4.2.1.Стационарни регистри

Стационарни регистри (регистри са паралелним пуњењем) предтавља веома ефикасан начин уноса податка у регистар јер се унос података врши у току једног такта рада процесора. Може се видети пример једног стационарног регистра са четири *D* флип флопа [1].



Слика 6. *Веза стационарног регистра*

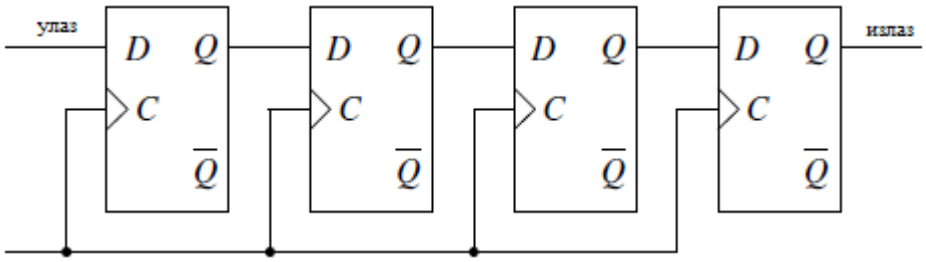
4.2.2. Померачки регистри

Померачки регистри за разлику од стационарних регистара имају лошију карактеристику зато што се унос података у померачком регистру врши у онолико такта рада процесора колико тај регистар има флип–флопа у свом склопу па је управо због тога потребно и више времена како би се унос података извршио.

Четворобитни податак се тако у регистар уписује што се по наиласку првог тактног импулса на излаз првог флип флопа прослеђује први наилазећи бит, тј. бит најмање тежине *(LSB – least significant beat)*, по наиласку другог такта импулса овај бит

[2] Флип–Флоп *(Flip-flop)* је бистабилно коло, који користи такозване такт сигнале, којима се може дефинисати промена стања у тачно одређеном тренутку. У пракси, се најчешће срећу тактована, тзв. синхрона секвенцијална кола, код којих се промене излаза дешавају синхроно са појавом такт импулса.

се прослеђује на излаз другог флип флопа, а наредни наилазећи бит се истовремено прослеђује на излаз првог. После трећег такта импулса *LSB* се прослеђује на излаз трећег флип–флопа, бит са излаз првог шаље се на излаз другог флип–флопа, наредни наилазећи бит поставља се на излаз првог. После четвртог такта импулса, истовремено са померањем садржаја првог, другог и трећег флип–флопа на излазе другог, трећег и четвртог, у први се уписује бит највеће тежине податка *(MSB – Most significant beat).* Тада је комплетна четворобитна информација смештена у регистар[1].



Слика 7. *Веза четворобитног померачког регистра*

Пример уноса четворобитног податка 1001 у померачком регистру приказан у табели:

Табела 1. *Унос четворобитног податка у померачком регистру*

Улазни податак	1001					
Првобитно стање		0000				
После 1. такт импулса			1000			
После 2. такт импулса				0100		
После 3. такт импулса					0010	
После 4. такт импулса						1001

4.3. Регистри према намени у процесору

Регистри који се користе у оквиру процесора у зависности од функција за које су намењени, могу се поделити у две групе и то:

- Корисничко видљиви регистри (адресибилни регистри)
- Управљачки и статусни регистри (интерни)

4.3.1. Корисничко видљиви регистри

Корисничко видљиви регисрти обезбеђују програмеру да на машинском или асемблерском језику минимизира обраћање главној меморији. Ови регистри се експлицитно специфицирају инструкцијом.

У овој групи регистара спадају:

- **Регистри опште намене** – користе се за чување података и адреса. Овим регистрима сам програмер додељује различите функције,

- **Регистри за податке** – користе се само за чување података, а не могу се користити код израчунавања адреса операнада,
- **Адресни регистри** – могу бити опште намене (за чување адреса) или могу бити намењени за реализацију адресног начина рада (показивачи сегмената, индексни регистри, показивачи магацина),
- **Маркер ригистар или регистар кода услова *CCR (Condition Code Registar)*** – делимично је видљив кориснику због чега спада у обе класе регистара (и корисничких и управљачких). У њему се чувају маркери које процесоров хардвер поставља као резултат извршења операције. Маркери обично чине део статусно/управљачке групе регистара.

4.3.2. Управљачки и статусни регистри

Постоји већи број *CPU*–ових регистара који се користе за управљање радом *CPU*–а. Највећи број ових регистара нису видљиви кориснику. Неки из ове групе регистара су видљиви преко машинских наредби које се извршавају у системском начину рада.

У групу управљачких регистара спадају:

- **програмски бројач *(PC)*** – садржи адресу наредбе која се припрема,
- **регистар инструкција *(IR)*** – чува наредбе која је последња прибављена из меморије,
- **меморијско адресни регистар *(MAR)*** – чува адресу по којој се обраћамо меморији,
- **прихватни регистар меморије *(MDR)*** – преко кога процесор и меморије размењују податке.

У групу статусних регистара спадају:

- **регистар кода услова *CCR (Condition Code Registar)*** – ово је регистар који обједињује више једнобитних регистара (индикатора, маркера) и чине га следећи индикатори:
  - *Sign* (знак) – садржи бит знака резултата задње аритметичке оерације,
  - *Zero* (нула) – поставља се када је резултат једнак 0,
  - *Carry* (пренос) – поставља се ако се извршењем операције јавио пренос (сабирање), или позајмљивање (одузимање) на месту бита највеће тежине,
  - *Equal* (једнако) – поставља се ако логичка компарација резултира једнакост,
  - *Overflow* (премештај) – указује на аритметички премештај.
- **системски регистар** или **системски бајт** – садржи информацију о томе да ли се процесор налази у супервизорском (извршавање програма оперативног система) или корисничком начину рада (извршавање корисничког програма), затим информацију о дозболи прекида *(IE)*, о маски прекида *(IM)*.

Понекад се све информације обједињују у u *PSW (Program Status Word)* склопу регистара. Код већине *CPU*– ова *PSW* чине регистри *CCR* и системски бајт [1, 4].

5. УПРАВЉАЧКА И АРИТМЕТИЧКО–ЛОГИЧКА ЈЕДИНИЦА

5.1. Управљачка јединица

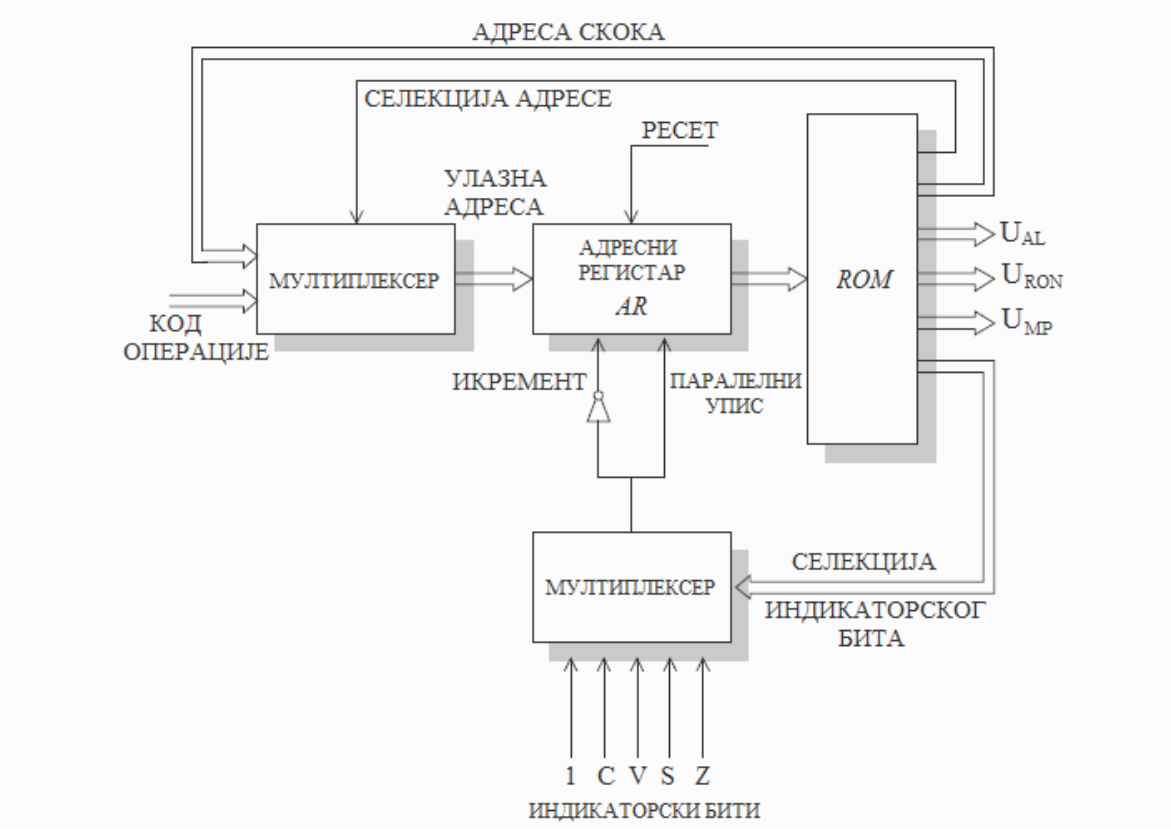
Управљачка јединица има задатак да правовремено и по одређеном редоследу генерише управљачке сигнале који одређују и синхронизују микрооперације свих делова микропроцесора и микрорачунарског система.

Управљачка јединица савремених микропроцесора реализује се на два начина:

- у облику микропрограмског аутомати,
- у облику сложене секвенцијалне мреже која је пројектована према захтевима миркопроцесора.

Управљачка јединица је синхрона секвенцијална мрежа која се описује скупом улазних и излазних сигнала, скупом стања, функцијом која генерише следећа стања и функцијом која генерише излазне сигнале.

Улазни сигнали управљачке јединице су сигнала ресет (*RESET*), код операције и бити индикаторског регистра као на слици. Излазни сигнали управљачке јединице су управљачки сигнали за остале јединице миркопорцесора и управљачки сигнали микрорачунарског система [6] .



Слика 8. *Схема управљачке јединице*

Улазни сигнал *RESET* уписује нулу у адресни регистар *AR*:

RESET: AR ⟵ 0.

Над адрсеним регистром *AR* дефинисане су две микрооперације, упис и повећање садржаја за 1 (инкремент), Који су одређени управљачким сигналима:

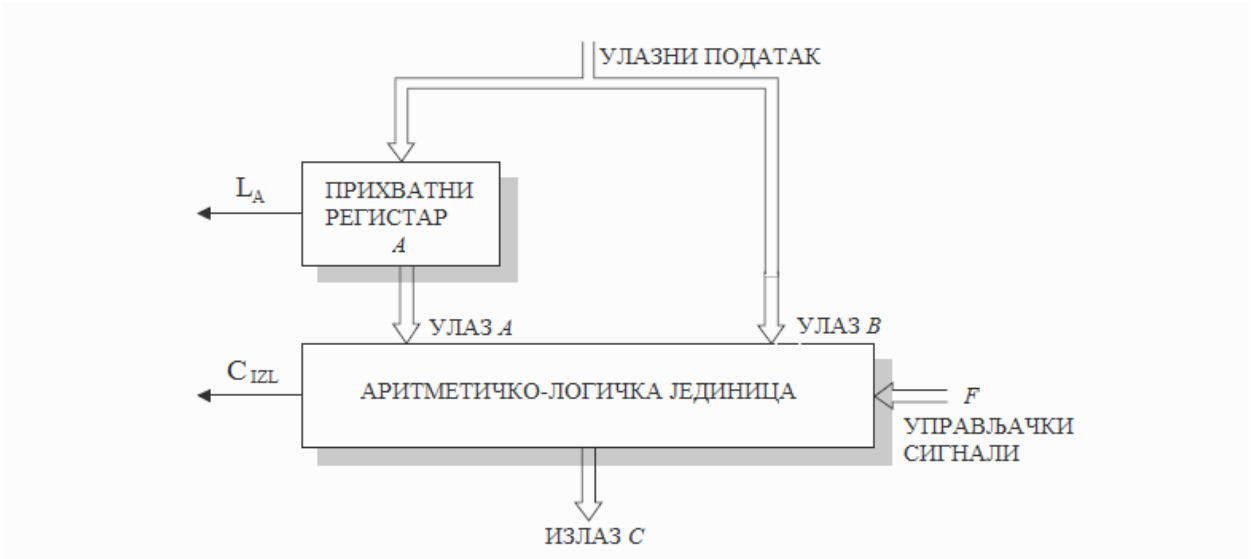
- инкремент: *AR* ⟵ Улазна адреса,
- повећање садржаја: *AR* ⟵ *AR* + 1.

Стање управљачке јединице је одређено адресом управљачког *ROM*–а која је смештена у адресном регистру *AR*. Следећа адреса се може добити на два начина: повећањем садржаја *AR* за 1 или уписом неке друге адресе. Уписом кода операције у *AR* започиње извршење новог микропрограма. Једном када започне извршење микропрограма, следећа адреса се добија *ROM*–а и може бити било која адреса у *ROM*– у, чиме се обавља програмски скок у микропрограму.

Начин добијања следеће адресе, зависи од индикаторског бита који се изабере мултиплексером: Ако је изабрани бит једнак 1, онда се врши следеће упис адресе, ако је 0 онда се следећа адреса добија повећањем садржаја за 1. Један од улаза мултиплексера је везан за логичку јединицу како би себи обезбедио безусловни микропрограмски скок [6].

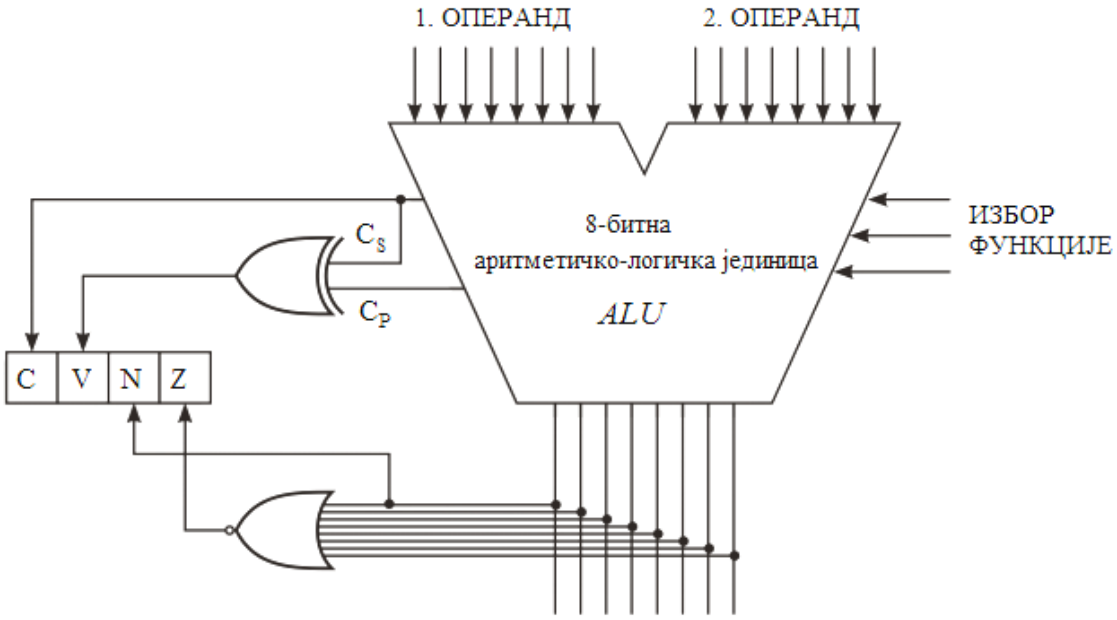
5.2. Аритметичко–логичка јединица

Аритметичко–логичка јединица (*ALU*) је комбинациона мрежа чија Булова функција, која опусује зависност излазних сигнала од улазних. Може да се бира скупом управљачких сигнала. ALU јединица има два скупа линија за улазне податке, један скуп линија за излазне податке, скуп линија за управљачке сигнале и линију за излазни сигнал преноса c_{izl}, слика.



Слика 9. *Унос података у аритметичко–логичку јединицу*

Двоструким стрелицама на слици означени су скупови линија за пренос сигнала сличне. Ради смањења броја улазних линија аритметичко–логичке јединице, улазни подаци за улазе *A* и *B* могу себи преносити заједничким линијама као што је приказано на слици. Тада је неопходно реализовати прихватни регистар *A* који привремено прихвата податке који се доводи на улаз аритметичкој–логичкој јединици. Регистар има паралелан улаз и паралелан излаз.

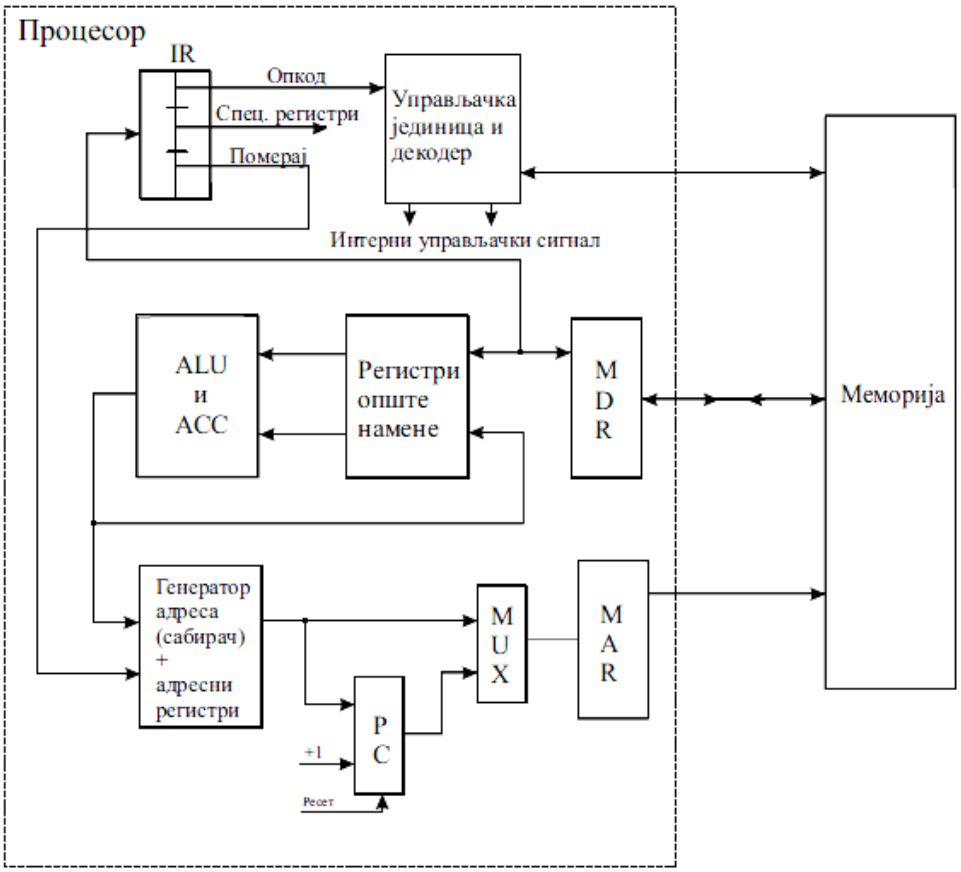


Слика 10. Схема аритметичко–логичке јединице

P је бит парности и налази се у регистру услова код микропроцесора ***Intel 8080*** за испитивање конкретног преноса података. Већина микропроцесора нема бит *P* у регистру услова, будући да је функција генерисања и испитивање парности пренета у надлежност програмибилне јединице У/И.

I је бит прекида. Обично се не употребљава у аритметичким операцијама. Бит *I* се поставља (*I* = 1) у случају прекида. Бит прекида се још назива и прекидна маска, јер се у случају да је *I* = 1 ако сам се појави захтев за прекидом нижег приоритета, прекид се неће догодити.

Неки микропроцесори имају један или више регистара који су по функцији издвојени од осталих, а називају се акумулатори. Поред тога што се употребљавају за привремено складиштење једног операнда, учествују они при извођењу аритметичких и логичких операција на подацима. И резултати аритметичких и логичких операција изведених у аритметичко–логичкој јединици складиште се у акумулатору. обично се акумулатор поставља на један од улаза у аритметичко–логичку јединицу и у комбинацији са привременим регистрима раздваја улаз аритметичко–логичке јединице од излаза. Раздвајање потребно је код микропроцесора стандардне архитектуре са Једном интерном сабирницом, Због утицаја излаза аритметичко–логичке јединице на улаз [6].



Слика 11. *CPU са регистрима опште намене*

Програм се налази у меморији, извршење програма почиње када се *PC* постави на вредност која указује на адресу прве инструкције у програму. Садржај *PC*–а се преноси у *MAR* и генерише се управљачки сигнал *Read* (читање) који се предаје меморији. Након одређеног времена, које одговара времену приступа меморији, чита се из меморије адресна реч, у овом случају то је прва инструкција у програму и смешта у регистар *MDR*. Затим се садржај *MDR*–а преноси у *IR*, песле чега је инструкција спремна за декодирање и извршење. Када је инструкцијом спрецификирано да се операција обави од стране *ALU*–а неопходно је да се прибаве операнди. Ако се операнд налази у меморији (може бити и у неком од регистара *R0 – Rn*) он треба да се прибави на тај начин што се његова адреса шаље у *MAR* и иницира циклус *Read*. и тада се операнд из меморије и смешта у *MDR*, а затим преноси из *MDR*–а на улаз *ALU*–а. На овај начин се може прибавити један или више *ALU*– ових операнада. Када су операнди спремни (приступни на улазима *ALU*–а) издаје се команда *ALU*-у да обави операцију. Резултат ове операције се може сместити у *ACC*, неки од регистара *R0 – Rn* или у меморију. Када се смешта у меморију он се прво уписује у *MDR*. Адресна локација где се резултат смешта се предаје у *MAR* и иницира циклус *Write* (упис). У току извршења инструкције, садржај *PC*–а се инкрементира како би показивао на инструкцију у програму коју треба извршити као наредну. На овај начин извршење наредне инструкције може да почне одмах након извршења текуће [1].

6. АДРЕСНОСТ ПРОЦЕСОРА

У предходном поглављу описан је пример и дата је слика *CPU*–а са регистрима опште намене, сав ток извршења скуп инстукција односно податка се неби извршио без локације и знања адресе где се жељена инструкција налази и како би се могла позвати и извршити одређена операција над том инструкцијом.

Сви подаци у меморији рачунара имају своју адресу како би при позиву процесора имали тачну локацију податка који се треба прибавити и треба обрадити. Податак може бити смештен у оперативној меморији, кешу или у регистрима процесора. Управо преко адресе се податак налази и шаље процесору како би се извршио. Због тога је адресност веома важан део процесора и самог рачунара [4].

6.1. Формат инструкције

Упис бинарног податка у меморију, се обавља у следећим корацима:

- процесор адресира жељену меморијску локацију постављањем њене адресе на адресну магистралу,
- процесор поставља податак који треба да се упише на магистралу података,
- процесор шаље команду меморији да обави упис податка.

Поступак читања бинарног податка из меморије одвија се на следећи начин:

- процесор адресира меморијску локацију из које жели да прочита податак постављањем њене адресе на адресну магистралу,
- процесор шаље команду меморији да на магистралу постави адресирани податак,
- процесор преузима податак са магистрале података.

Да би дохватио неку инструкцију, процесор мора да има информацију о томе на којој локацији у меморији се она налази. На почетку рада, адреса прве инструкције програма се хардверски уноси у регистар процесора у коме се чува адреса наредне инструкције. Пошто се овај регистар обично реализује као бројач а реч је о *PC* регистру, адреса сваке следеће инстукције програма добија се аутоматским инкрементирањем бројача (уколико није наредба скока).

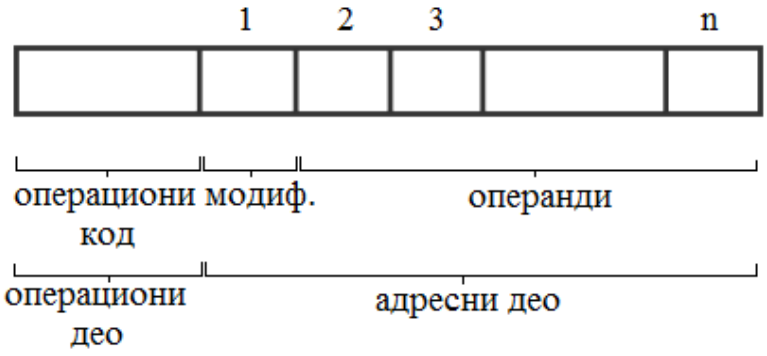
На основу познате адресе, из меморије се допрема одговарајућа инструкција и смешта у регистар за инструкције, након чега следи њено извршавање.

Свака инструкција се састоји из два дела: операционог дела (који дефининише операцију) и адресног дела (који конкретно специфицира податке, тј. операнде).

Адресни начин рада одређује где се прецизно налази жељени податак, тј. где је његова ефективна адреса *EA*. Другим речима, спецификација адресног начина рада, која је садржана у оквиру инструкције одређује како процесор треба да користи битове инструкције и остале информације да би израчунао ефективну адресу. Због тога, се у формату инстукције у адресном делу додаје још једно поље, тзв. модификатор. Модификатор уобичајено садржи информацију којом, се описује начин адресирања и

указује о додатним условима који се односе на приступ подацима или на начин извршења операција.

Операциони код је обима један до два бајта и одређује тип операције (сабирање, множење, копирање и др.) коју обавља процесор.



Слика 12. *Базични формат инструкције*

Адресни део садржи инфомацију о:

- адресама операнада над којима се операција извршава,
- адреси где треба да се смести резултат и
- адреси наредне инструкције коју треба извршити након што се обави текућа.

Бинарни низови којима се кодирају поједине врсте информација у инструкцији смештају се у одговарајућим деловима које зовемо поља, изглед инстструкције зове се формат инструкције.

Четвороадресни формат садржи све поменуте адресе, значи укупно четири (две адресе операнада, једна за памћење резултата и једна која указује одакле треба прибавити наредну инструкцију). Овакви формати се веома ретко срећу код реалних машина. Највећи број процесора изводи се као једноадресни, двоадресни или троадресни формат инструкција.

До троадресних инструкција се долази укидањем поља које садржи адресу наредне инструкције, при чему се адреса наредне инструкције добија преко имплицитно посебног регистра, тзв. програмског бројача *PC*- а. Ове инструкције захтевају релативно дуги формат и обично их срећемо код 32–битних *RISC* процесора.

Код двоадресног регистра се укида поље за смештање адресе резултата а резултат се памти на месту једног од операнада, чиме се уништава садржај једног операнда. Овај начин адресирања се веома често користи код 8, 16 и 32–битних микропроцесора типа *CISC*.

Код једноадресног формата место једног операнда се имплицитно подразумева у неком регистру, нпр. у акумулатору. Резултат се такође смешта у акумулатор. И на крају, укидањем свих адресних поља добија се нулаадресни формат инструкције, код кога се имплицитно подразумева да су оба операнда на врху стека (магацина). Резултат, се такође памти на врху стека [1, 4].

7. НАЧИНИ АДРЕСИРАЊА ПРОЦЕСОРА

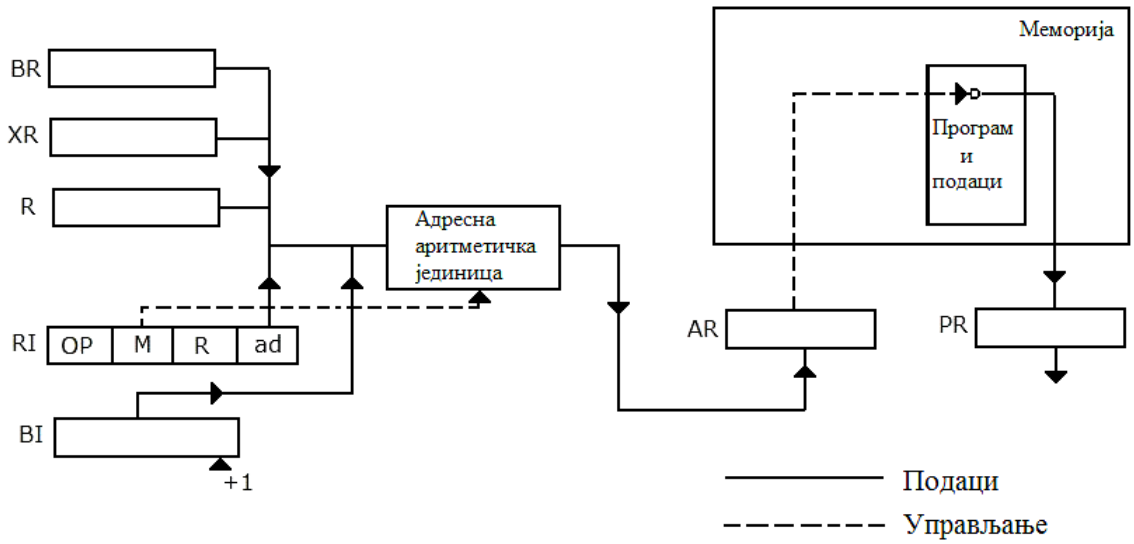
Сваки податак који се чува у некој локацији оперативне меморије може се идентификовати задавањем те адресе локације. Инструкцијама програма дефинишу се операције над подацима који се налазе у оперативној меморији или у неком регистру централног процесора. Због тога су у инструкцији садржане информације о адресама операнада или појединим компонентама адресе. На основу тих информација процесор налази само операнде.

Код свих савремених рачунара адресни део инструкције најчешће не представља стварну адресу, по којој се налазе подаци у меморији, већ само једну компоненту за формирање те адресе. Адреса по којој се врши стварно обраћање меморији, тј. физичка адреса податка у оперативној меморији, формира се на основу више компонената и назива се ефективна адреса или извршна адреса.

За формирање ефективне адресе користе се посебни адресни системи у процесору. Поступак одређивања ефективне адресе на основу адресних информација назива се начин адресирања или метод адресирања.

7.1. Формат ефективне адресе

Битовима у пољу модификатора M дефинише се начин формирања ефективне адресе у адресној аритметичкој јединици. Управљачким сигналима који се генеришу на основу садржаја поља M дозвољава се или забрањује подацима из различитих регистара процесора да учествују у формирању ефективне адресе.



Слика 13. *Формирање извршне адресе*

Метод адресирања који се у некој инструкцији користи за формирање ефективне адресе спецификује се у инструкцији најчешће у посебном пољу модификатора *M* које обично садржи 2 или 3 бинарне позиције. Како у формирању ефективне адресе операнада често може учествовати и неки од регистара централног процесора, то је потребно задати у инструкцији и адреси тог регистра.
Тако инструкција треба да има следећи општи формат: *OP, M, R, ad*, где је *OP* – симболичка ознака операције, *M* – модификатор, *R* – адреса или ознака регистра процесора, *ad* – адресни део инструкције[4, 5].

7.2. Апсолутно адресирање

Апсолутно адресирање је такав начин адресирања код кога адреса у инструкцији специфицира локацију операнда без коришћења било које додатне информације. Тај тип ефективне адресе, се зове апсолутна адреса. Разликујемо више начина апсолутног адресирања и на:

- **непосредно адресирање** – је најједноставнији облик апсолутног адресирања. У овом случају адресни део инструкције специфицира вредност операнда, тј. операнд је константа и саставни је део инструкције.
- **директно адресирање** – је такав начин адресирања код кога је у одговарајуће поље опренда смештена адреса меморијске локације која одговара податку (операнду) коме се приступа, тј. адресни део инструкције сам представља адресу на основу које се врши обраћање меморији, због чега се ова адреса зове директна адреса.
- **имплицитно адресирање** – је адресирање код кога операнд није специфициран у адресном делу наредбе, већ се његово место подразумева и зависи од опкод поља. Типично, овакав начин адресирања користимо код постављања вредности маркера регистра [1].

7.3. Индиректно адресирање

Индиректно адресирање је такав начин адресирања код кога адреса у инструкцији не специфицира меморијску локацију операнда, већ локацију у којој се налази адреса операнда. Разликујемо следећа два начина индиректног адресирања:

- **индиректно адресирање преко меморије** – је адресирање код кога се адресом специфицира меморијска локација у којој се чува адреса операнда. За разлику од директног адресирања код кога је за добијање вредности операнда потребна једна операција прибављања, код индиректног су потребне две.
- **регистарско индиректно адресирање** – је адресирање где адресно поље инструкције указује на регистар у коме се чува адреса операнда [1].

7.4. Релативно адресирање

Релативно адресирање – је такав начин адресирања код кога поље операнда садржи релативну адресу (*D – displacement, offset*) или размештај, а комплетна адреса операнда формира се од стране *CPU*-а. Инструкцијом се, такође, експлицитно или имплицитно идентификују остале меморијске локације које садрже додатну адресну информацију. Ефективна адреса операнда *EA*, представља неку функцију *f (R0,R1,...,Rn,D).* У највећем броју случајева адресном операнаду, из скупа адресних регистара опште намене, придружује се јединствени адресни регистар *Ri*, након тога ефективна адреса, се израчунава додавањем *D* садржају *Ri*, тј. $EA = Ri + D$

Разликујемо следеће начине релативног адресирања:

- ***PC* relativno** – је адресирање где је регистар коме се имплицитно обраћамо програмски бројач (*PC).* То значи да се текућа адреса инструкције сабира са размештајем и на тај начин формира текућа адреса *EA*. Код ове операције размештај се третира као број у двојичном комплементу. Као резултат ЕА представља релативни размештај у односу на адресу инструкције, тј.
  $EA = PC + D,$
  С обзиром да је садржај РС–а делује као показивач на инструкцију која се текућа извршава, вредност размештаја указује на растојање између операнда и саме инсрукције.
- **базно–регистарско адресирање** – је начин адресирања код кога регистар коме се обраћамо *R*, чува меморијску адресу, адресно поље садржи размештај *D* (обично је то неозначена целобројна вредност) у односу на меморијску адресу. Обраћање овом регистру може бити експлицитно или имплицитно
  $EA = R + D,$
- **индексно адресирање** – је адесирање код кога адресно поље *A*, саржи адресу главне меморије, а у регистру *R*, коме се обраћамо, смештен је позитиван размештај у односу на ту адресу. Без обзира на разлику у интерпретацији у односу на базно-регистарско адресирање и овде се ефективна адреса израчунава као $EA = Ri + D.$
- **аутоиндексирање** – је адресирање којим се обезбеђује ефикасан механизам за обављање итеративних операција. Појављује се код архитектура код којих се, с обзиром на често коришћење индексног регистра за обављање итеративних операција, уграђује онај део циклуса инструкције који аутоматски врши инкрементирање или декрементирање индексног регистра. Резултантни адресно-модификујући процес се зове аутоиндексирање, тако се, у зависности од архитектуре, могу уочити разне модификације, као што су: предекрементирање (садржај специфицираног адресног регистра ће се прво декрементирати, затим следи извршење инструкције), постинкрементирање (прво се изврши инструкција, након тога обави аутоматско инкрементирање), затим преинкрементирање и постдекрементирање.
- **адресирање магацина** – је адресирање које се користи за приступање магацину преко специјалног регистра, тзв. показивача магацина. Магацин представља линеарно поље локација, тј. резервисани број локација главне меморије. Ставци

магацина се може приступати само са једног краја који се зове врх магацина, *TOS – top of stack*. Операција упис у магацин, *Push*, смешта нову ставку на врх магацина, док операција читања, *Pop*, избавља ставку са врха магацина. Операцијама *Pus/Pop* мења се положај врха магацина за одговарајућу дужину операнда који се смешта/избавља у/из магацина. Приступ магацину регулише се наменским регистром, тзв. показивач магацина, *SP – stack pointer*. У *SP* се чува адреса задњег операнда који је смештен у магацин. Адреса на коју показује *SP* аутоматски се подешава након сваке *Push* и *Pop* операције, тако да *SP* увек показује на нови врх магацина. Операција *Push X* прво декрементира *SP*, након тога смешта *X* у локацију коју указује на *SP*. Операција *Push* повећава обим магацина за једну ставку. Насупрот томе, *Pop Y* смањује обим магацина смештајући елемент са врха магацина на локацију, Затим инкрементира показивач магацина за једну ставку. На основу претходне анализе можемо закључити да се учинак аутоиндексирања могу успешно користити код адресирања магацина. Велики број процесора има уграђен специјалан аутоиндексни адресни регистар *SP* који се користи као имплицитни показивач магацина.

- **комбиновано (мултикомпонентно) адресирање** – представља комбинацију претходно поменутих начина адресирања, при чему се ефективна адреса добија као збир већег броја компонената [1].

8. МЕХАНИЗАМ ПРЕКИДА ПРОЦЕСОРА

Механизам прекида врло је важан концепт у рачунарским системима који је уведен како би се избегло да процесор троши време чекајући на спољашње догађаје. Значај овог механизма биће илустрован на конкретном примеру решавања проблема који следи.

Проблем: Познато је да су периферије знатно спорије од процесора. Претпоставимо да процесор треба да пошаље већу количину података штампачу да их одштампа. Због ограничених могућности штампача, подаци се морају слати у више делова. Када процесор један део пошаље на штампање, он мора да сачека да штампач заврши свој посао, тј. да одштампа приспеле податке, како би послао следећи део. Чекање на периферију (док она заврши свој посао) представља изгубљено време за процесор, јер тада је он беспослен. С обзиром да је процесорско време драгоцено, било је неопходно наћи неко прихватљиво решење за превазилажење овог проблема.

Решење проблема: Проблем је решен тако што је уведен механизам прекида који процесору допушта да извршава друге активности за време док периферија обавља свој посао. Механизам прекида обезбеђује ефикаснији рад рачунара са периферијама тако што за разлику од принципа чекања на периферију да заврши свој посао, уводи нови принцип опслуживања периферије на њен захтев. То значи да када процесор пошаље податке периферији, одмах наставља са радом даље не чекајући да обави свој посао периферија. У тренутку када периферија заврши посао, она обавештава процесор о томе тада процесор предузима мере за слање наредне количине података на периферију [4].

Поступак опслуживања прекида одвија се на следећи начин:

- када периферија постане спремна за пријем података од процесора, она то сигнализира процесору слањем захтева за прекидом. Захтев садржи код прекида који одговара периферији која га је послала,
- по пријему захтева, процесор завршава извршавање текуће инструкције само на кратко прекида извршавање текућег програма како би опслужио приспели захтев. Сваки процесор има скуп прекида које је у стању да опслужи. Сваки прекид унапред је дефинисана тзв. прекидна рутина која представља подпрограм који треба да се изврши у случају приспећа захтева за тим прекидом. Прекидне рутине су смештене у меморији на одређеним адресама. Све адресе прекидних рутина смештене су у табелу прекида (*Interrupt Pointer Table*) која се, такође, налази у меморији,
- процесор опслужује прекид тако што на основу кода прекида из приспелог захтева проналази у табели прекида адресу одговарајуће прекидне рутине и извршава рутину. У прекидној рутини се опслужује одговарајућа периферија,
- по завршетку прекидне рутине, процесор се враћа тамо где је стао у програму који је извршавао у тренутку наиласка захтева за прекидом и наставља са радом.

Да би описани механизам прекида могао успешно да функционише, неопходно је обезбедити да се при повратку из прекидне рутине процесор нађе и потпуно истим условима у којима је био када је почео опслуживање прекида. Ти услови, се називају *контекстом процесора*. То значи да када процесор прихвати захтев за прекидом, пре него што пређе на његово опслуживање, мора да сачува контекст у коме је извршавао текући програм. Контекст обично чине тренутни садржаји појединих регистара процесора. Након завршетка прекидне рутине, сачувани контекст процесора се рестаурира (постављају се вредности запамћене у регистре) и процесор зна одакле треба да настави извршавање започетог програма и какав је био статус након последње операције коју је *ALU* извршила [4, 5].

Памћење контекста процесора је неопходно зато што се прекидна рутина извршава као и сваки други програм, што значи да се том приликом користе исти процесорски регистри, па обично долази до измене њиховог садржаја.

Прекиди се могу класификовати на различите начине. Једна од подела је на:

- спољашње или екстерне прекиде
- унутрашње или интерне прекиде

Спољашњи прекиди долазе од периферија, док унутрашњи прекиди могу бити последица извршавања инструкције прекида, или последица неке нерегуларности у извршавању текуће инструкције. Прекидима се придружују приоритети који указују на њихову важност. Предност при опслуживању имају прекиди са већим приоритетом.

Интерни прекиди су вишег приоритета од екстерних, тако да ако у току екстерног опслуживања прекида пристигне захтев за интерним прекидом, процесор прекида опслуживање екстерног прекида и прелази на интерно опслуживање прекида. Међутим, ако у току опслуживања интерног прекида стигне захтев за екстерним прекидом, процесор ће га прихватити тек када заврши започети посао [2].

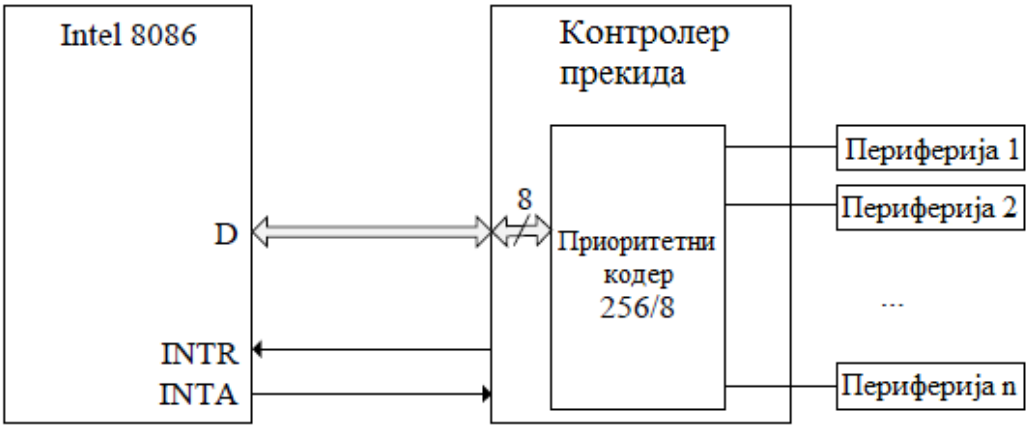
Даља разрада и реализација механизма прекида биће анализирана на *Intel 8086* процесору.

Процесор *Intel 8086* има могућност опслуживања како екстерних, тако и интерних прекида. За пријем захтева за екстерним прекидима предвиђена су два пина процесора: *INTR* (*Interrupt Request*) и *NMI* (*Non Masked Interrupt*).

Линија *INTR* је обично повезана са програмабилним контролером прекида *Intel 8259*. Улога контролера прекида је да прихвата захтеве за прекидима које упућују периферије повезане са њим и да их прослеђује процесору. У датом тренутку контролер може да добије један или више захтева за прекидом. Уколико је добио више захтева, контролер помоћу приоритетног [3]кодера одређује који од приспелих захтева има највиши приоритет и ако је тај захтев вишег приоритета од захтева који процесор тренутно опслужује, контролер активира линију *INTR*.

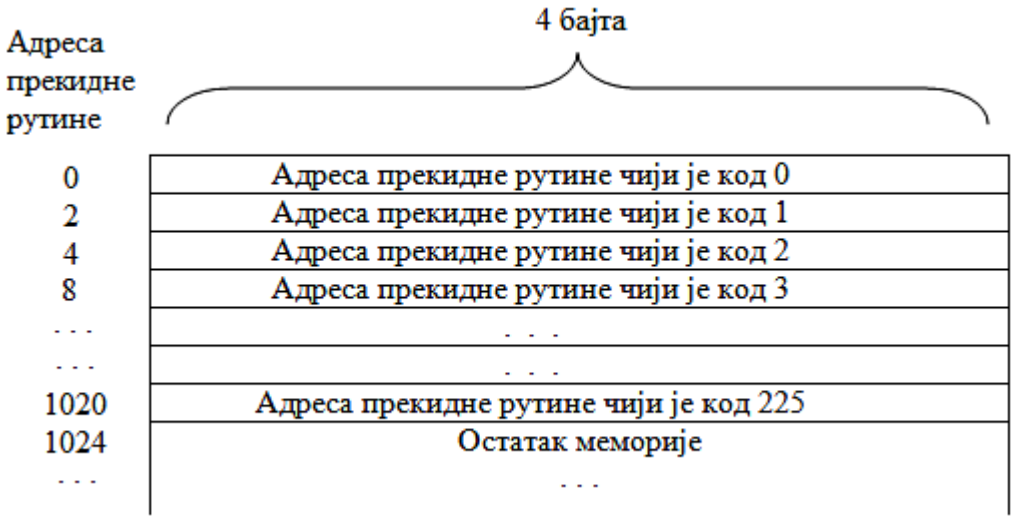
[3] Кодери су комбинационе мреже помоћу којих се изводи шифровање (кодовање) опште познатих симбола (словних, симбола, симбола децималног бројног система, итд.). Овде се ради о мрежи која има супротну функцију од декодерске мреже, кодер може имати највише 2^n улаза, $m \leq 2^n$, ако се кодовање изводи са *n* бита на излазу.

Приоритетни кодер је врста кодера која допушта истовремено довођење сигнала на више улаза, при чему се на излазу појављује кодирана информација која одговара активном улазу са највишим приоритетом [3].



Слика 14. *Повезивање процесора Intel 8086 са контролером прекида*

Адреса прекидне рутине добија се из табеле прекида (*Interrupt Pointer Table*) на основу добијеног кода прекида. Процесор *Intel 8086* може да опслужи 256 врста прекида. Стога табела прекида има 256 улаза (за сваку врсту прекида по један). За памћење појединачних адреса прекидних рутина користе се 4 бајта, па је величина табеле прекида 1 *KB* (256 x 4 *B* = 1024 *B*). Табела прекида, се налази увек у првом килобајту меморије као што је приказано на слици.



Слика 15. *Табела прекида*

Пре него што пређе на извршавање прекидне рутине, процесор мора да сачува контекст у коме је извршавао текући програм. То је неопходно како би након завршетка прекидне рутине процесор могао да се врати тамо где је стао и наставио са извшавањем програма који је прекинут.

У оквиру контекста процесора, треба сачувати:

- садржај *PSW* регистра, јер се у њему налази тренутни статус процесора,
- садржај *IP* регистра, јер се у њему налази адреса наредне инструкције програма.

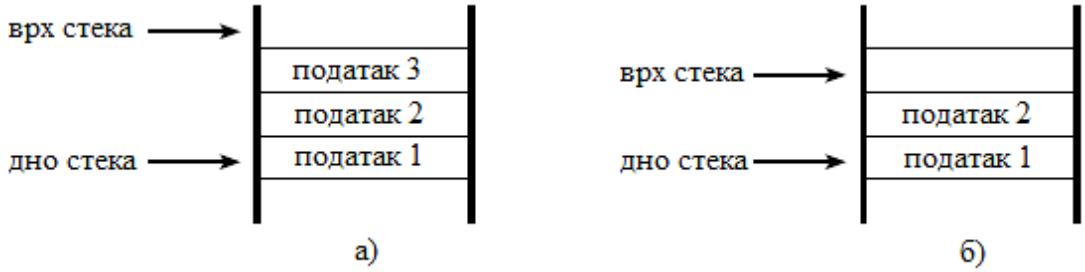
Памћење контекста се врши аутоматски, непосредно пре преласка на извршавање прекидне рутине. Такође, по завршетку прекидне рутине, поменути регистри, се аутоматски реиницијализују на старе, сачуване вредности. У пракси, осим *PSW* и *IP* регистара, често је неопходно сачувати и садржаје појединих регистара опште намене. Наиме, у зависности од врста прекида, током извршавања прекидне рутине, поједини регистри опште намене могу променити свој садржај. Да би процесор могао да се врати у првобитно стање, програмер треба да утврди који регистри опште намене могу да промене садржај, затим да експлицитно, софтверским путем, захтева памћење њиховог садржаја. Након завршетка прекидне рутине, програмер мора сам да реиницијализује ове регистре опште намене, такође, софтверским путем, тј. да испише део кода који обавља њихову реинициjализацију.

Контекст процесора се чува у посебном делу меморије који се назива стек (*stack*). Штек подржава *LIFO (last in, first out)* дисциплину приступа, што значи да се са штека прво узимају подаци који су последњи унети у њега. Део меморије предвиђен за штек ограничен је показивачима на меморијске локације које представљају дно стека и врх стека. Приступ стеку, у циљу уписа, било ради узимања неког податка, увек себи врши преко врха стека. Може бити стек имплементиран на различите начине. Једна од имплементација приказана је на сликама 18 и 19.

Када је стек празан, врх стека и дно стека указују на исту меморијску локацију (слика 18 (а)). При стављању првог податка на стек, податак се уписује у меморијску локацију на коју указује врх стека, затим се врх стека помера на прву наредну локацију (слика 18 (б)). Показивач на дно стека се не помера (увек указује на исту локацију). У овој имплементацији врх стека увек указује на наредну празну локацију на коју треба уписати следећи податак.

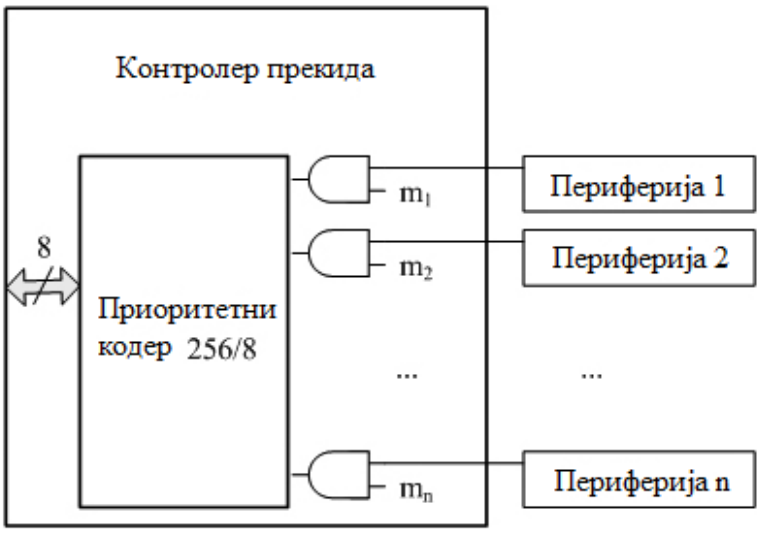
Слика 16. *Упис података у стек*

Читање податка са стека обавља се тако што се најпре провери да ли је стек празан (оба показивача указују на исту локацију). Ако стек није празан (слика 19 (а)), показивач на врх стека се помери на претходну локацију, затим се из ње преузима податак (слика 19 (б)). Врх стека остаје на празној локацији из које је узет податак, тако да постоје услови за евентуални будући упис новог податка.



Слика 17. *Читање податка из стека*

Процесор *Intel 8086* има могућност маскирања прекида који долазе по *INTR* линији. Маскирање се обавља тако што процесор шаље контролеру прекида одговарајућу команду која садржи маску са одређеним броја бита m_i, као што је приказано на слици 20. Сваки бит m_i маскира захтев за прекидом упућен од једне периферије тако што се заједно са захтевом доводи на улаз *I* кола. Ако је вредност бита маске 1, захтев за прекидом се прослеђује на излаз *I* кола, а ако је бит маска 0, захтев за прекидом се игнорише. Контролер прекида разматра само један захтеве који су прошли кроз *I* кола, тј. дошли до приоритетног кодера и прослеђује их процесору [2, 3].



Слика 18. *Маскирање прекида*

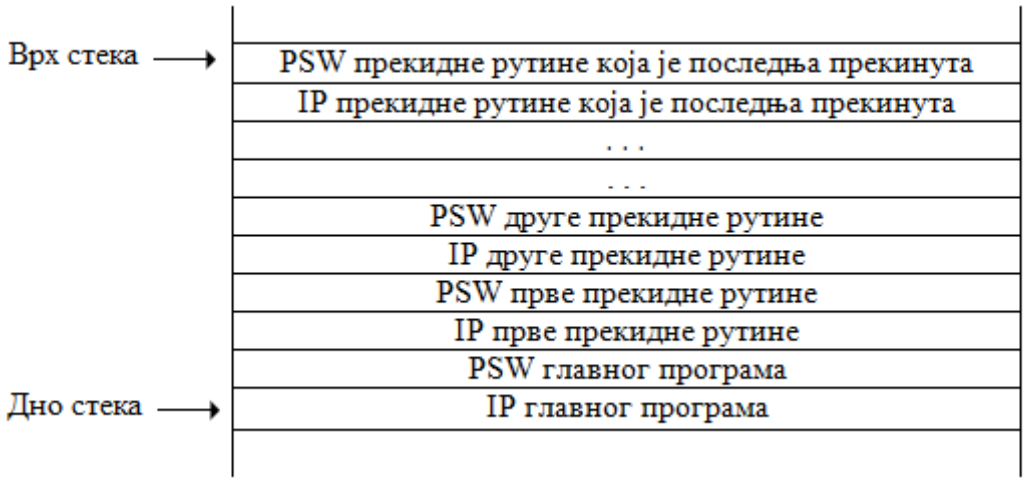
Осим спољашњих прекида који долазе по *INTR* линији, процесор *Intel 8086* може да опслужи и спољашње прекиде који долазе по *NMI* линији, као и унутрашње

прекиде. Поступак њиховог опслуживања је исти као ивећ описани поступак за опслуживање прекида који стижу по *INTR* линији.

Прекиди који стижу по *NMI* линији називају се немаскирајућим прекидима. Овај назив потице од чињенице да не постоји могућност њиховог неприхватања. То значи да се они не могу игнорисати ни постављањем маске, ни постављањем флега *fl* у *PSW* регистру. Разлог за овакав третман је у томе што се ради о прекидима који могу да имају катастрофалне последице по процесор. На пример, они настају када дође до губитка напајања, грешке приликом трансфера на магистрали, грешке у раду меморије итд. Прекиди који долазе по *NMI* линији су вишег приоритета од било ког прекида који стиже по *INTR* линији.

Унутрашњи прекиди настају приликом извршавања инструкције интерног прекида, тј. асемблерксе инструкције *INT*. Ова инструкција садржи код прекида на основу кога процесор активира извршавање одговарајуће прекидне рутине. Интерни прекиди су вишег приоритета од екстерних прекида.

Због велике разноврсности могућих захтева за прекидом, уведена је строга дисциплина у њиховом опслуживању која се заснива на приоритетима. Као што је речено, највиши приоритет имају интерни прекиди, затим спољашњи који долазе по *NMI* линији и на спољашњи крају који стижу по *INTR* линији. Ова хијерархија у приоритету је битна уколико током извршавања једне прекидне рутине дође нови захтев за прекидом. Ако је приспели захтев нижег приоритета, мора да сачака да би био опслужен. Међутим, ако је вишег приоритета, процесор прекида извршавање текуће прекидне рутине и прелази на прекидну рутину која одговара приспелом захтеву. Када је заврши, процесор се враћа на рутину коју је прекинуо и наставља са њеним извршавањем. Овакав начин опслуживања прекида може да иде и у "дубину", тј. да постоји више прекидних рутина чије је извршавање у току. Да би сви приспели захтеви били коректно опслужени, при преласку на нову прекидну рутину неопходно је сачувати текући контекст процесора. Он се чува на стеку као што је приказано на слици.



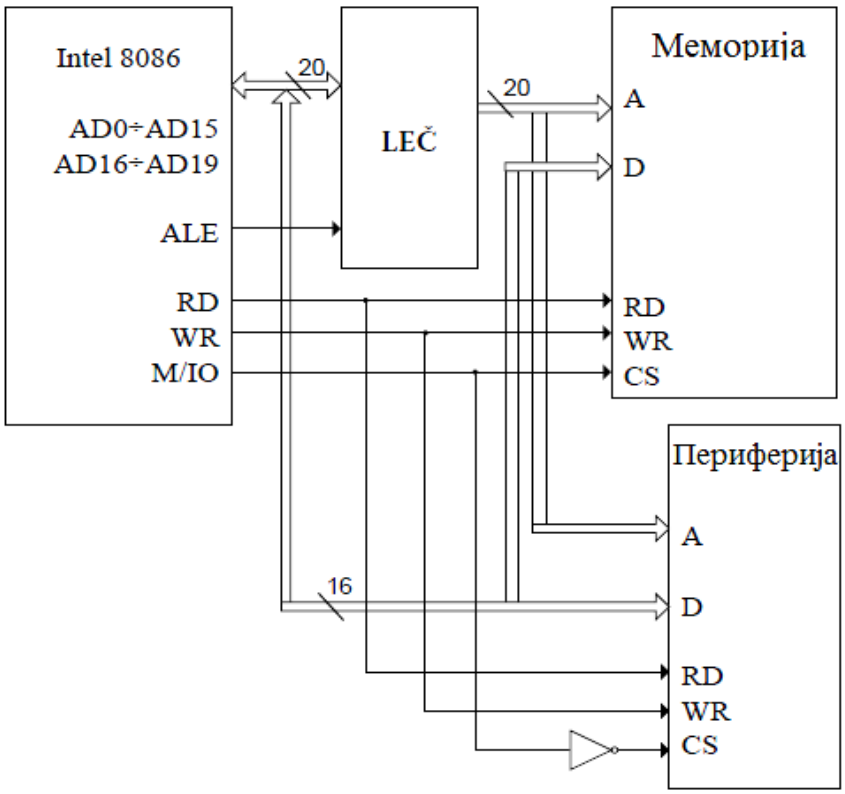
Слика 19. *Чување контекста процесора на стеку*

У овом примеру, процесор је извршавао главни програм када је стигао захтев за прекидом. Да би опслужио прекид, процесор је прекинуо главни програм, текући контекст ставио на стек и прешао на извршавање прекидне рутине. Током извршавања прекидне рутине, дошао је захтев за прекидом вишег приоритета, па је и извршавање прекидне рутине морало бити заустављено. Пре преласка на следећу прекидну рутину, на стеку је сачуван контекст који одговара тренутном стању у коме је прекидна рутина прекинута. Сличан поступак понавља се за све наредне прекиде вишег приоритета. Када се прекид највишег приоритета опслужи, са стека се узима запамћени контекст који одговара претходној рутини и наставља се са њеним извршавањем. Када се опслуже сви прекиди, са стека се реинијализује контекст који одговара главном програму и процесор наставља његово извршавање [2, 3].

9. РАЗМЕНА ПОДАТКА СА ОКРУЖЕЊЕМ

Потреба за разменом бинарних информација између процесора и окружења (меморије, периферија) је врло честа, јер се подаци и програми налазе на једној локацији (у меморији и на периферијама), а њихова обрада се обавља на другој локацији (у процесору). На слици приказан је начин повезивања процесора *Intel 8086* са меморијом и периферијама у циљу остваривања међусобне комуникације.

Процесор *Intel 8086* размењује информације са остатком система путем 20–битне магистрале. Магистрала је једним делом мултиплексирана зато што се њених 16 бита *(АД0–АД15)* користи за пренос и адреса и података, док се преостала 4 бита *(А16–А19)* користе само за адресе. То значи да се магисталом података преносе 16–битни подаци, док је адресна магистрала 20–битна и омогућава адресирање меморије капацитета *1 MB*. Проблем који овде настаје је у томе што није могуће да се на истим пиновима истовремено појаве и адреса и податак, њихова истовремена расположивост је неоходна приликом уписа и читања садржаја. Овај проблем је решен употребом леча (*latch*). Леч је секвенцијално коло слично регистру, с том разликом сто се након уписа податка у леч, податак аутоматски појављује на његовом излазу. Ова особина леча је искоришћена за привремено памћење адресе [2, 3].



Слика 20. *Размена података са окружењем*

Процесор размењује податке са меморијом двосмерно, тј. може да уписује податак у меморију или да га из ње чита. Поступак уписа бинарног податка у меморију одвија се на следећи начин:

- процесор адресира жељену меморијску локацију постављањем њене адресе на адресну магистралу *(AD0–A19),*
- процесор активира *ALE* сигнал којим се адреса уписује у леч и прослеђује на његов излаз. Пошто је излаз леча директно повезан са адресним улазом меморије, а адреса, се прослеђује до меморије,
- на магистралу података *(AD0–AD15)* процесор поставља податак који треба да буде уписан у меморију и он се директно прослеђује до меморијског улаза за податке *D,*
- активирањем сигнала *WR*, који је директно повезан са улазом за сигнал уписа у меморију *WR*, процесор задаје команду меморији да изврши упис податка са *D* на локацију чија је адреса на улазу меморије.

Читање бинарног податка из меморије, се обавља у следећим корацима:

- процесор адресира меморијску локацију из које жели да прочита податак постављањем њене адресе на адресну магистралу *(AD0–A19),*
- процесор активира *ALE* сигнал којим се адреса уписује у леч и прослеђује на његов излаз. Пошто је излаз леча директно повезан са адресним улазом меморије, адреса се прослеђује до меморије,
- активирањем сигнала *RD*, који је директно повезан са улазом за сигнал читања из *RD* меморије, процесор задаје команду меморији да са адресе на њеном улазу прочита податак и постави га на магистралу података *D,*
- због директне везе *D* излаза меморије и магистрале података *(AD0–AD15),* процесор преузма податак.

Осим са меморијом, процесор на сличан начин размењује податке и са периферијма. Као и меморијске ћелије, свака периферија има своју адресу. Приликом обраћања периферији, процесор поставља њену адресу на адресну магистралу. Одговор који добија потиче од адресиране периферије.

Проблем настаје због чињенице да се адресе меморијских локација поклапају са адресама периферија. Уколико не би постојала никаква додатна информација о томе коме процесор жели да се обрати, десило би се да му истовремено одговоре и меморија и периферија, што би довело до грешке у систему због судара података који долазе са разлицитих страна на исту магистралу података. Овај проблем је решен увођењем сигнала *M/IO* на основу кога се тачно зна чију адресу је процесор поставио на магистралу. Ако *M/IO* сигнала има вредност 1, процесор се обраћа меморији, ако Је 0 обраћа се периферији. Сигнал *M/IO* генерише процесор и не води на *CS (Chip Select)* улазе меморије и периферија. *CS* улази омогућавају рад чипа. То значи да ако сигнал *M/IO* има вредност 1, он се прослеђује на *CS* улаз меморије и њој може да се приступа. Истовремено, због инвертора, на *CS* улаз периферија долази вредност 0 и приступ периферијама није могућ. Обрнуто, ако се сигнал *M/IO* постави на 0, рад са меморијом је блокиран, може се приступати периферијама јер је на њиховом *CS* улазу вредност *1* (Због инвертора) [2, 3].

10. Intel ПРОЦЕСОРИ

Од развоја првих процесора до данас важи закон који је заснован на брзину процесора и број транзистора у његовом склопу а то је Муров закон

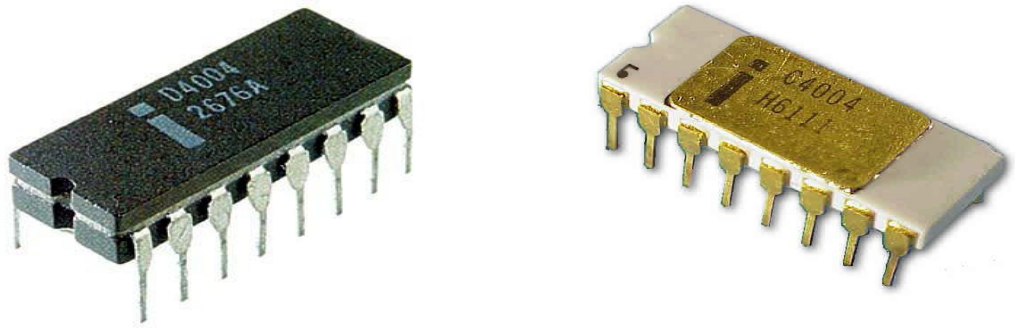
Муров закон гласи да се снага рачунара удвостручује приближно сваких 18 – 24 месеца. Добио је име по Гордону Муру (*Gordon Moore*) једном од оснивача *Intel*–а, на чијим се предвиђањима и заснива.

Производња микропроцесора прати ову законитост од 1965. године, када је чланак објављен у Електроникс магазину (*Electronics*). У истраживању од 1973. годину за почетну тачку у развоју микропроцесора (*Intel 8008* – процесор из 1972. године), до 2008. године прошло је 35 година тј. 420 месеци, тако да се број транзистора на чипу удвостручио 23 пута тј. број транзистора се повећао 8.388.608 пута [7].

10. 1. Почеци развоја Intel фамилије процесора

Седамдесетих година прошлог века, на тржишту су се појавили први процесори. У току своје кратке историје (последњих четрдесетак година), просецори су доживели огроман технолошки напредак. Њихове перформансе у погледу брзине су побољшане више од хиљаду пута, архитектура је знатно унапређена, могућности значајно проширене. Овако интензиван развој био је праћен често беспоштедним надметањем фирми које су производиле процесоре за превласт на светском тржишту.

У новембру 1971. године произведен је први **Intel**–ов процесор са ознаком **4004** који је био упакован у само један чип. За његову израду је коришћена тада доступна технологија 10 *μm*. Радни такт овог процесора је био од 0.4 до 0.8 *MHz*, имао је 4-битну магистралну података и 640 *B* спољашње меморије.



Слика 21. *Први Intel-ов процесор са ознаком 4004 из 1971. године*

Већ у априлу 1972. године произведен је процесор **Intel 8008** са 8–битном магистралом и 16 *KB* спољашње меморије. Радни такт није битно промењен.

У први комерцијално раположив рачунар *Altiar* уграђен је **Intel**-ов процесор

следеће генерације са ознаком **8080**. Само за неколико месеци продато је на хиљаде комада ових рачунара, што им је донело прво место на листи продаје. Овај процесор је радио на такту од 2 *MHz* и имао је 64 *KB* меморије.

Значајни помак се десио 1978. године када је произведен први *16*–битни процесор **Intel 8086** који је *IBM* уградио у свој први персонални рачунар *PC/XT*. У то време се користила 3 *μм* технологија. Овај процесор, се производио са тактом у верзијама 2, 5, 8 и 10 *MHz*, имао је 20–битну адресну магистралу која је адресирала 1 *MB* меморије као и концепт директног приступа меморији [7, 8].

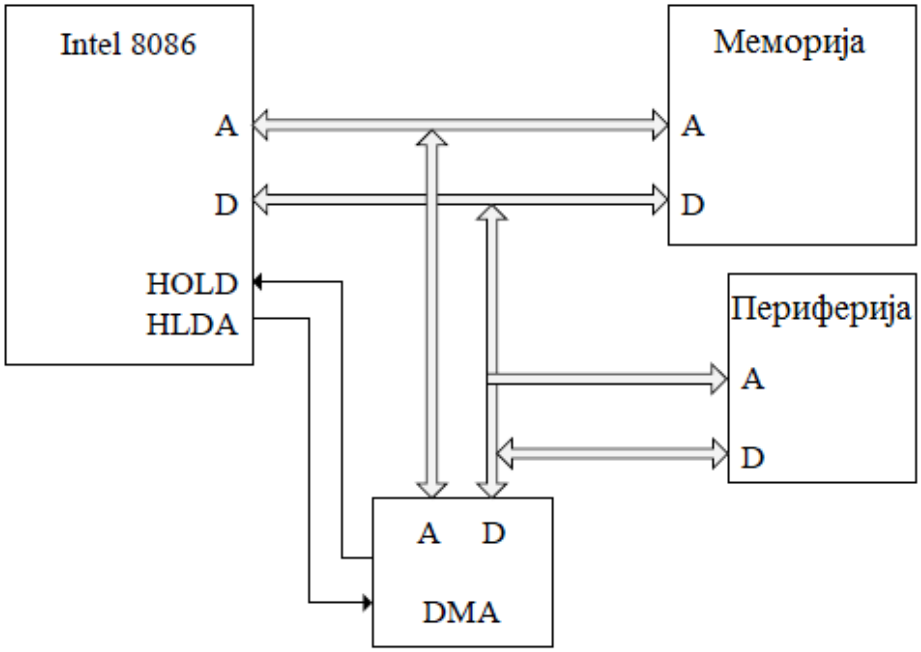
10.1.1. Концепт *DMA*

DMA (*Direct Memory Access*) је концепт присутан у свим модерним рачунарима који омогућава директан пренос података између периферије и меморије, без посредовања процесора. Захваљујући њему, процесор као најскупљи и најоптерећенији ресурс у рачунарском систему је ослобођен многих непотребних послова. Наиме, у систему који нема *DMA*, да би подаци били пребачени од извора до одредишта (на пример неке периферије до меморије) процесор би морао да копира део по део података са извора и да их прослеђује на одредиште, при чему, током целог тог процеса не би био расположив ни за какву другу активност. Отежавајућа околност је и та још што су периферне јединице много спорије од оперативне меморије, због чега, се губи додатно време. Увођењем *DMA* концепта, процесор се ослобађа овог посла и за време трансфера података од извора до одредишта и може да обавља друге послове. Ипак, треба имати у виду да у тим пословима не може да користи магистралу којом, се обавља *DMA* пренос.

При *DMA* преносу важну улогу има *DMA* контролер (он преузима послове око које преноса процесор има у систему без *DMA*). *DMA* пренос процесор иницира слањем одговарајуће команде *DMA* контролеру. Након тога, одговорност за потпуну одговорност за пренос преузима *DMA* контролер (генерисање адреса, трансфер података). По завршетку преноса, *DMA* контролер обавештава процесор да је пренос завршен и да је магистрала слободна [2].

DMA пренос се примењује у следећим случајевима:

- Када је потребно пренети велику количину података између периферије и меморије, па би посредовање процесора знатно успорило пренос,
- Када је периферија релативно брза (као пример на хард диск).



Слика 22. *Повезивање Intel 8086 са DMA контролером*

Прегледности ради, на слици није приказан леч, али, се подразумева да он постоји унутар процесора. Сваки пренос података између и периферија и меморије представља *DMA* циклус. Он се одвија на следећи начин:

- *DMA* контролер захтева од процесора да му омогући приступ магистрали тако што активира сигнал *HOLD,*
- Када процесор прими сигнал *HOLD*, завршава текући циклус на магистрали и активира сигнал *HLDA*, чиме предаје магистралу *DMA* контролеру,
- Док *DMA* контролер користи магистралу за пренос података, процесор не сме да јој приступа (нема уписа и читања из меморије), али може да настави са обрадом податак који се тренутно у њему налазе,
- Када заврши *DMA* циклус, *DMA* контролер деактивира сигнал *HOLD*, чиме ослобађа магистралу и предаје је процесору.

Интересантно је да се у том тренутку није уочавала будућа потреба за већом меморијом [3, 4].

Године *1982.* настао је 16 битни процесор **Intel 80286**, кога је *IBM* уградио у *Advanced Technology* свој персонални рачунар *PC/AT* који је за 6 година производње доживео 15 милиона инсталација, чиме је превазишао све раније рекорде. То је био први прави процесор икада јер је увео концепт заштићеног мода *(Protected Mode)*, у зависности који је омогућавао да више програма раде истовремено, независно један од другог *(multitasking)*. Ова могућност је касније нашла примену у оперативним системима као на пример *Windows.* Радни такт овог процесора је ишао и до 20 *MHz* са 16 *MB* RAM меморије.

Први 32–битни процесор **Intel 80386** је произведен 1985.године и представља велики технолошки напредак у Intel–у. Развијена по цела фамилија ових процесора. Радни такт је био до 33 *MHz*, а последња верзија је могла да адресира 4 *GB RAM*–а.

Поред тога, процесору осим што процесира, треба и нешто где ће одложити оне податке са којима тренутно ради, да му буду при руци. Као што ми, када решавамо неку сложену математичку операцију користимо дигитрон, па на парчету папира привремено запишемо добијени резултат, како би га поново могли касније искористити у даљем рачунању, тако и процесору треба меморија која ће му служити као парче папира на којем привремено бележи успешне вредности потребне у даљем раду. Та меморија се зове *cashe* меморија.

Почетком *1989*. године настао је *32*–битни процесор **Intel 80486** чији је радни такт у последњим верзијама достизао 100 *MHz*. Био је дупло бржи од свог претходника јер је имао побољшану архитетктуру. То је први процесор са интегрисаном кеш меморијом величине 8 *KB* у коју су се стављали следеће инструкције и подаци, тако да процесор није морао да приступа спољашњој меморији. Такође овај тип процесора се сматра првим представником који садржи имплеметарну *Pipeline* технику процесирања.

Такође овај тип процесора се сматра првим представником који садржи имплеметарну *Pipeline* технику процесирања [7].

10.1.2. Концепт *Pipeline*

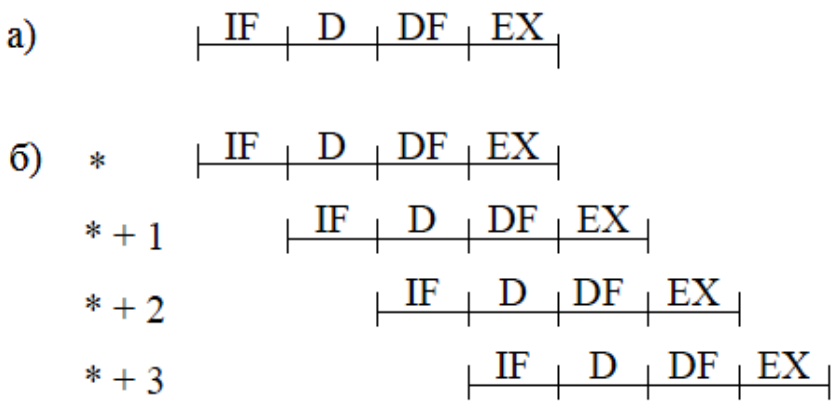
Pipeline представља концепт који је уведен са циљем да се убрза процес извршавања програма. Концепт се заснива на коришћењу конкурентности приликом извршавања инструкција. Претпоставимо да се одвија процес извршавања инструкције у четири фазе:

- IF (*Instruction Fatch*): дохватање инструкције из меморије и њено смештање у регистар за инструкције,
- D (*Decode*): декодирање операционог кода инструкције,
- DF (*Data Fatch*): дохватање податка (операнада) било из меморије или из одговарајућих регистара,
- EX (*Execute*): извршавање инструкције.

Идеја у ***Pipeline*** концепту је да се омогући да се у истом тренутку различите инструкције налазе у различитим фазама извршавања и да се тако оствари знатна уштеда у времену извршавања програма. То значи, на пример, да наредна инструкција може да се дохвата из меморије док траје декодирање текуће инструкције. С обзиром да су ове две активности међусобно независне, нема никаквих препрека да се обављају истовремено.

Pipeline машине управо покушавају да обезбеде сталну заузетост по свим фазама, тј. да у сваком такту постоји по једна инструкција у свакој од фаза извршавања.

На слици 5. (б) приказан је концепт *Pipeline* на примеру извршавања 4 узастопне инструкције од којих се прва налази на некој адреси.



Слика 23. *Извршавање инструкције: а) обично; б) у Pipeline-у*

Као што се види, у првом такту дохвата се прва инструкција из меморије, у другом, се она декодира и истовремено дохвата друга инструкција из меморије. У трећем такту дохватају се операнди за прву инструкцију, друга инструкција се декодира и истовремено се трећа инструкција дохвата из меморије. У четвртом такту је остварено највеће убрзање пошто у свим фазама постоји по једна инструкција која се обрађује: Прва инструкција је у фази извршења, друга у фази дохватања операнада, трећа у фази декодовања и четврта у фази допремања из меморије. У петом такту извршава се друга инструкција, дохватају се операнди за трећу, четврта декодира. У шестом такту трећа инструкција се извршава, за четврту се допремају операнди. На крају, у седмом такту, извршава се четврта инструкција. Дакле, све су извршене четири инструкције. У периоду од 7 тактова, док би без примењеног *pipeline* концепта за њихово извршење било потребно 16 тактова, по 4 за сваку инструкцију.

Убрзање које се може остварити применом концепта *pipeline* зависи од степена броја (фазе) или дубине *pipeline*-а тј. фаза у извршавању инструкције у датом примеру број степени је 4, па је максимално убрзање 4 пута под условом да је машина добро подешена. Наравно, постизање убрзања има своју цену. Она се огледа у већој сложености и већој цени процесора са оваквим механизмом извршавања инструкција [3, 4].

Године 1993. отпочиње производњу нових генерација **Intel**-ових процесора под новим именом **Pentium**. До промене имена (уместо 586) дошло је због правних проблема.

Pentium I је имао радни такт од 233 *MHz*, два независна кеша (за наредбе и податке) а могао је да извршава две инструкције у току једног такта. Такође, два оваква процесора могла су да раде заједно у систему. Најпознатији у овој фамилији били су **Pentium Pro** i **Pentium MMX**.

Pentium II представља неуспели покушај обједињавања најбољих особина претходника. Имао је такт до 450 *MHz*, два кеша различитих брзина и капацитета *(*32 *KB* L1 кеша и до 512 *KB L2* кеша који је знатно спорији од *L1*, али много бржи од оперативне меморије). Кеш меморија другог нивоа (*L2* ниво) има знатно већи

капацитет и код данашњих микропроцесора је такође смештена унутар самог процесорског чипа. Зависно од типа процесора, ова кеш меморија може да ради на пуном такту процесора или на нижем такту (обично половина учестаности такта процесора).

Pentium III је імао проширен скуп MMX-ових инструкција (за 70 нових инструкција) чиме је побољшано процесирање 3Д анимација. Радни такт овог процесора је ишао и до 1 *GHz*. Да би побољшао сигурност *online* трансакција, Intel је одлучио на сваки процесор овог типа угради серијски број чипа (*processor serial number – PSN*) који је могао да се прочита преко мреже или интернета. Пошто су корисници сматралида је то напад на њихову приватност, Intel је био принуђен допусти искључење *PSN*-а у *BIOS*-у.

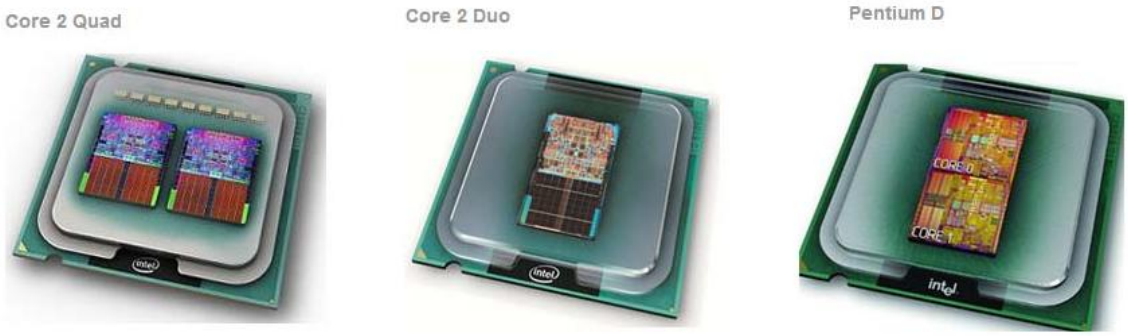
Pentium IV је имао нову, *NetBurst* архитектуру која је омогућавала повећавање брзине у будућности. Ова архитектура обухватала је четири нове технологије: *Hyper Pipelined Technlogy* (могућност повећања брзине процесора проширењем *pipeline*-а, тако да се може извршавати више инструкција истовремено), *Rapid Execution Engine* (као последица проширења *pipeline*-а, уведене су две аритметичко- логичке јединице које раде дупло брже), урађене су неопходне поправке на кешу и уведена системска магистрала од 400 *MHz*. Радни такт овог процесора је ишао до 3.6 *GHz* [4].

Процесор ради по принципу једне ствари у једно време. Значи, не може радити две или више ствари у исто време . То процесор ради једно иза другог, брзина којом се обавља је дефинисана радним тактом , тј. што је радни такт бржи, то он брже одрађује. Такт се мери у милионима или милијардама операцијама у секунди, тј. *MHz* или *GHz*.

Како се та брзина не може повећавати у недоглед, произвођачи су се сетили да ставе уместо једног процесора, два процесора у исто кућиште, тј. два језгра *(Dual Core)*, тако би оба радила упоредно или поједностављено, једно језгро од 3 *GHz* би радила посао као два од 1.5 *GHz*. Први *Intel*–ов процесор са два језгра је ***Intel Pentium D*** настао 2005.

Core 2 Duo процесор је који има два језгра (два процесора у једном кућишту) и за сваки процесор има посебну меморију (*cashe*), што значи сваки ради са својом меморијом.*Core2Duo* има два језгра која су састављена и комуницирају међусобно.

Pentium Dual Core рађен је на истој платформи, тј. има два језгра али која деле заједничку меморију (*cashe*), која је уједно и мања. Зато су *Dual Core* нешто спорији у мултитаскинг операцијама, ипак по свим тестирањима не више од 30 %. Цена варира, али је око 30 % јефтинији. У принципу, оба једнако ефикасно раде, троше енергију и снагу. За рачунар то не представља битну разлику [8].



Слика 24. *Кућишта процесора са више језгра*

Pentium Dual Core има два језгра која су одвојена и међусобно комуницирају преко *FSB.*

Core2Quad – Dual Core2Duo, тј. *2 × 2* језгра састављена и комуницирају преко *FSB* . Такт се не сабира, већ се паралелизује.

У процесоре са више језгра убрајамо и *Intel i3, Intel i5* и *Intel i7* који су данас поред наведених процесора са више језгра највише заступљени код савремених рачунара [8].

ЗАКЉУЧАК

Данас је рачунар део свакодневнице, рачунар целим склопом извршава неке операције у разним програмима које корисник захтева. Да би рачунар одговорио захтевима које му се задају, рачунар мора поседовати добре карактеристике у складу са савременим добом да би одоворио на захтеве корисника. Поред других делова у рачунару, главну улогу у обради података има процесор, јер он представља мозак рачунара, односно део рачунара који обрађује и извршава инструкције које корисник задаје.

У архитектури рачунарских система, процесор представља део рачунара који се бави обрадом података које корисник задаје рачунару. Сваким даном се ради на усавршавању процесора, пре свега се раде нови концепти како би процесор у свом раду био ефикаснији и бржи у циклусу обраде и извршења података који се од њега захтевају. Од појаве првих процесора па до данас, направио се битан помак у усавршавању процесора, његове карактеристике су знатно боље, радни такт је већи, као и његова меморија. Са повећањем карактеристика процесора и његовом усавршавању, склоп процесора је сложенији, број транзистора већи, па се може рећи да и данас Муров закон у архитектури процесора важи, односно, да се број транзистора у процесору дуплира на сваких 18 месеци са развојем процесора. Тако да процесор представља један сложен скуп електронских елемената који врши обраду података у рачунару.

Литература

[1] **Радуловић Ј.**: *Архитектура рачунарских система*, предавања, Факултет инжењерских наука Крагујевац, 2013.

[2] **Стојчев М.**: *Архитектура и програмирање микрорачунарских система заснованих на фамилији процесора 80x86; I издање*, Универзитет у Нишу, Електронски факултет, 1999.

[3] **Миленковић Н.**: *Архитектура и организација рачунара*, Универзитет у Нишу, Електронски факултет, 2004.

[4] **Томашевић В.**: *Основи рачунарске технике*, Универзитет Сингидунум Београд, 2009.

[5] **Марчићевић Ж.**: *Основи информационо комуникационих технологија*, Висока пословна школа струковних студија Нови сад,

[6] **Гоцић М.**: *Информатика*, Висока школа примењених струковних студија Врање, 2009.

[7] Shanley T.: *x86 Instruction Set Arhitecture, comprehensive 32- and 64-bit Coverage, MindShare press,* Colorado Springs USA, 2009.

[8] http://ilijev.wordpress.com *(приступљено: јул, 2013)*