**Универзитет у Крагујевцу**

**Факултет инжењерских наука**

**Назив студијског програма: Машинско инжењерство**

**Ниво студија: Основне академске студије**

**Модул: Информатика у инжењерству**

**Предмет: Архитектура рачунарских система**

**Број индекса: 704/2011**

**Милан С. Радмановац**

# CISC процесори

## завршни рад

**Комисија за преглед и одбрану:**

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

Датум одбране:_____

Оцена:_____

У оквиру овог завршног рада кандидат треба да проучи препоручену, као и ширу литературу која се односи на CISC архитектуру процесора. На бази усвојеног знања кандидат треба да обради архитектуру CISC процесора, наведе и укратко опише значајне представнике, а затим да се определи за један тип CISC процесора и опширније објасни његову архитектуру.

Препоручена литература:

[1] Стојчев. М. (1999), „Архитектура и програмирање микрорачунарских система заснованих на фамилији процесора 80x86“, Универзитет у Нишу, Електронски факултет, Ниш.

[2] Хајдуковић. М. (2004), „Архитектура рачунара“, Факултет техничких наука, Нови Сад.

Крагујевац, датум

ментор:

др Јасна Радуловић

_____

**Резиме рада:**

У првом делу је описана централна процесорска јединица - CPU, која је претходила развоју архитектуре микропроцесора. Затим је обрађена архитектура CISC процесора, наведени су и значајни типови CISC процесора, и укратко су описани. У другом делу приказан је типичан пример CISC процесора, Intel 8086, објашњени су подсистеми и операције, регистри, типови инструкција и важније преформансе.

**Кључне речи:**

CPU, CISC процесор, CISC архитектура процесора, микропроцесори, инструкције.

**Summary of work:**

The first part describes the central processing unit - CPU, which is precede developing architecture microprocessors. Then processed CISC processor architecture, it listed the major types of CISC processors, and are briefly described. The second part shows a typical example of CISC processors, Intel 8086, explained subsystems and operations, registers, instruction types and importantly performance.

**Key words:**

CPU, CISC processor, the CISC processor architecture, microprocessors, instructions.

# Садржај

# 1. Увод

Рачунар је, знамо, направа која прима податке, складишти их, обрађује и издаје резултате. Процесор је задужен како за комплетно процесирање тј. обраду података (одатле му потиче и име), тако и за координацију рада осталих компонената рачунарског система. Процесор је, једноставно речено, део рачунара који може да се програмира за процесирање тј. обраду некаквих података.

Процесор, у рачунарству, је извршна јединица - прима и извршава инструкције прочитане из одговарајуће меморије. Када се каже само „процесор“ најчешће се мисли на централни процесор (*central processing unit - CPU*, централна процесорска јединица), али постоје и процесори специјалних намена као што су процесори сигнала, разни графички процесори итд. Сам по себи процесор не чини рачунар, али је један од најважнијих делова сваког рачунара.

Први процесори су били механички и практично нису били засебан део рачунара, затим електромеханички (релејни) па на бази електронских вакуумских цеви и били су јако велики. До значајног смањења димензија и повећања перформанси дошло је употребом транзистора (минипроцесори) и, у другој половини 20-ог века, интегралних кола (микропроцесори).

Данашњи кућни и персонални рачунари садрже у себи више чипова који би могли да се сврстају под дефиницију микропроцесора коју смо управо изложили: такозвани видео контролер је специјализовани микропроцесор који се бави искључиво генерисањем слике, генератор звука производи песмице које слушамо док се забављамо разним играма, *DMA - direct memory access* (директан приступ меморији) контролер се брине за пренос података између периферијских јединица и меморије, диск контролер комуницира са диск јединицом. Уколико је неки рачунар већ опремљен разним специјализованим контролерима, централни микропроцесор ће извршавати програме које корисник задаје и тек с времена на време координирати рад осталих контролера. Корисник обично нема никаквих могућности (а ни потребу) да комуницира са било којим хардверским модулом свога рачунара осим посредством централног микропроцесора. Микропроцесор се не налази само у центру било којег микрорачунара - разне микропроцесоре проналазимо у штампачима, часовницима, аутоматима за игре, видео рекордерима, микроталасним рернама и сличним уређајима.

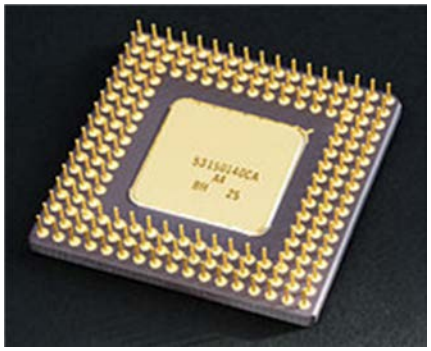
# 2. Централна процесорска јединица - CPU

Централна процесорска јединица, *CPU*, је хардвер у оквиру рачунарског система који извршава инструкције рачунарских програма обављањем основних аритметичких, логичних, и улазних/излазних операција система. Процесор има улогу нешто аналогну до мозга у компјутеру. Термин је у употреби у индустрији рачунара барем од раних 1960-их. Облик, дизајн и имплементација процесора су се драматично промениле од најранијег примера, али њихов рад остаје фундаментално исти.

На великим машинама, процесори захтевају једну или више штампаних плоча. На персоналним рачунарима и малим радним станицама, процесор је смештен у једном силицијумском чипу назива микропроцесор. Од 1970 микропроцесор класа процесора је скоро у потпуности преузела све остале *CPU* имплементације. Модерни процесори су великих размера интегрисана кола у пакете обично мање од четири центиметра квадратних, са стотинама повезивања пинова.

Слика 1. Intel 80486DX2 CPU одозго

Две типичне компоненте процесора су аритметичка логичка јединица - *arithmetic logic unit* - *ALU*[1], која врши аритметичке и логичке операције, а контролна јединица - *control unit* - *CU*[2], која издваја инструкције из меморије и декодује и извршава их, позива се на *ALU* када је то потребно.

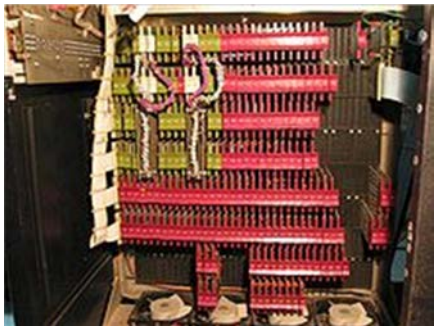

Слика 2. Intel 80486DX2 CPU одоздо

Не ослањају се сви рачунарски системи на централни процесор. Процесор или векторски процесор има више паралелних компјутерских елемената, без једне јединице сматра "центар". У модел дистрибуираног рачунарства, проблеми су решени од стране дистрибуираног повезаног скупа процесора.

[1] Дигитални склоп који врши аритметичке и логичке операције
[2] Централни део механизма који контролише његов рад

## 2.1. Дискретни транзистори и интегрисана кола процесора

Комплексност дизајна процесора повећана као различите технологија омогућила је изградњу мањих и поузданијих електронских уређаја. Први такав напредак дошао је са појавом транзистора - *transistor*[3]. Транзисторизовани процесори током 1950-их и 1960-их више нису морали да буду изграђени од гломазних, непоузданих, крхких и комутационих елемената као што су вакуум цеви и електрични релеји. Са овим побољшањем сложенији и поузданији процесори су изграђени на једном или више штампаних плоча које садрже дискретне компоненте.



Слика 3. Процесор, меморијско језгро и спољни бус интерфејс на *DEC PDP-8/I.*

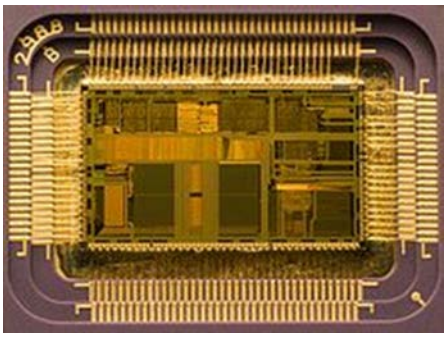
Током овог периода, метод за производњу транзистора у компактном простору стекао је популарност. Интегрисана кола - *integrated circuit - IC*[4] дозволила су велики број транзистора да буду произведени на једном од полупроводника или "чип." У почетку само врло основна неспецијализована дигитална кола, као што су капије *NOR gates*[5] минијатуризоване су у *ICs*. *CPUs* на основу ових "грађевинских блокова" *ICs* се генерално називају "интеграција малих размера" /*small-scale integration - SSI*[6] уређаји. *SSI ICs*, попут оних коришћених у *Apollo*[7] навођење рачунару, обично садржи и до неколико оцењујућих транзистора. Да изгради цео процесор ван *SSI ICs* потребно је хиљаде појединачних чипова, али и даље конзумирају много мање простора и енергије него раније дискретни транзистор дизајни. Као микроелектронске напредне технологије све већи број транзистора је постављен на *ICs*, чиме се смањује количина појединих *ICs* потребних за комплетан *CPU*. *Medium-scale integration - MSI*[8] и *large-scale integration - LSI*[9] (средње и велике интеграција) интегрисана кола повећала су број транзистора на стотине, а затим хиљаде.

Слика 4. Аполон рачунар DSKY кориснички интерфејс јединица.

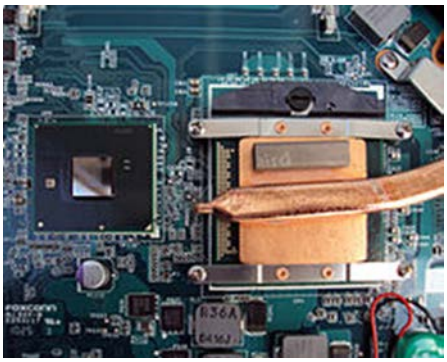
[3] Је полупроводнички уређај који се користи да се појачају и пребаце електронски сигнали и електрична енергија
[4] Integrated circuit - Електронски склоп произведен и од стране литографа
[5] Је дигитална логичка капија која спроводи логично NOR
[6] Small-scale integration
[7] Дигитални рачунар произведен за Apollo програм који је инсталиран на броду Apollo
[8] Уређај који садржи стотине транзистора на сваком чипу
[9] Уређај који садржи десетине хиљада транзистора на сваком чипу

## 2.2. Микропроцесори

У 1970-ој фундаменталне проналаске од *Federico Faggin*[10] (Silicon Gate MOS кола са само поравнатим капијама заједно са својом новом насумице одабраном методологији логичког дизајна) заувек су променили дизајн и имплементацију процесора. Од увођења првог комерцијално доступног микропроцесора - *Intel 4004* 1970, и први широко коришћени микропроцесор - *Intel 8080* у 1974, ова класа процесора је скоро у потпуности преузела све остале централне процесорске јединице за имплементацију. Главни и миникомпјутер произвођачи овог времена покренули су власничке програме привредног развоја да унапреде своје старије архитектуре рачунара, и на крају произвели сет инструкција компатибилних микропроцесора који су уназад компатибилни са њиховим старијим хардвером и софтвером. У комбинацији са појавом и евентуалног успеха на свеприсутне персоналне рачунаре, термин процесор се сада примењује готово искључиво према микропроцесорима. Неколико процесора се могу комбиновати у једном чипу за обраду.

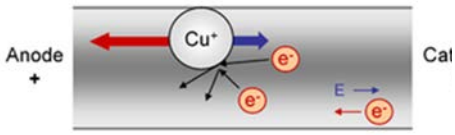


Слика 5. *Intel 80486DX2* микропроцесора (величина: 12×6.75 mm)

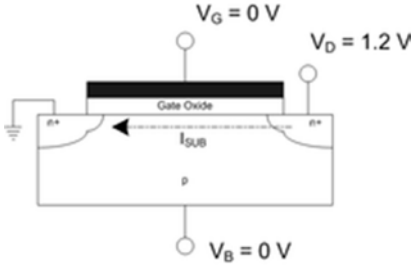


Слика 6. I*ntel Core i5* процесор на *Vaio E* серији матичне плоче лаптопа (са десне стране, испод топлотне цеви)

Претходне генерације процесора су реализовани као дискретне компоненте са бројним малим интегрисаним колима - *ICs* с једне или више плоча. Микропроцесори, с друге стране, су процесори произведени у веома малом броју *ICs*, обично само један. Укупна величина мањих процесора као резултат се спроводи на једној матрици, значи брже пребацивање времена због физичких фактора као смањене капије *parasitic capacitance*[11] - паразитске капацитивности. То је дозволило да синхрони микропроцесори имају такт - *clock rates*[12] у распону од неколико десетина мегахерца - MHz[13] до неколико гигахерца - GHz.




Слика 7. *Electromigration* због преноса момента из електрона се крећу у жице




Слика 8. *Subthreshold* цурење *nFET*

[10] Италијански физичар и инжењер електротехнике
[11] Је неизбежан и обично нежељена капацитивност која постоји између делова једне електронске компоненте
[12] Обично се односи на фреквенцију на којој процесор ради
[13] Јединица за фреквенцију дефинисан као број циклуса у секунди на периодичне појаве

Док сложеност, величина, конструкције, и општи облик процесора се огромно променио од 1950, приметно је да се основни дизајн и функција није много променила уопште. Скоро сви процесори заједнички се данас могу веома прецизно описати као *von Neumann* - складишти програмске машине. Екстремне минијатуризације електронских капија изазива ефекте феномена као што су *electromigration*[14] и *subthreshold leakage*[15] цурења да постану много значајнија. Ови новији проблеми су међу многим факторима који истраживаче наводе да истражују нове методе израчунавања, као што је *quantum computer*[16] - квантни компјутер, као и да прошири употребу *parallel computing*[17] - паралелизма и других метода које проширују корисност класичног *von Neumann* модела.

## 2.3. Операција

Основни рад већине процесора, без обзира на физичке форме коју узимају, је да изврши низ ускладиштених инструкција под називом програм. Програм је представљен низом бројева који се држе у некој врсти *computer memory*[18] - меморије рачунара. Постоје четири корака које готово сви процесори користе у свом раду: нађе, декодира, извршавају и врати назад.

Слика 9. Предњи панел од *IBM 701* компјутера уведеног 1952.

Први корак подразумева преузимање инструкције (која је представљена од стране једног броја или низа бројева) из програмске меморије. Локација у програмској меморији зависи од програмског бројача - *program counter* - *PC*[19], који складишти број који идентификује тренутну позицију у програму. Након што је инструкција учитана, *PC* се инкрементира за дужину инструкције речи у смислу меморијских јединица. Да би инструкција била преузета мора бити враћена из релативно споре меморије, изазивајући мировање процесора док чека инструкцију да се врати. Ово питање је у великој мери решено у модерним процесорима од стране кеша - *caches*[20] и умрежених архитектура - *pipeline architectures*[21].

[14] Транспорт материјала изазван постепеним кретањем јона у проводнике
[15] Је струја која тече између извора и MOSFET одвода када је транзистор у субтхресхолд(subthreshold) региону
[16] Уређај за израчунавање који омогућава директне употребе квантно-меаничких појава да врше опреације над подацима
[17] Облик обрачуна у којем многи прорачуни се врше истовремено
[18] Односи се на физичке уређаје који се користе за складиштење програма (низови инструкција) или податке, привремено или трајно, за употребу у рачунару или неког другог дигиталног електронског уређаја
[19] Процесор је регистар који показује где се рачунар налази у својој програмској секвенци
[20] Је компонента која транспарентно складишти податке, тако да будући захтеви за те податке могу бити задовољени брже
[21] Је скуп повезаних елемената обраде података у серији, тако да излаз једног елемента је улаз у следећи

Инструкција коју процесор преузима из меморије се користи да одреди који процесор да ради. У кораку декодирања, инструкција је разбијена на делове који имају значај на друге делове процесора. Начин на који се тумачи нумеричка вредност инструкција дефинисан је скуп инструкција у архитектури процесора. Често, једна група од бројева у инструкцији, звана операциони кôд, указује која операција да се изврши. Преостали делови броју обично пружају информације потребне за ту инструкцију, као што операнди пружају за додатак операције. Такви операнди могу бити дати као константна вредност или као место за лоцирање једној од вредности: регистрација - *processor register*[22] или меморијска адреса - *memory address*[23], на основу неког режима адресирања. У старијим пројектима су делови процесора задужени за декодирање инструкција били непроменљиви хардверски уређаји. У више апстрактних и компликован процесора, микропрограм - *microprogram*[24] се често користи да помогне у превођењу инструкција у разним конфигурационим сигналима за *CPU*. Овај микропрограм је понекад уписан тако да може бити модификован да промените начин на који *CPU* декодира инструкције чак и након што је произведен.

Слика 10. Пример *RAM*-а

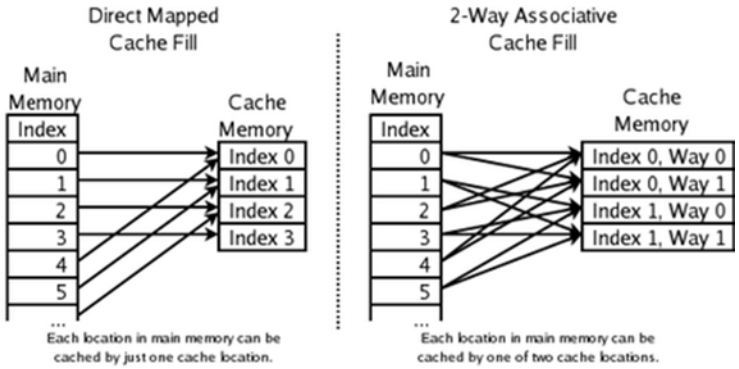
Након пронађених и декодираних корака, извршни корак се изводи. Током овог корака, разни делови процесора су повезани тако да могу да обављају жељену операцију. Ако, на пример, додатак операција тражи, аритметика логичка јединица ће бити повезана са скупом улаза и скупом излаза. Улази ће обезбедити бројеве које треба додати, а резултати ће садржати коначну суму. Аритметичка логичка јединица садржи кола да врше једноставне аритметичке и логичке операције на улазима (као додатак операцијама и операције над битовима). Ако додавање операција производи сувише велики резултат за *CPU* да обради, аритметика *overflow flag*[25] - прекорачења у *FLAGS register* или *status register*[26] се такође може подесити.

Слика 11. 8 Bit *ALU*

Последњи корак, повратак, једноставно "пише назад" резултате извршених корака ка неком облику меморије. Врло често резултати су написани на неки интерни регистар *CPU*-а за брз приступ каснијим инструкцијама. У другим случајевима, резултати могу бити написана на спорију, али јефтинију и већу, основну меморију. Неке врсте инструкција радије манипулишу бројачем програма него да директно произведу

[22] Мала количина складиштеног капацитета као део процесора или други дигитални процесор
[23] Су концепт подаци који се користи на различитим нивоима софтвера и хардвера за приступ примарне меморије рачунара за складиштење
[24] Microcode у одређеној примени процесора
[25] Је обично један бит у систему статус регистра који се користи да означи када је прекорачена аритметика у току раду
[26] Садржи тренутно стање процесора

резултате. То се обично назива "скок - *jump*[27]" и олакшава понашање као што су петље и условна извршавања програма (помоћу условног скока), и функција у програмима. Многе инструкције ће такође променити стање цифара у "*status*" регистар. Ови регистри се могу користити да утичу како се програм понаша, јер они често указују на исход различитих операција. На пример, једна врста "упореди" инструкција разматра две вредности и поставља низ у статусним регистрима према којем је један већи. Овај регистар би онда могао да се користи касније као скок инструкција да утврди ток програма.



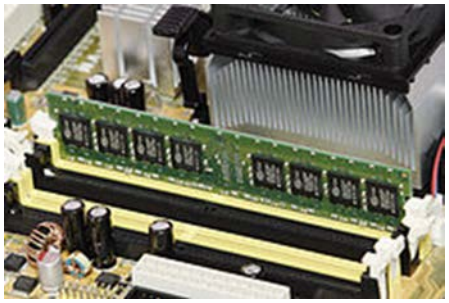

Слика 12. *Direct Mappes Cache Fill*: свака локација у главној меморији може бити повезана са једном кеш-*Cache* локацијом
*2-Way associative Cache Fill*: свака локација у главном меморији може бити повезана са једном од две кеш-*Cache* локације

Након извршења инструкција и повратка на резултујуће податке, цео процес се понавља, са следећег циклуса инструкција нормално налажење следеће секвенце инструкција због инкрементиране вредности у програмском бројачу. Ако је завршена инструкција скок, програмски бројач ће бити модификован да садржи адресу инструкције која је скочила, и извршење програма наставља нормално.

## 2.4. Дизајн и имплементација

Основни концепт процесора је следећи:

Укорењен у дизајну процесора списак основних операција које процесор може обављати, зове се сет инструкција. Такве операције могу укључивати додавање или одузимање два броја, упоређујући бројеве, или скакање на други део програма. Свака од ових основних операција представља одређену секвенцу битова - *bit*[28], овај низ се зове операциони кôд за ту операцију. Слање одређеног операционог кôда у *CPU* ће довести до тога да изврши операцију коју представља тај операциони кôд. Да изврши инструкцију у рачунарском програму, процесор користи операциони кôд за те инструкције, као и своје аргументиме (на пример два броја да се додају, у случају додавања операције). Компјутерски програм је секвенца инструкција, свака са инструкцијом укључујући и операциони кôд и ту операцију као аргументе.



Слика 13. 1GB SDRAM-а монтирана на личном рачунару. Пример примарног складиштења.

[27] Врста процеса који има дискретне покрете, радије назван “скок“, него мали покрет
[28] Је основни капацитет података у рачунарству и телекомуникацијама, bit се представља само са 1 или 0

Стварну математичку операцију за сваку операцију изводи подјединица процесора позната као аритметичка логичка јединица или *ALU*. Поред коришћења његове *arithmetic logic unit* за обављање операција, процесор је такође одговоран за читање следеће инструкције из меморије, чита податке из аргумената, из меморије, и писање резултата у меморију.

У многим дизајнима *CPU*-а, сет инструкција ће се јасно разликовати од операција које учитавају податаке из меморије, и они који обављају математичке операције. У овом случају се подаци који се учитавају са меморије чувају у регистрима, и математичка операција не узима никакве аргументе него једноставно врши математику о подацима у регистрима и пише их у нови регистар, чија је вредност одвојена операција и може се уписати у меморију.

### 2.4.1. Контролна јединица

Контролна јединица процесора садржи кола која користи електричне сигнале да усмери цео компјутерски систем за извршавање ускладиштених инструкција програма. Контролна јединица не извршава инструкције програма, већ их усмерава на друге делове система да то учине. Контролна јединица мора да комуницира са обе, аритметичком логичком јединицом и меморијом.

### 2.4.2. Такт процесора

Такт процесора је брзина којом микропроцесор извршава инструкције. Сваки рачунар садржи интерни такт који регулише брзину којом се извршавају инструкције и синхронизује све компоненте рачунара. Процесор захтева фиксни број трајања такта (или циклуса) да изврши сваку инструкцију. Што је такт већи, више инструкције процесор може да извршава по секунди.

Слика 14. Минијатурни *4MHz* кварц кристал затво-рен у херметичкој амбалажи, полован *HC-49/US* као резонатор у кристалном осцилатору.

Већина процесора, па и већина секвенцијалних логичких уређаја - *sequential logic*[29], синхрони су у природи. То јест, они су дизајнирани и функционишу на претпоставкама о синхронизацији сигнала. Овај сигнал, познат као такт сигнал - *clock signal*[30], обично поприма облик периодичног

[29] Је врста логичког кола чији излаз зависи не само од садашње вредности његових улазних сигнала, него и претходних улаза

[30] Је посебан тип сигнала који осцилира између високог и ниског стања и користи се као метроном да координира акцију кола

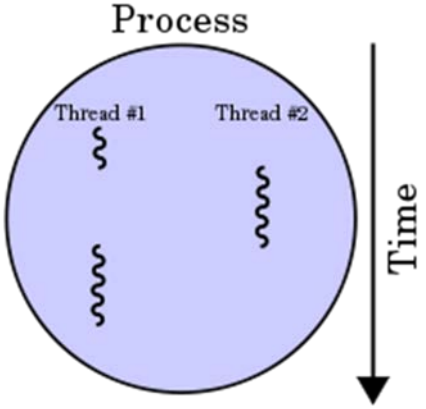
квадратног таласа - *square wave* [31]. До обрачуна је максимално време кад електрични сигнали могу да се крећу у различитим гранама многих процесорских кола, дизајнери могу изабрати одговарајући период за такт сигнала.

Овај период мора да буде дужи од временског периода који је потребан за сигнал да се креће, или шире, у најгорем сценарију. У постављању такта периода у вредности далеко изнад најгоре, кашњења - *propagation delay*[32], могуће је дизајнирати цео *CPU* и начин на који премешта податке око "*edge*[33]" на успону и паду тактног сигнала. Ово има предност да знатно поједностављује процесор, како из перспективе дизајна тако и из компонента-рачунање перспективе. Међутим, она такође носи ману која мора да чека цео процесор на њеним најспоријим елемената, иако су неки много бржи. Ово ограничење се у великој мери компензује разним методама повећања *CPU* паралелизма.

### 2.4.3. Истовремени рад са више програма

Опис основне операције на процесору у претходном одељку се описује у најједноставнијем облику да процесор преузме. Овај тип процесора, обично се назива субскаларни - *subscalar*, послује на и извршава једну инструкцију на један или два комада података истовремено.

Овај процес доводи до неефикасности у субскаларном процесору. Пошто само једна инструкција се извршава на време, цео процесор мора да сачека ту инструкцију да заврши пре него што пређе на следећу инструкцију. Као резултат тога, *subscalar* процесор добија "прекид извршења" на инструкције које трају дуже од једног цилкуса да заврши циклус извршења. Чак и додавањем друге извршне јединицу не побољшава много перформансе, него један пут се прекине извршење инструкције, сада се два пута прекине извршење инструкције и број неупотребљених транзистора се повећава. Овај дизајн, где извршни извори процесора могу да раде само на једној инструкцији у једном тренутку, може само да евентуално достигне скаларне перформансе (једна инструкција по циклусу). Међутим, перформансе су скоро увек субскаларне (мање од једне инструкције по циклусу).



Слика 15. Процес са две нити извршења на једном процесору.

Покушаји да се постигне скалар и боље перформансе довели су у различитим дизајн методологијама које изазивају да се процесори понашају мање линеарно и више паралелно. Када се говори о паралелизму у процесорима, два термина се обично користе да класификују ове технике ди-

[31] Је врста не-синусоидалног таласа, већина се обично користи у електроници
[32] Је дужина времена које се узима у количини од интереса да достигне свој циљ
[33] Је транзиција у дигиталном сигналу или од ниског до високог (0 до 1) или од високог до ниског(1 до 0)

зајна. Ниво инструкције паралелизама – *Instruction Level Parallelism - ILP*[34] настоји да повећа стопу по којој се инструкције извршавају унутар *CPU*-а (то јест, да повећа искоришћеност на самом чипу извршења ресурса), и на нивоу паралелизма - *Thread Level Parallelism-TLP*[35] да повећа број нити (ефикасно индивидуални програми) да процесор извршава истовремено. Свака методологија се разликује у начину на који се примењују, као и релативну ефикасност да приуште у повећању перформанси процесора за апликације.

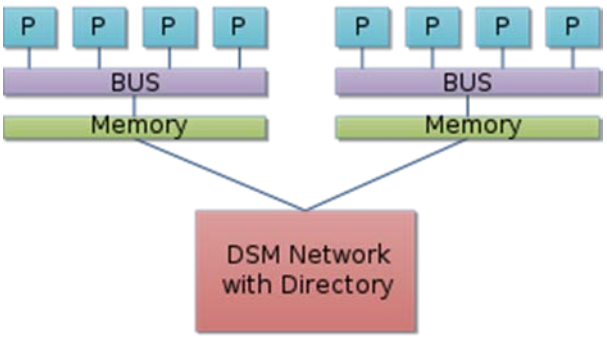
### 2.4.3.1. Паралелизам на нивоу инструкција

Један од најједноставнијих метода да постигнете повећање паралелизам је да започне прве кораке преузимања инструкције и декодирање пре извршења инструкције. Ово је најједноставнији облик технике познат као инструкција протицања обраде - *instruction pipeline*[36], и користи се у готово свим модерним процесорима опште намене. Протицање обраде дозвољава да се више од једне инструкције изврши у било ком датом тренутку прекидањем извршавања инструкције у дискретним фазама. Ово раздвајање се може упоредити са покретном траком, у којима је инструкција комплетнија у свакој фази све док се заврши протицање обраде и затвори.

### 2.4.3.2. Нивои паралелизма

Друга стратегија постизања учинка је да извршава више програма или нити у паралели, што је познато као паралелно рачунарство. У *Flynn's taxonomy*[37], ова стратегија је позната као *Multiple Instructions Multiple Data - MIMD*[38].

Једна од технологија која се користи за ову сврху је мултипроцесинг – *Multiprocessing - MP*[39]. Главни принцип ове технологије познат је као симетрични мултипроцесинг – *Symmetric Multiprocessing - SMP*[40], где мали број процесора деле кохерентан поглед на њихове меморије система. У овој шеми, сваки *CPU* има додатни хардвер који константно има увид на стање меморије. Заобилажењем заузете мемо-

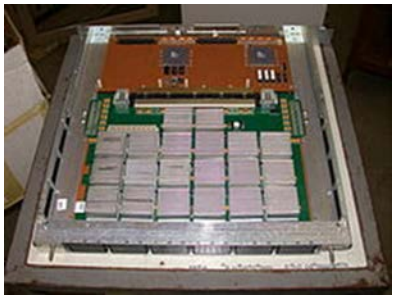



Слика 16. Једна могућа архитектура *NUMA*

[34] Је мера колико операција у рачунарском програму може да се истовремено изврши
[35] Задатак паралелизам се фокусира на дистрибуцију извршавања процеса преко различитих паралелних компјутерских чворова
[36] Је техника која се користи у пројектовању рачунара да повећају свој проток за руковање
[37] Је класификација рачунарских архитектура
[38] Је запослено техника за постизање паралелизам
[39] Је употреба две или више централних јединица за обраду у оквиру једног рачунарског система
[40] Укључује мултипроцесорску хардверску архитектуру, где су два или више идентична процесора повезана на јединствену дељену меморију

рије, *CPU*-и могу да сарађују на истом програму и програми могу да мигрирају са једног процесора на други. Да бисте повећали број сарадње процесора иза неколицине, шеме као што су нејединствени приступ меморији – *Non Uniform Memory Access-NUMA*[41] и директоријум на бази кохерентности протокола, уведени су 1990. *Symmetric Multiprocessing* системи који су ограничени на мали број процесора док су *NUMA* системи изграђени са хиљадама процесора.

### 2.4.3.3. Подаци паралелизама



Слика 17. Cray J90 процесор модул са четири скаларна / векторска процесора

Сви важнији примери процесора се баве паралелизмом података. У ранијим разговорима сви помињу процесоре као неку врсту скаларног уређаја. Као што назив каже, векторски процесори се баве са више података у контексту једне инструкције. Ово је супротност са скаларним процесорима, који се баве једним податаком за сваку инструкцију. Коришћење *Flynn's taxonomy* - класификација рачунарских архитектура, ове две шеме суочавања са подацима се генерално називају *Single Instruction Multiple Data - SIMD*[42] и *Single Instruction Single Data - SISD*[43]. Велики алати за креирање процесора који се баве векторима података лежи у оптимизацији задатака који имају тенденцију да захтевају исту операцију (на пример, збир или производ тачака) који треба извести на великом скупу података. Неки класични примери ових типова задатака су мултимедијалне апликације (слике, видео и звук), као и многе врсте научних и инжењерских послова. Док скаларни процесор мора да заврши цео процес прибављање, декодирање и извршење сваке инструкције и вредност у скупу података, векторски процесор може да изврши једну операцију на релативно великом скупу података са једном инструкцијом. Наравно, ово је могуће само када апликација има тенденцију да захтева многе кораке који примењују једну операцију на великом скупу података.


Слика 18. Pentium са MMX спецификациіом

Већина раних векторских процесора, као што су *Cray-1* (супер компјутер), повезани су искључиво са научним истраживањима и криптографским апликацијама. Међутим, као што је мултимедија углавном пребачена на дигиталне медије, потреба за неком врстом *SIMD*-а у процесорима опште намене је постала значајна. Убрзо након што је укључивање децималних тачака - *floating point*[44] јединице извршења су почеле да постају

[41] Је компјутерска меморија која се користи у мултипроцесингу, где време приступа зависи од меморијске локације у односу на процесор
[42] Је класа паралелних рачунара у Flynn's taxonomy
[43] Је термин који се односи на рачунарску архитектуру у којој један процесор извршава један ток инструкција, да се ради о подацима ускладиштеним у једној меморији
[44] Начин представљања реалних бројеве на такав начин да може да подржи широк спектар вредности

уобичајена појава у процесорима опште намене, спецификација и имлементацијама *Single Instruction Multiple Data* извршних јединица такође су почели да се појављују у процесорима опште намене. Неке од ових раних *SIMD* спецификација као што је *HP*-ов (*Hewlett-Packard*) *Multimedia Acceleration eXtensions - MAX* и *Intel's MMX* су недецималне - *integer*[45]. Ово се показало као значајна препрека за неке програмере, пошто многе од апликација које имају користи од *SIMD* првенствено се баве децималним бројевима. Ови рани дизајни су допуњени и преправњени у неке од уобичајених, модерним *SIMD* спецификацијама, које су обично повезане са једним *Instruction Set Architecture - ISA*[46]. Значајни савре-мени примери су *Intel's SSE* и *PowerPC*[47] повезан са *AltiVec* (познат као *VMX* - децимални и целобројни *Single Instruction Multiple Data* сет инструкција).

## 2.5. Преформансе

Перформансе или брзина процесора зависи од такта процесора (обично дата у количини херца - *Hz*) и инструкција по такту - *Instructions Per Clock - IPC*[48], које су заједно фактори за инструкције по секунди - *Instructions Per Second - IPS*[49] који може да обавља процесор. Многе пријављене *IPS* вредности су представљене као "максимална" извршења такта на вештачкој секвенци инструкција са неколико грана, док се реална оптерећења састоје од мешавине инструкција и апликација, од којих се неке дуже извршавају од других. Перформансе меморијске хијерархије такође веома утиче на перформансе процесора, проблем се разматрати у *Million instructions per second - MIPS* прорачунима. Због ових проблема, разне стандардизоване тестове, често називају "*benchmarks*[50]" за ову намену, као што *SPECint*[51] - су развијени да покушају да измере реално ефективне преформансе у често коришћеним апликацијама.

Слика 19. *Microprocessor Intel Core i7-3770K*: 4 језгра такта 3.5 GHz, до 3.9 GHz у турбо моду

[45] Је форма природних бројева укључујући 0, 1, 2,... и негативне бројеве -1, -2,...
[46] Је део архитектуре рачунара који се односи на програме, укључујући и матерње типове података, инструкције, регистаре, обрађајуће режиме, меморијску архитектуру, и екстерни I/O.
[47] RISC архитектура креирана од стране Apple–IBM–Motorola алијансе
[48] Учинак процесора: просечан број извршених инструкција за сваки циклус
[49] Је мера брзине процесора рачунара
[50] Је акт покренутог компјутерски програм, скупа програма или друге операције, у циљу процене релативне перформансе објекта, обично тако што ћете покренути низ стандардних тестова и оцењивања против њега.
[51] Је рачунар са benchmark спецификацијама за целобројну прераду процесорске снаге

Обрада перформансе рачунара се повећава коришћењем вишејезгралних процесора, што се у суштини усредсреди на два или више појединачна процесора у једном интегрисаном колу. У идеалном случају, двојезгрални процесор би скоро двоструко био снажнији него процесор са једним језгром. У пракси, међутим, појачање перформансе је далеко мање, само око 50%, због несавршених софтверских алгоритама и примена. Повећање броја језгара у процесор (тј. *dual-core, quad-core*, итд) повећава оптерећење које рачунар може да савлада. То значи да процесор сада обрађује бројне асинхроне догађаје, прекиде, итд које могу да смање губитак на *CPU*, када је претрпан. Најбоље је да мислимо о овим многобројним језгрима као и различитих нивоа у преради, да сваки ниво извршава другачији задатак. Понекад, ова језгра ће носити исте задатке као језгара суседна њима ако једно језгро није довољно да обради информацију да спрече пад система.

## 2.6. Скуп инструкција

Скуп инструкција, или сет инструкција архитектуре - *ISA*, је део архитектуре рачунара који се односи на програме, укључујући и матерње типове података, инструкције, регистре, режим адресирања, меморијску архитектуру и спољни *I/O*. *Instruction Set Architecture* укључује спецификацију скупа *opcodes*[52]-а (машински језик), као и домаће команде реализоване одређеним процесором.

Скуп инструкција архитектура се разликује од микроархитектуре, која је скуп процесорских дизајна која се користи да спроведе сет инструкција. Рачунари са различитим микроархитектурама могу да деле заједнички скуп инструкција. На пример, *Intel Pentium* и *AMD Athlon* спроведе скоро идентичне верзије *x86* скупа инструкција, али имају радикално различите унутрашње дизајне.

### 2.6.1. Класификација скупова инструкција

Комплексан скуп инструкција рачунара, C*omplex Instruction Set Computer - CISC* има многе специјализоване инструкције, које могу бити ретко коришћене у практичне програме. Редуковани скуп инструкција рачунар, *Reduced Instruction Set Computing - RISC* поједностављује процесор спроводећи инструкције које се често користе у програмима; необичне операције се примењују као потпрограм, где се додатно време потребно процесору за извршења надокнађује њиховом ретком употребом. Теоретски важни типови су минимални скуп инструкција рачунара и један скуп инструкција рачунара, али они се не спроводе у комерцијалним процесора. Друга варијација је *VLIW - Very Long Instruction Word*[53] где процесор прима многе инструкције кодиране и преузете у једној инструкција речи.

[52] Део инструкције машинског језика која одређује операцију која треба да се изврши
[53] Се односи на архитектуру процесора дизајнирану да искористе паралелизам на нивоу инструкција

### 2.6.2. Типови инструкција

Операције које су на располагању у већини скупова инструкција укључују:

- Управљање подацима и меморијске операције,
  - подешавање регистара на фиксну константну вредност,
  - премештање податка из меморијске локације у регистар, или обрнуто. То је урађено ради прибављања податка за обављање каснијег рачунања или да сачува резултате неког израчунавања,
  - читање и писање података са хардверских уређаја,
- Аритметика и логика,
  - сабирање, одузимање, множење или дељење вредности два регистра стављајући резултат у регистар,
  - обављање операција над битовима,
  - упоређивање две вредности у регистрима,
- Контрола протока,
  - разгранавање на другу локацију у програму и извршавање те инструкције,
  - условно разгранавање на другу локацију, ако неки услов важи,
  - индиректно разгранавање на другу локацију, штедећи на локацији следеће инструкције као тренутак да се врате (позив).

### 2.6.3. Делови инструкције

На традиционалним архитектурама, инструкција обухвата операциони кôд наводећи операцију коју треба извршити, као што су "*add contents of memory to register*" - додај садржај меморије у регистар, и нула или више операнада спецификаторима, што може навести регистре, меморијске локације или дословне податке. Операнд спецификатора може имати адресарски режим који одређују њихово значење или може бити у фиксним пољима.

У *very long instruction word* архитектури, која укључује многе *microcode*[54] архитектуре, више истовремених опкôдова и операнади су наведени у једној инструкцији.

Неки егзотични скупови инструкција немају поље *opcode* (као што су *Transport Triggered Architectures*[55] - TTA или *Forth virtual machine*[56]), само операнд. Друге необичне "0-операнд" сет инструкција немају никакав операнд спецификатор поља.

[54] Слој структурних података укључених у реализацију виших нивоа машинског кода
[55] Је врста процесорског дизајна у коме програм директно контролише унутрашње аутобуса за превоз процесора
[56] Је адреса нижег нивоа кôда потребан за извршење операције

### 2.6.4. Дужина инструкције

Величина или дужина инструкција широко варира, од толико мало као четри bit-а у неким микроконтролерима до више стотина bit-а у неким *Very Long Instruction Word* системима. Процесори који се користе у персоналним рачунарима, главним рачунарима и суперкомпјутерима имају инструкције величина између 8 и 64 bit-а. Најдужа могућа инструкција на *x86* је 15 byte-ова (120 bit-ова). Унутар сета инструкција, различите инструкције могу имати различите дужине. У неким архитектурама, посебно рачунари који имају смањени сет инструкција (*RISC*), инструкције су фиксне дужине, обично одговарају архитектури *word size*[57]. У другим архитектурама инструкције имају промењљиве дужине, обично саставни већи број byte-а или *halfword*.

### 2.6.5. Број операнада

Сетови инструкција могу бити категорисани по максималном броју операнада изричито наведеним у инструкцијама. Свака инструкција наводи известан број операнада (регистри, меморијске локације или непосредне вредности) експлицитно. Неке инструкције дају имплицитно један или оба операнада, као такви се налазе на врху гомиле, или у имплицитном регистару. Када је неки од операнада имплицитно дат, број операнада наведених у једној инструкцији је мањи од *arity*[58]-ја операције. Када "дестинација операнда" експлицитно наводи дестинацију, број операнада спецификаторима у инструкцији је већа од *arity*-ја операције. Нека сетови инструкција имају различите бројеве операнада за различите инструкције.

[57] Је термин за природне јединице података које користе одређени дизајн процесора
[58] Је број аргумената или операнада које функција узима

## 2.7. Организација процесора

Централни процесори обично садрже:

- Управљачку јединицу (*control unit*), која управља радом осталих компоненти, конкретно операционе јединице,
- Операциону јединицу (execution unit), која типично садржи:
  - Аритметичко логичку јединицу (*Arithmetic logic unit*), која врши аритметичке и логичке операције,
  - Регистре, који служе за привремено складиштење података при извршавању програма (регистри опште намене), као и за чување информација о тренутном стању програма који се извршава (програмски бројач – *program counter*, показивач стека – *stack pointer*, прихватни регистар инструкције - *reception instruction register*, програмска статусна реч - *program status word* и др.),
- Подсистем за везивање са меморијом и периферијама.

Модернији процесори имају и јединице за рад са бројевима са покретним зарезом, брзу интерну меморију итд. Супер скаларни процесори - *superscalar CPU*[59] имају и по више операционих јединица. што им омогућава да извршавају неколико инструкција истовремено (када оне нису међусобно зависне).

### 2.6.1. Подела процесора

Постоји више подела процесора, а то су:

- По томе са колико инструкција и са колико података раде у једном „кораку“ - *SISD*, *SIMD*, *MIMD*, *MISD* процесори,
- По архитектури и скупу инструкција - *CISC* и *RISC* процесори.

Мање званичне подваријанте и комбинације поделе процесора:

- Процесоре са јако редукованим скупом инструкција (*ERISC - Extremely Reduced Instruction Set Computer*),
- Процесоре са "проширеним" редукованим скупом инструкција, у којима се већина инструкција не преводи већ су директно подржане али садрже и комплексне инструкције (*ERISC - Extended Reduced Instruction Set Computer*). Овакав скуп инструкција је и угодан за програмирање и резултује већом брзином извршавања али сам процесор постаје значајно комплекснији.

[59] Имплементира облик паралелизма у једном процесору

#### 2.6.1.1. Према количини обрађујућих података у једном „кораку“

Ову поделу је 1972. године предложио Мајкл Џ. Флин, и по њему се она назива „Флинова таксономија“:

- једну инструкцију са једним податком (тзв. скаларни процесори, *SISD - Single Instruction Single Data*). Подваријанта (супер скаларни) могу да у току извршавања одреде које парове инструкција и података могу да изврше у исто време и то и учине,
- једну инструкцију извршавају на више података одједном (тзв. векторски процесори, *SIMD - Single Instruction Multiple Data*),
- извршавају више независних инструкција, сваку на једном, заједничком, податку( *MISD - Multiple Instruction Single Data*).
- извршавају више независних инструкција, сваку на својим подацима ( *MIMD - Multiple Instruction Multiple Data*),

Иако се *SIMD* некада користио углавном на векторским супер рачунарима попут оних које је 70-их година 20. века популарисао *Cray*, потреба за обрадом мултимедијалних података је довела до додавања *SIMD* инструкција у архитектуру процесора опште намене, тренд који је изродио технологије као што су *PowerPC*-јев *AltiVec, Intel*-ови *MMX, SSE, SSE2, SSE3* и *SSE4, AMD*-ов *3DNow!, SPARC*-ов *VIS, Sun*-ов *MAJC, PA-RISC*-ов *MAX и MIPS*-ови *MDMX* и *MIPS-3D.*

#### 2.6.1.2. По архитектури и скупу инструкција

Делимо на:

- Процесоре са комплексним скупом инструкција, у којима се свака комплексна инструкција интерно преводи у низ *microcode* инструкција (*CISC - Complex Instruction Set Computer*). Овакав скуп инструкција је обично „угоднији“ за програмирање, али генерално резултује мањом брзином извршавања. Свакако значајан пример *CISC* архитектуре је интелова (и *АМД*-ова) *80x86* фамилија, а овакав дизајн су користили и *CDC 6600, System/360, z/Architecture, VAX, PDP-11* и *Motorola 68000,*
- Процесоре са редукованим скупом инструкција, у којима се инструкције не преводе већ су директно подржане (*Reduced Instruction Set Computer*). Овакав скуп инструкција је обично „незгоднији“ за програмирање али генерално резултује већом брзином извршавања. Повећање величине радне меморије, као и развој компајлера су довели до тога да је данас *RISC* општеприхваћена филозофија у дизајну микропроцесора. Чак и модерни *Intel*-ови и *AMD*-ови процесори, иако програмеру изгледају као *CISC* машине, интерно разбијају *CISC* инструкције у низове интерних *RISC* инструкција (микрооперација) које се затим извршавају на суперскаларном језгру.

# 3. Опште прихваћена подела

Процесори са редукованим скупом инструкција, у којима се инструкције не преводе већ су директно подржане, су обично „незгоднији“ за програмирање али генерално резултују већом брзином извршавања. Повећање величине радне меморије, као и развој компајлера су довели до тога да је данас *RISC* општеприхваћена филозофија у дизајну микропроцесора. Чак и модерни *Intel*-ови и *AMD*-ови процесори, иако програмеру изгледају као *CISC* машине, интерно разбијају *CISC* инструкције у низове интерних *RISC* инструкција (микрооперација) које се затим извршавају на супер скаларном језгру.

## 3.1. CISC процесори - Complex Instruction Set Computer

Код процесори са комплексним скупом инструкција се свака комплексна инструкција интерно преводи у низ микрокôд инструкција. Овакав скуп инструкција је обично „угоднији“ за програмирање, али генерално резултује мањом брзином извршавања.

### 3.1.1. Шта је CISC

*CISC*, што је скраћеница за *Complex Instruction Set Computer*, је филозофија за дизајнирање чипова и програма који чине ефикасно коришћење меморије. Свака инструкција у *CISC* сету инструкција омогућава извршавање низ операција унутар процесора. Ово смањује број инструкција потребних за спровођење одређеног програма и омогућава програмеру да науче мали, али флексибилан скуп инструкција.

Слика 20. Intel 386 Family

*CISC* је ретроактивна дефиниција која је уведена да се раздвоји од *RISC* дизајна микропроцесора. За разлику од *RISC*-а, *CISC* чипови имају велику количину различитих и сложених инструкција. Аргумент за њене даље употребе указује да дизајнери чипова чине живот лакшим за програмере смањујући количину инструкција потребних за програмирање процесора. Због високе цене меморије и микропроцесора *CISC* складиштење сматрано је супериорним, као и због захтева за малим и брзим кôдом. *CISC* базирани системи покривају огромну већину тржишта потрошача десктопова. Већина ових система су засноване на *x86*[60] архитектури или варијанти. У *Amiga, Atari* и пре 1994-те *Macintox* системи такође користе *CISC* микропроцесор.

[60] 32-битна серија Intel рачунарских микропроцесора

Пошто су најраније машине програмиране на асемблерском језику, а меморија је спора и скупа, *CISC* филозофија је задовољавајућа, а најчешће примењена у таквим великим рачунарима као што је *PDP-11*[61] и *DEC System*[62] 10 и 20 машина.

Најчешћи микропроцесор дизајн, укључујући *Intel (R) 80x86*[63] и *Motorola 68k*[64] серије, такође прати *CISC* филозофију.

### 3.1.2. CISC филозофија 1: Писање Microcode-а

Процесор је дизајниран на основу идеје да декодира и изврши сваку инструкцију у упутству „Подесите процесор“. Ово је добро за једноставне дизајне са неколико регистара, али је сложеније архитектуре тешко изградити јер се као контролна путања логике може тешко спровести. Дизајнери су укључену тактику саградили за неку једноставну логику за контролу путање података између различитих елемената процесора и при том се користи поједностављен *microcode* сет инструкција за контролу логике путање података. Ова врста имплементације је позната као *microprogrammed*[65] имплементација. У *microprogrammed* систему главни процесор има неку уграђену меморију (обично *ROM*[66]), која садржи групе *microcode* инструкција које одговарају сваком машинском језику. Када инструкција за машински језик стигне у централни процесор, процесор извршава одговарајућу серију *microcode* инструкција.

Због инструкција може бити преузето до 10 пута брже од локалног *ROM*-а него из главне меморије. Дизајнери су почели да стављају онолико инструкција колико је могуће у *microcode*-ове. Неки процесори могу се наручити са прилагођеним *microcode*-ом које ће заменити ако их често користе, али споро раде у одређеној апликацији.

#### 3.1.2.1. Предности Microcode имплементације

- *Microcode* меморија може да буде много бржа од главне меморије, сет инструкција се може имплементирати у *microcode* без губљења брзине преко кабла имплементације,
- Нови чипови се лакше имплементирају и захтевају мање транзистора од примене истог сета инструкција са посебном логиком,
- *Microprogrammed* дизајн може бити модификован за брзо руковање потпуно новим сетовима инструкција.

[61] 16-битна серија миникомпјутера
[62] Digital Equipment Corporation
[63] 16-битни микропроцесор чип
[64] Породица 32-битним CISC микропроцесора
[65] Написан microcode
[66] Read-only memory

Коришћење *microcode* комплета инструкција, *IBM-360*[67] серија је била у стању да понуди исти модел за програмирање преко низа различитих хардверских конфигурација.

Неке машине су оптимизоване за научно рачунање, док су друге оптимизоване за пословну употребу. Међутим, пошто сви они деле исти сет инструкција, програми су могли да буду пребачени из машине у машину без поновне компилације (али са могућношћу повећања или смањења у перформансама у зависности од основног хардвера).

Ова врста флексибилности и снаге направили су од *microcoding*-а жељени начин да се изграде нови рачунари.

### 3.1.3. CISC филозофија 2: Редуковање реализационих инструкција

Једна од последица коришћења *microprogrammed* дизајна је да дизајнери изграде већу функционалност у сваком облику наставе. Ово не само да смањује укупан број инструкција потребних за реализацију програма већ је и ефикасније коришћење главне меморије, али у суштини чини језик програматора једноставнијим.

Убрзо, дизајнери су унапредили свој комплет инструкција, посебно на асемблерском језику програматора. Таква побољшања укључују ниске манипулације, специјалне операције *looping*[68] конструкте, као и специјалне модове адресирања за индексирање кроз табеле у меморији.

На пример:

- ABCD - Додај проширени децимални број,
- ADDA - Додај адресу,
- ADDKS - Додај са продужењем,
- ASL - Аритметички шифт леви,
- KAS - Упоредите и замените операције,
- NBCD - Негира децимал са проширеном операцијом,
- EORI - Логичко искључиво или непосредно,
- TAS - Тест операције и сета.

[67] Систем рачунар дизајниран да покрије комплетан асортиман апликација
[68] Метод контроле протока у компјутерским наукама

### 3.1.4. CISC филозофија 3: Директно превођење

Када су дизајнери почели да граде *programer-friendly* инструкције комплета, логичан следећи корак био је да се изграде сетови инструкција које мапе директно преводе на висок ниво језика. Не само да поједностави задатак компајлера, али и омогућава компајлеру да емитује мање инструкција на линији изворног кôда.

Модерни *CISC* микропроцесор, као што је 68000[69], спроведе неколико таквих инструкција, укључујући и рутине за креирање и уклањање оквира са једним позивом.

На пример:

- DBCC - Стање теста, смањења и филијала,
- ROXL - Ротирање са продуженим левим (Extend Left),
- SBCD - Одузимање децимала са продужењем (Extend),
- CMP2 - Упоредити *Register* против горње и доње границе.

### 3.1.5. CISC инструкције

Неке заједничке карактеристике *CISC* инструкција су:

- *2 operate* - формат инструкције, које имају извор и одредиште. На пример, додатак инструкција "*add # 5, D0*" - додаје се број 5 на садржај *D0*, региструје и поставља резултат у регистар *D0*,
- *Региструј се* - Ова инструкција региструје податке и команде у меморију. Више се баве режимом за меморију, укључујући специјализоване режиме за индексирање кроз низове,
- *Променљиве дужине инструкције* - дужина често варира у зависности од адресирања меморије,
- *Инструкције које захтевају више циклуса за извршење* - ако инструкција захтева додатне информације пре него што може да се покрене (на пример, ако процесор треба да се чита у две меморијске локације пре него што делује на њих), прикупљање додатних информација ће захтевати додатне циклусе такта. Као резултат, неке *CISC* инструкције ће потрајати дуже док не одради претходно започете.

[69] Motorola 68000 16/32-битни микропроцесор

#### 3.1.5.1. Нове инструкције

У 1970-им анализе високог нивоа језика назначиле су неке сложене имплементације машинског језика и утврђено је да нове инструкције могу побољшати преформансе. Неке инструкције су додате иако никада није намеравано да се користе у асемблеру, али добро се уклапају са језицима високог нивоа. Компајлери су ажурирани да искористите ове инструкције. Предности семантички богатих инструкција са компактним кôдирањем могу се видети у модерним процесорима, посебно у сегменту високих перформанси где је складиштена централна компонента (за разлику од већине уграђених система). То је зато што су ови брзи, али сложени и скупи, меморија је суштински ограничена у величини, чинећи компактни кôд корисним. Наравно, основни разлог за то је да су главне меморије (*RAM*, односно динамичка данас) и даље споре у односу на (*high performance*) *CPU* језгра.

#### 3.1.5.2. Дизајн проблеми

Док су многи пројекти постигли циљвећу пропусну моћ по нижој цени, а такође је дозвољен језик високог нивоа конструкције (да изражава мање инструкција), примећено је да то није увек случај. На пример, јефтине верзије комплексних архитектура (тј коришћење мање хардвера) може да доведе до ситуација у којима је било могуће да се побољшају преформансе не користећи сложену инструкцију (као што је поступак позовите или унесите инструкцију), али уместо тога користите низ једноставнијих наредби.

Један од разлога за то је да архитекте (*microcode* дизајнери) понекад "превише" дизајнирају наредби асемблеру, односно укључујући и карактеристике које није било могуће ефикасно имплементирати на основном хардверу на располагању. То би могао, на пример, бити "нежељени ефекат", као што је постављање регистара или меморијске локације која је ретко коришћена; ако је то урађено путем обичне интерне магистрале, или чак екстерном магистралом, захтевало би додатне циклусе сваки пут и на тај начин било прилично неефикасно.

Чак и у балансираним високих преформанси дизајна, високо кодирана и (релативно) на високом нивоу наредба може бити компликована да се декодира и извршава ефикасно у оквиру ограниченог буџета транзистора. Стога је у таквим архитектурама потребно много рада на делу дизајнера процесора у случајевима где је једноставније, али (обично) спорије, решење на основу декодирања табелама и/или *microcode* пултом није примерено. У време када су транзистори и остале компоненте били ограничени ресурс, ово је такође оставило мање компоненти и мање могућности за друге врсте оптимизације перформанси.

### 3.1.6. Хардверске архитектуре

Заједничке карактеристике већине *CISC* хардверских архитектура су:

- *Комплекс декодирање инструкција логике* - једном инструкцијом је могуће подржати више начина адресирања,
- *Мали број регистара опште намене* - у овим регистрима налазе се инструкције које могу директно да раде на меморији и ограничене количине чипова простора, али који нису посвећену декодирању и извршењу инструкција и *microcode* складиштењу,
- *Неколико регистара специјалне намене* - у овим регистрима се налазе инструкције које врше одређене радње, нпр. посебни регистри за *stack pointer*[70]. То може донекле поједноставити дизајн хардвера,
- *"Стање кôд" регистар* - подешен је као споредни ефекат већине инструкција. Овај регистар одражава резултат последње операције - што је мање него, једнако или веће од нуле, а записи о грешци ако постоје јављају се.

### 3.1.7. Идеалне CISC мшине

*CISC* процесори су дизајнирани тако да потпуно изврше сваку инструкцију пре почетка следеће, већина процесора извршавају инструкције у одређеним фазама - чим се заврши једна фаза, процесор доноси резултат и прелази у следећу фазу.

#### 3.1.7.1. Фазе извршавања инструкција

- *Настава је преузета из главне меморије*,
- *Инструкција се декодира* - Контролни кôд из *microprogram*-а идентификује тип операције које треба извршити, где се могу наћи подаци о којима се врше операције и где ставити резултат. Ако је потребно, процесор чита додатне информације из меморије,
- *Инструкција се извршава* - Контролни кôд из *microprogram*-а одређује коло/хардвер који ће обављати операцију,
- *Резултати се пишу на меморији.*

У идеалним *CISC* машинама свака комплетна инструкција би захтевала само један циклус такта (што значи да ће се свака фаза завршити у делићу циклуса). У ствари, то је максимална могућа брзина за машине која извршава инструкцију, једна у једном тренутку.

[70] Указује на тренутни врх стека

### 3.1.8. CISC машине

Посматрано реално, неке инструкције могу захтевати више од једног такта по циклусу. Међутим, *CISC* дизајн може да толерише ово успоравање узимајући у обзир идеју да укупан број малих циклуса рашава компликоване ствари у сваком циклусу. Уобичајена једначина за одређивање перформанси је збир свих инструкција (број циклуса инструкција по инструкцији * време циклуса) = време извршавања. Ово омогућава да се убрза процесор на 3 различита начина користећи мање инструкције за одређени задатак, при чему се смањује број циклуса неких инструкција, или убрза такт (смањење времена циклуса).

### 3.1.9. Предности и мане CISC архитектуре

#### 3.1.9.1. Предности CISC архитектуре

У време њиховог почетног развоја *CISC* машине користиле су расположиве технологије ради побољшања перформанси рачунара.

*Microprogramming*[71] је једноставан за коришћење као асемблерски језик.

Лакоћа *microcoding*[72] нових инструкција дозвољава дизајнерима да направе *CISC* машине навише компатибилне: нови рачунар могао да ради исте програме као што је раније рачунарима, јер би нови рачунар садржи надскуп инструкцијама ранијих рачунара.

Како свака инструкција добија све више могућности, мањи број инструкција се може користити за имплементацију датог задатка. То је ефикасније коришћење релативно споре главне меморије.

Зато што *microprogram* сетови инструкција могу бити написани да се подударају са конструкцијом језика високог нивоа, преводилац не мора бити компликован.

[71] Писање microcode-а
[72] Секвенца микро инструкција

#### 3.1.9.2. Недостаци CISC архитектуре

Ипак, дизајнери су убрзо схватили да је *CISC* филозофија имала своје проблеме, укључујући:

- Раније генерације једне породице процесора обично су садржани као подскуп у свакој новој верзији, тако су скуп инструкција и хардвер постали сложенији са сваком генерацијом рачунара,
- Тако да што је више инструкција могуће сачувати у меморији уз најмање заузетог простора, могла би бити појединачни инструкције од скоро било које дужине, то значи да ће различите инструкције имати различиту количину такт времена да изврше, смањујући укупне перформансе машине,
- Многе специјализоване инструкције се не користе довољно често да оправдају своје постојање, приближно 20% расположивих инструкција се користе у типичном програму,
- *CISC* инструкције обично постављају услов кôдова као споредни ефекат инструкције. Не само да постављају услов потребног времена кодова, него програмери треба да имају на уму да испитају услов кôда битова пре промене наредне инструкције.

## 3.2. RISC процесори - Reduced Instruction Set Computer

Смањени сет инструкција рачунарства, или *RISC*, је *CPU* дизајн стратегија која се заснива на увиду да поједностављена (за разлику од комплекса) инструкција може да пружи боље перформансе ако јој та једноставност омогућава много брже извршење сваке инструкције. Рачунар заснован на овој стратегији који има смањени сет инструкција се такође назива *RISC* рачунар.

Слика 21. Sun UltraSPARC, RISC microprocessor

Разни предлози су учињени у погледу прецизне дефиниције *RISC*-а, али општи концепт је систем који користи мали, високо оптимизован скуп инструкција, а не више специјализовани скуп инструкција који се често налазе у другим врстама архитектуре. *RISC* системи користе учитавање/складишћење архитектуру - *load/store architecture*[73]. Супротстављена архитектура је позната као сложени скуп инструкција, односно *CISC*.

### 3.2.1. Историја и развој

Број система, да се вратимо у 1970-е (па чак и 1960-е) су пренета као прва *RISC* архитектура, делимично на основу њиховог коришћења учитавање/складишћење приступа. Израз *RISC* је сковао Дејвид Патерсон на *Berkeley RISC* (један од два истраживачка пројекта) пројекату који је започео 1980-е, иако је донекле сличне концепте имао пре него што су се појавили.

*Berkeley RISC* је заснован на стицању перформансе кроз употребу *pipelining*[74](снабдевања) и агресивном коришћењу технике познате као *register windowing*[75]. У традиционалном *CPU*-у, један има мали број регистара, а програм може користити било који регистар у било ком тренутку. У процесору са *register windowing*, постоји огроман број регистара, нпр. 128, али програми могу да користе само мали број њих, нпр. осам, у било ком тренутку. Програми који се ограничавају на осам регистара по процедури могу врло брзо доћу до процедуре *calls* (позиви): Позив једноставно помера прозор "доле" од осам, на скуп од осам регистра се користи тај поступак, а повратак се враћа кроз прозор *Berkeley RISC* пројекат који се испоручио *RISC-I* процесор у 1982.

[73] Дозвољава да меморија може приступити учитавању и складишту пословања, а све вредности за операције треба да се учита у меморију и бити присутна у регистрима
[74] Је скуп елемената за обраду података повезаних у серији, тако да излаз једног елемента је улаз у следећи
[75] Је техника да се побољша учинак, посебно заједничке операције

У раним 1980-им година, значајне неизвесности опколиле су *RISC* концепт, и било је неизвесно да ли то може имати комерцијалну будућност, али од средином 1980-их концепти су довољно сазрели да се види као комерцијално одржива. У 1986 *Hewlett Packard* су почели да користе рану имплементацију свог *PA-RISC* (сет инструкција) у неким од њихових рачунара. У међувремену, *Berkeley RISC* напор је постао толико познат да је на крају постао назив за цео концепт и *Sun Microsystems* (компанија која је продавала компјутере) је почела 1987. испоруке система са *SPARC* процесорима и директно на основу *Berkeley RISC-II* система.

Познати *RISC* породице укључују *DEC Alpha, AMD 29k, ARC, ARM, Atmel AVR, Blackfin, MIPS, PA-RISC, Power (*укључујући *PowerPC), SuperH, and SPARC*. У 21. веку, употреба процесора *ARM* архитектуре у смарт телефонима и таблет рачунарима као што су *iPad* и *Android* таблет обезбеђује широку базу корисника засноване за *RISC* системе. *RISC* процесори се такође користе у супер компјутерима, као што су *K* рачунар, најбржи на *TOP500* листи у 2011, а други на листи 2012.

### 3.2.2. RISC идеја

Кола која обављају активности које је дефинисао *microcode* у многим (али не свим) *CISC* процесорима је, само по себи, процесор који у много чему подсећа на структуру врло раног *CPU* дизајна. У раним 1970-им то је довело до идеје да се врате у једноставнији дизајн процесора како би било више изводљиво да се носе без (тада релативно великих и скупих) *ROM* табела и/или *PLA* структура за редослед и/или декодирања. Први *RISC* процесор означен (*IBM 801 - IBMs Watson* истраживачки центар, средином 1970-их) је чврста линија снабдевања, једноставна машина првобитно намењена да се користи као унутрашње *microcode* језгро или мотор, у *CISC* дизајну, али и постао процесор који је увео *RISC* идеју нешто већој јавности. Једноставност и правилност у видном сету инструкција омогућава процесору да лакше спроведе и преклапа фазе (*pipelining*) на нивоу машинског кôда, међутим, (односно ниво виђен компајлером) *pipelining* на том нивоу је већ у употреби у неким *CISC* процесорима високих перформанси како би се смањило време циклуса инструкција (упркос компликацијама примене у оквиру ограниченог број саставних делова и сложености повезивања и изводљивости у исто време). Интерно *microcode* извршење у *CISC* процесору, с друге стране, може бити више или мање преклапање у зависности од конкретног дизајна, а самим тим мање-више личи на основну структуру *RISC* процесора.

### 3.2.3. Сет инструкција

Заједничко неразумевање фразом "*Reduced Instruction Set Computer*" (смањени сет инструкција рачунара) је погрешна идеја да се инструкције једноставно елиминишу, што је резултирало мањим скупом инструкција. У ствари, током година, сетови *RISC* инструкција су нарасли у величини, и данас многи од њих имају већи скуп инструкција од многих *CISC* процесора. Неки *RISC* процесори, као што су *PowerPC* имају инструкцију постављања велику као, кажу, *CISC IBM System/370*[76], и обрнуто, *DEC PD*-8[77], јасно *CISC* процесори јер многе од његових инструкција укључују више меморијских прилаза, има само 8 основних инструкција, плус неколико проширених инструкција.

Термин "смањио" у тој фрази је требало да описује чињеницу да је количина рада била једна инструкција које се смањује највише један циклус-меморија у односу на "комплекс" од *CISC* инструкција процесора који могу да захтевају десетине података меморијских циклуса како би извршавали једну инструкцију. Конкретно, *RISC* процесори обично имају одвојене инструкције за *I/O*[78] и обраду података.

## 3.3. CISC и RISC услови

Услови *CISC* и *RISC* добили су мање смисла са сталном еволуцијом оба *CISC* и *RISC* дизајна и имплементације. Први високо (или добро) снабдевени имплементирани процесори *x86, 486* су пројекти компаније *Intel, AMD, Cyrix* и *IBM*-а и подржавају сваку инструкцију које су њихови претходници урадили, али постиже максималну ефикасност само на прилично једноставном *x86* подскупу који је само нешто више од типичног *RISC* сета инструкција (тј. без оптерећења типичним *RISC* - продавница ограничења). *Intel P5 Pentium* генерација био је унапређена верзија ових принципа. Међутим, модерни *x86* процесори такође (типично) декодирају и деле инструкције у динамичке низове интерно упрошћених микрооперација, које не само да помажу да изврши већи подскуп инструкција у *pipelined* (преклапање) моду, али и олакшава издвајање више напредних паралелизма од *code* потока, за још веће перформансе.

Насупрот популарном поједностављењу, нису сви *CISC* процесори микрокодирани или имају "сложене" инструкције. Како је *CISC* постао кутија за све термин који значи било шта не односи се на складиштење (*RISC*) архитектуре, то није број инструкција, нити сложеност имплементације, који дефинишу *CISC*, али чињеница да аритметичке инструкције такође обављају прилаз меморији. У поређењу са малим 8 битним *CISC* процесором, *RISC floating point* инструкција је сложена. *CISC* чак и не морају да имају комплексне начине адресирања, 32 или 64 битни *RISC* процесори могу да имају више сложених начина адресирања од малих 8 битним *CISC* процесора.

[76] Велики рачунарски систем
[77] Први успешан комерцијални миникомпјутер
[78] input/output - Комуникација између система за обраду података

*A PDP-10, a PDP-8, an Intel 386, an Intel 4004, a Motorola 68000, a System z mainframe, a Burroughs B5000, a VAX, a Zilog Z80000,* и *a 6502* све дивље варирају у броју, величини и формату инструкција; броју, врсти и величини регистара, као и расположивим типовима података. Неке имају хардверску подршку за операције попут скенирања низова, произвољно прецизној *BCD* аритметици, или трансцедентних функција, док други имају само 8 битну функцију сабирања и одузимања. Али они су сви у *CISC* категорији, јер они имају "оптерећење" које учитава наредбу и/или складишти меморију садржану у оквиру истих инструкција које обављају стварне калкулације. На пример, *PDP-8*, има само 8 фиксних дужина инструкција и не садржи *microcode*, али због начина рада инструкције је *CISC, PowerPC*, која има преко 230 инструкција (више од неких *VAXes*) и сложене унутрашњости попут преименовања регистра и редослед *buffer*[79]-а је *RISC*, док најмањи *CISC* има 8 инструкција, али је јасно *CISC* јер комбинује меморијски приступ и рачунање у истим инструкцијама.

Неки од проблема и противречности у овој терминологији ће можда нестати као виши системски услови, као што су (не) учитавање/складиштење, постаје популаран и на крају бива замењен са мало непрецизним и контра-интуитивни *RISC*/*CISC* условима.

[79] Меморија која се користи за чување привремено излазних или улазних података

# 4. Значајни примери CISC архитектуре

Свакако значајан пример *CISC* архитектуре је *Intel*-ова (и AMD-ова) *80x86* фамилија, а овакав дизајн су користили и *CDC 6600, z/Architecture, System/360, VAX, PDP-11* и *Motorola 68000*.

## 4.1. CDC 6600

*CDC 6600* је главни рачунар из *Control Data Corporation*, први испоручен 1964. на *Lawrence* Radiation Laboratory, део Калифорнијског универзитета у Берклију. Првенствено се користи за високе енергетске нуклеарне физике истраживања, посебно за анализу нуклеарних догађаја снимљених унутар *Alvarez* мехур комори. Први *CDC 6600* је испоручен око годину дана раније на *Conseil Européen pour la Recherche Nucléaire (CERN)* код Женеве у Швајцарској за употребу у високо енергетска истраживања нуклеарне физике. Сматра се да је први успешни суперкомпјутер, надмашујући свог најбржег претходника *IBM 7030 Stretc*, око три пута. Са радом око 1 megaFLOPS[80], остао је најбржи рачунар на свету од 1964-69, када је препустио тај статус његовом наследнику, *CDC 7600*.

Слика 22. CDC 6600 system consol

## 4.2. IBM System/360

*IBM System/360* (S/360) је главни компјутерски систем представљен од стране *IBM*-а 7. априла 1964. То је прва фамилија рачунара дизајнираних да покрију комплетан асортиман апликација, од малих до великих, и комерцијалне и научне. Дизајн је направио јасну разлику између архитектуре и имплементације, што омогућава *IBM*-у да објави пакет компатибилних дизајна по различитим

Слика 23. IBM 704 главни рачунар (1964)

[80] Је мера перформансе рачунара, посебно у области научних прорачуна који чине тешку употребу децималних калкулација

ценама. Сви, али и најскупљи, системи користе *microcode* да спроведу сет инструкција, који карактерише 8 битни byte[81] адресирном и бинарном, децималним и покретним зарезом калкулације.

Најспорији *System/360* модели најављени у 1964. имали су брзину од 0.0018 до 0.034 *MIPS*[82], *System/360* најбржи модели били су око 50 пута бржи са 8kB и до 8MB интерне главне меморије, мада ово друго је било необично, и до 8MB спорије *Large Core Storage - LCS*[83]. Велики систем може имати мало, 256kB од главне меморије, али је 512kB, 768kB или 1024kB више заступљено.

*360s су* били изузетно успешни на тржишту, омогућавајући корисницима да купе мањи систем са знањем да ће увек бити у стању да мигрирају на горе, ако су њихове потребе расле, без репрограмирање апликативног софтвера. Дизајн је по многима један од најуспешнијих у историји рачунара, који утиче на дизајн рачунара у годинама које долазе.

Примена ниво компатибилности (са неким ограничењима) за *System/360* софтвер се одржава до данашњег дана са *IBM*-овим рачунарима *zSeries*[84].

## 4.3. z/Architecture

*z/Architecture*, у почетку и на кратко звана *ESA Modal Extensions - ESAME*, односи се на 64 битну *IBM*-ову архитектуру за *IBM*-ове главне рачунаре. *IBM* је представио своју прву *z/Architecture*-базиран систем, *zSeries* модел 900, крајем 2000. Касније *z/Architecture* системи укључују *IBM z800, z990, z890, System z9, System z10, zEnterprise 196,* и *zEnterprise 114*. *z/Architecture* задржава компатибилност са претходним 32 bit data/31 bit адресирањем *ESA/390* архитектуре и њених претходника, све назад на 32 bit data/24 bit адресирања *System/360*.


Слика 24. IBM zSeries 800

Већина оперативних система, укључујући *z/OS* ограничава извршавања кôда на првих *2GB (*31 bits) сваког виртуелног адресног простора из разлога ефикасности и компатибилности уместо архитектонског ограничења. Имплементација виртуелне меморије - *Virtual memory*[85] *z/OS*[86]-а подржава више адресних простора од 2GB, дозвољавајући више од 2GB истовремено задржавања

[81] Је јединица дигиталних информација у рачунарству и телекомуникацијама које се најчешће састоји од осам битова
[82] Million instructions per second
[83] Је компонента модела IBM System/360
[84] Је марка од стране IBM-а свим својим главним рачунарима у 2000.
[85] Је техника управљања меморијом развијена за вишеструко извршавање процеса у истом временском периоду
[86] 64-bit operating system за главне рачунаре

програмског кôда. 64 битна верзија *Linux*-а на *System z* омогућава извршавање од 64 битне адресе.

*z/VSE* верзија 4, *z/TPF* верзија 1 и *z/VM* верзија 5 оперативни системи, и вероватно њихови наследници, захтевају *z/Architecture*.

*Architecture* подржава покретање више истовремених оперативних система и апликација, чак и ако они користе различите начине адресирања. Ово омогућава програмерима да изаберу режим адресирања који је најповољнија за апликације и структуре података.

## 4.4. PDP-11

*PDP-11* је била серија 16-битних миникомпјутера које су продавали *Digital Equipment Corporation (DEC)* од 1970 до 1990, један од низа производа у *PDP* серији. *PDP-11* заменио *PDP-8* у реалним апликацијама - *real-time applications*[87], мада обе линије производа трају паралелно више од 10 година. *PDP-11* је имао неколико јединствено иновативних карактеристика, био је лакши за програмирање од својих претходника са коришћењем општих регистара. Његов наследник средњег опсега миникомпјутер *niche* је 32 bit-ни *VAX-11*.


Слика 25. PDP-11/40

Дизајн карактеристика *PDP-11* утицао на дизајн микропроцесора, као што су *Motorola 68000*, дизајн својих оперативних система, као и друге оперативне системе из *Digital Equipment*-а, утицао на дизајн других оперативних системима као што је *CP/M*[88] а тиме и *MS-DOS*. Прва званично именован верзија *Unix*[89]-а ради на *PDP-11/20* у 1970. Најчешће се наводи да је *C* програмски језик искористио предност од неколико *PDP-11* зависних програмских функција ниског нивоа, иако нису оригиналног дизајна.

[87] је студија хардверских и софтверских система који су предмет " real-time ограничења"
[88] Масовно тржиште оперативног система направљен за Intel 8080/85
[89] Је мултитаскинг, вишекорисnички оперативни систем првобитно развијен 1969

## 4.5. VAX

*VAX* (*Virtual Address eXtension*) је успостављена линија средње класе серверских рачунара настали средином 1970-их. 32 битни сложени рачунарски сет инструкција (*CISC*) *ISA*, је дизајниран да прошири или замени *DEC* поједине програмиране податке процесора (*PDP*) *ISA*. *VAX* назив је коришћен од стране *DEC* за породицу рачунарских система заснованих на овој процесорској архитектури.

Следи *DEC PDP-11* у 1978-ој и уводи нови оперативни систем, *VMS* (*Virtual Memory System*). *VAX* је укључио 32 битни процесор и виртуелну меморију. Историјски гледано, *VAX* се такмичио са бројем *Hewlett Packard* и *IBM* рачунара у малом предузећу и универзитетско научном тржишту.

У ранијим временима, ова величина и распон цена од рачунара била је позната као миникомпјутер. Данас, *VAX* и његови конкуренти продају "сервере" за пословне мреже које користе *client/server*[90] модел рачунара.

## 4.6. Motorola 68000 family

Слика 26. Motorola 68000 процесор

*Motorola 680x0/m68000/68000* је породица 32 битних *CISC* микропроцесора. Током 1980-их и раних 1990-их, они су били популарни у персоналним рачунарима и радним станицама, а били су главни конкуренти за *Intel's x86* микропроцесоре. Они су највише били добро познати као напајање процесора раног *Apple Macintosh*-а, на *Commodore Amiga*-и, на *Sinclair QL*, на *the Atari ST, SEGA Megadrive/Genesis* и још неколико других(персонални рачунари). Иако не постоје модерни, десктоп рачунари су засновани на *68000*, изведени процесори су још увек у широкој употреби у уграђеним апликацијама.

[90] Описује однос између два компјутерска програма

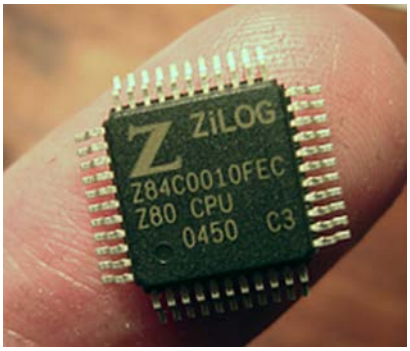
# 5. Intel 8086 - 80x86

*8086* (такође зван *iAPX 86*) је 16 битни микропроцесор чип дизајниран од стране компаније *Intely* у периоду од почетка 1976 до средине 1978, када је пуштен у производњу. *8086* је дао повод за *x86* архитектуру будућим интеловим процесорима. *Intel 8088*, објављен 1979, био је незнатно измењена чип са 8 bit *data bus*[91] - спољном магистралом података (омогућава коришћење јефтинијих и мање логичких чипова који подржавају), а приметан је као процесор коришћен у оригиналном *IBM PC*-у.

Слика 27. Површински носач Intel 80386SX процесора

## 5.1. Први x86 дизајн

*8086* пројекат је почео у мају 1976 и био је првобитно замишљен као привремена замена за амбициозни и одложени *iAPX 432*[92] пројект. То је био покушај да се скрене пажња са мање одложених 16 и 32 битних процесора других произвођача (као што су *Motorola, Zilog* и *National Semiconductor*), а у исто време да би се супротставили претњи од *Zilog Z80* – 8 bit *microprocessor* (дизајниран од стране бивших запослених у *Intel*-у), који су постали веома успешни. И физичка и архитектура чипова су стого развијени веома брзо од стране мале групе људи, и користе исте основне елементе микроархитектуре - *microarchitecture*[93] и физичке имплементационе технике као запослени за мало старије *8085* микропроцесоре.

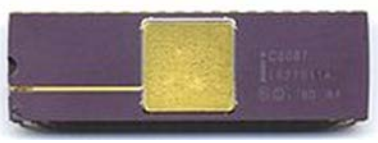


Слика 28. Z80

Рекламиран као извор компатибилни – *source-compatible*[94], *8086* је дизајниран да омогући асемблерском језику - *Assembly language*[95] за *8008, 8080* или *8085* да се аутоматски конвертују у еквивалентном *8086* изворном кôдом, са мало или нимало ручног едитовање. Програмски модел и скуп инструкција је (широко) базиран на *8080* да би ово било могуће. Ипак, *8086* дизајн је проширен да подржи пуну 16 битну обраду, уместо на прилично основним 16 битним могућностима за *8080/8085*.

[91] Је подсистем који преноси податке између компонената унутар рачунара или између рачунара
[92] Је био веома амбициозан мулти чип микропроцесор архитектуре уведен у 1981
[93] Је скуп инструкција архитектуре имплементиран на процесору
[94] Рачунар који може да покрене исти изворни код намењен да буде састављен и покренути на другом рачунару
[95] Је ниског нивоа програмски језик за рачунаре, микропроцесоре, микроконтролере и друге програмабилне уређаје

Нове врсте инструкција су додате као пуна подршка за учитане целобројне вредности, база + померај адресирање, и саме понављајуће операције биле су налик на *Z80* дизајн, али су сви били направљени нешто општије у *8086*. Инструкције директно подржавају груписане *ALGOL* - породице језике, као што су *Pascal* и *PL/M* такође додати. Према речима главног архитекте *Stephen P. Morse*, резултат је више средишњег софтверског приступа него у дизајну ранијих *Intel*-ових процесора (дизајнери су имали искуство у раду са имплементацијом компајлера). Остала побољшања укључују *microcoded* умножавање и деле инструкције и *bus* структуру боље прилагођену будућим сарадницима процесора (као што су *8087* и *8089*) и вишепроцесорски систем.



Слика 29. Intel 8087

Прва ревизија сета инструкција и на високом нивоу архитектуре је спремна после три месеца, и готово ниједан *CAD* алат није коришћен, четири инжењера и 12 распоређених људи истовремено ради на чипу. *8086* је требало нешто више од две године од идеје до радног производа, који је сматран прилично брзим за комплексне дизајне у 1976-1978.

*8086* радослед користи мешавину случајне логике и *microcode*-а и реализован је коришћењем трошења оптерећења *nMOS* кола са око 20.000 активних транзистора - *transistor* (29,000 рачунајући све локације *ROM*-а и *PLA*[96]). Убрзо је премештен на нови предефинисан *nMOS* производног процеса званог *HMOS* (за високе перформансе *MOS*) које је *Intel* првобитно развио за производњу брзих статичких *RAM* производа. Ово је било праћено *HMOS-II, HMOS-III* верзијама, и на крају, потпуно статичана *CMOS*[97] верзија за уређаје напајаним батеријама, произведен помоћу *Intel's CHMOS* процеса. Оригинални чип мери 33 mm² и минимална величина чипа је 3.2 μm.

Слика 30. TO-3, TO-126, TO-92, SOT-23

[96] Је врста програмабилног логичког уређаја који се користи за имплементацију логичких комбинаторних кола

[97] Је технологија за изградњу интегрисаних кола

## 5.2. Детаљи

### 5.2.1. Подсистеми и операције

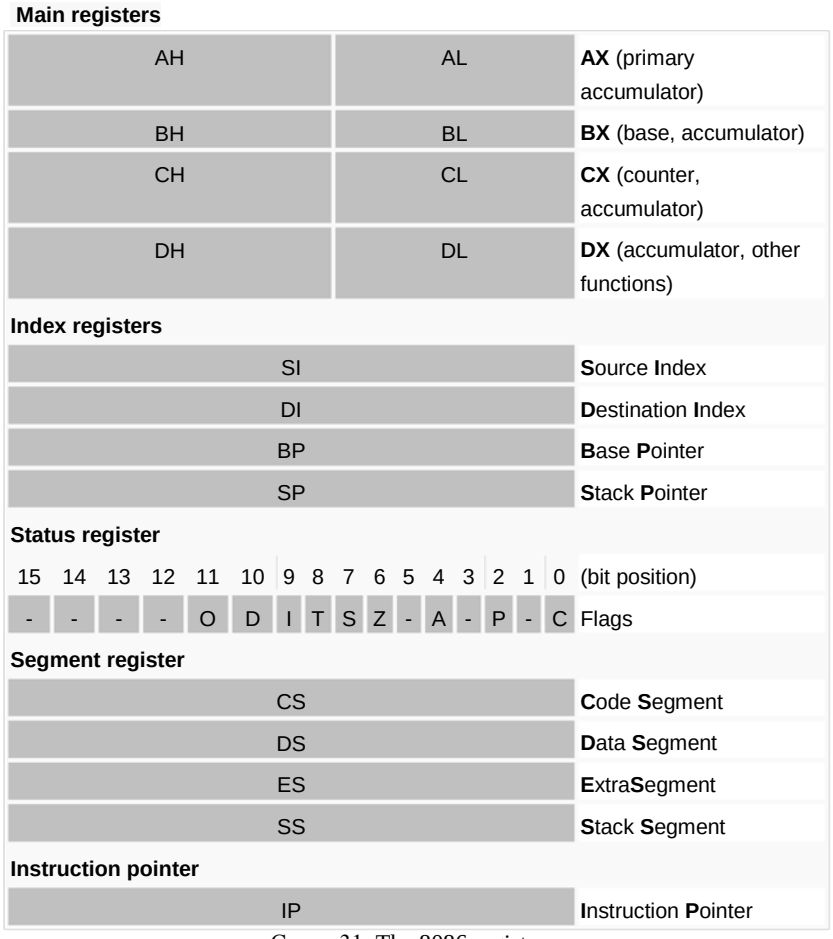
Сви интерни регистри, као и интерни и екстерни подаци су подсистемски, 16 бита, чврсто успостављање "16 битних микропроцесора" идентичних као *8086*. 20 битна спољни адреса подсистема даје 1MB физичког адресног простор ($2^{20}$ = 1,048,576 bytes). Овај адресни простор је упућен путем интерне 'сегментације'. Подаци подсистема су мултиплексирани са адресним магистралама, како би се уклапали у стандардне 40-пин дупле линијске пакете. 16 битне *I/O* адресе значи 64KB од посебног *I/O* простора ($2^{16}$=*65,536*). Максимални линеарни адресни простор био је ограничен на 64KB, једноставно зато што су интерни регистри били само 16 бита. Програмирање преко 64KB граница укључује подешавање сегмента регистара и тако остао док *80386* није представио сложенији хардвер управљања меморијом.

Неки од контролних пинова, које носе неопходне сигнале за све спољне операције, имају више од једне функције у зависности да ли уређај извршава операције у најспоријем или најбржем режиму. Бивши је намењен малим системима са једним процесором, док је други за средње или велике системиме, користећи више од једног процесора.

## 5.2.2. Регистри и инструкције

*8086* је имао осам (више или мање уопштено) 16 битних регистра укључујући стек поинтер - *stack pointer*, али искључујући показивач инструкција, заставу регистар - *flag register*[98] и сегмент регистра. Четири од њих, *AX*, *BX*, *CX*, *DX*, такође се може приступити као два пута онолико 8 битних регистара , док друга четири, *BP*, *SI*, *DI*, *SP*, били су само 16 битни.

Захваљујући компактном кодирању инспирисан 8 битним процесорима, већина инструкције су биле једно адресне или дво адресне операције, што значи да је резултат ускладиштен у једном од операнада. Највише један од операнада може бити у меморији, али та операнд меморија може бити дестинација, док други операнд, извор, може бити регистар.



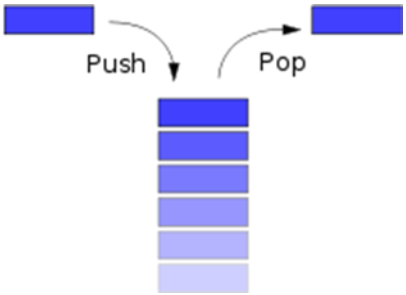
Слика 31. The 8086 registers

Једна меморијска локација може често да се користи и као извор и одредиште које, између осталих фактора, додатно је допринела густини кôда - *code density*[99] упоредива са највише 8 битним машинама.

Иако је степен уопштености већине регистара био много већи него у *8080* или *8085*, још увек је прилично слаб у односу на типичне савремене миникомпјутере, а регистри су такође понекад коришћени, обухваћени инструкцијама. Док савршено разумно за скуп програмера, овај компликовани регистар додатак за компајлере у односу на редовне 16 и 32 битне процесоре, као што је *PDP-11*, *VAX*, *68000*, итд, а са друге стране, у поређењу са полусавременим једноставним (али популарним и свуда присутним) 8 битни микропроцесорима, као што су *6502* и *6809*, или *8085*, то је било знатно лакше генерисати кôд за дизајн *8086*.

[98] Or status register је колекција flag bits-а за процесор
[99] Важна карактеристика било ког скупа инструкција

*8086* садржану 64KB од 8 бита у *I/O* простору. 64KB (један сегмент) *stack*[100] који расте ка нижим адресама је подржан од стране хардвера, речи од 2 byte-а су дстигли на *stack* и *stack* врх је истакао *SS:SP*. Постоје 256 прекида, који се могу покренути од стране хардвера и софтвера. Прекиди могу опадати, користећи *stack* за чување адресе повратка.



Слика 32. Једноставно репрезентација stack-a

## 5.2.3. Типови инструкција

### 5.2.3.1. Stack инструкција

*x86* архитектура има хардверску подршку за механиазм извршења слагања. Инструкције попут *push, pop, call* и *ret* се користи за правилно постављење *stack*-ом да прође параметре, да издвоје простор за локалне податке, као и да сачува и врати поене. *Ret* величина инструкције је веома корисна за ефикасно попуњавање простора (и брзо) *calling convention*[101] где су позиваоци одговорани за регенерисање *stack* простора заузет параметрима.

Приликом подешавања *stack* оквира за поседовања локалних података обновљивих процедура постоји неколико избора, висок улазни ниво инструкција узима процедуру дубинског убацивања аргумента, као и локалну величину аргумента, а може бити и бржи од више експлицитног манипулисања регистра (као што су *push bp, mov bp, sp, sub sp, size*), али се углавном не користи. Да ли је брже зависи од конкретног спровођења *x86* (тј. процесора), као и позивне конвенције и шифра намењена да ради на више процесора, обично ће радити брже на већини циљеве без њега.

Комплетан асортиман адресарских режима (укључујући *immediate* и *base+offset*), чак и за инструкције као што су *push* и *pop*, чини директно коришћење *stack*-а за *integer, floating point* и *address data*[102] једноставним, као и задржавање *ABI*[103] спецификације и механизма релативно једноставним у поређењу са неким *RISC* архитектурама.

[100] Је последњи улаз, први излаз апстрактног типа податка и линеарне структуре података
[101] Је шема колико подпрограми примају параметра од свијих позиваоца и како да се врати резултат
[102] Подаци који се користи на различитим нивоима софтвера и хардвера за приступ складишта примарне меморије рачунара
[103] Описује низак ниво интерфејса између компјутерског програма и оперативног система или неки другог програма

#### 5.2.3.2. Floating point инструкција

*x86* скуп језика садржи инструкције за *stack* заснованог на *floating point* јединице. Они укључују сабирање, одузимање, негирање, множење, дељење, остатак, квадратних корена, цела одузимања, децимална одузимања. Операције обухватају конверзије инструкција које се учитавају или чувају неку вредност из меморије у неки од следећих формата: бинарни кодирани децимални, 32 битни цео број, 64 битни цео број, 32 битни децимални, 64 битни децимални или 80 битни децимални (на учитавању, вредност се претвара у тренутно користећи децимални мод). *x86* такође укључује бројне функције синуса, косинуса, тангенте, аркустангенс, степеновање са основом 2 и логаритми на основама 2, 10, или *e*.

Као целих бројева, први операнд је и први извор операнда и одредиште операнда. *fsubr* и *fdivr* треба издвојити као прву замену изворних операнада пре обављања одузимање или подела. Сабирање, одузимање, множење, дељење, чување и поређење инструкција обухватају инструкција режиме који ће се појавити на врху *stack*-а након што је њихов рад је завршен.

#### 5.2.3.3. Инструкција за манипулисање података

*x86* процесор такође укључује комплексне начине адресирања за суочавање меморије са хитним померајем, регистром, регистром са померајем, регистар промењене величине са или без помераја, и регистар са могућим померајем и други регистар промењене величине. Тако, на пример, један регистар може да кодира *mov eax*, као једна инструкцији која учитава 32 битни податак са адресе обрачунатог као померај из *ds selector*, а складишти га у *eax* регистар.

*x86* сет инструкција укључује мрежно учитавање, складиштење, премештање, скенирање и упоређивање инструкција (*lods*, *stos*, *movs*, *scas* и *cmps*) које обављају сваки рад на одређеној величини (*b* за 8 bit byte, *w* за 16 bit word, *d* за 32 bit double word) онда повећава/умањује (зависно од *DF*, смера *flag*-а) имплицитно адресном регистару (*si* за *lods*, *di* за *stos* и *scas*, и оба за *movs* и *cmps*). За учитавање, складиштење и скенирање операција, имплицитно мета/извор/поређење регистар је у *al*, *ax* или *eax* регистару (у зависности од величине). Имплицитни сегмент регистар који се користи је *ds* за *si* и *es* за *di*. *cx* или *ecx* регистар се користи као опадајући бројач, а операција се прекида када бројач достигне нулу или када је неједнакост откривена.

*Stack* је имплементиран уз посредно смањење (*push*) и повећавања (*pop*) *stack* поинтера. У 16 битном моду, овај имплицитни *stack pointer* третирамо као SS:[SP], у 32 битном режиму је SS:[ESP], а у 64 битном режиму је [RSP]. *Stack* показивач заправо указује на последњу вредност која је ускладиштена, под претпоставком да ће његова величина одговарати оперативоми режиму процесора (тј., 16, 32, или 64 бита), да одговарају ширини подразумеваних *push*/*pop*/*call*/*ret* инструкцијама. Такође, укључене су инструкције улаза и излаза које одвајају и уклањају податке са врха *stack*-а, док се поставља оквир *stack pointer*-а у *bp*/*ebp*/*rbp*.

#### 5.2.3.4. MOV инструкција

У *x86* асемблеру, *MOV* инструкција је помоћник за копирање података са једне локације на другу. *x86* асемблер има неколико различитих move-померај инструкција. У зависности од тога да ли је програм у 16 битном или 32 битном кôд сегменту (у заштићеном режиму) и да ли је прекидач за инструкције у користи, *MOV* инструкција може пренети 8 бита, 16 бита или 32 бита података (или 64 бита у x86-64 моду). Подаци могу бити копирани и из меморије и регистара.

Реч померај за ове операције је, строго говорећи, погрешан назив: има мало везе са физичким концептом кретање објеката од А до В, са места А и постаје празан; *MOV* уместо да прави копију стања објекат на А и замени старо стање В у овом процесу. Ово се огледа у неким другим асемблерима помоћу речи као *load, store* или *copy* уместо *move.*

#### 5.2.3.5. PUSH и POP инструкције

Ове инструкције се користе за смештање/узимање података у/из магацина. Постоји укупно шест форми инструкција *PUSH* и *POP*, а то су оне које манипулшу са регистрима, меморијом, непосредном вредношћу, сегментним регистрима, маркерима и свим регистрима. Инструкције *PUSH imm/POP imm* као и *PUSHA/POPA* доступне су само код *80286 Pentium*-а. Поред стандардних инструкција *PUSH* треба уочити следеће инструкције:

1. PUSHA (push all) копира у магацин садржај интерног регистарског скупа са изуетком сегментних регистара,
2. PUSHF (push flags) копира у магацин садржај маркер регистара (flag registar)
3. PUSHAD  копира у магацин садржај 32 битног регистарског скупа.

Инструкција типа *POP* обавља инверзну операцију од *PUSH*. Код *8086, 80286* избавља податак из магацина и смешта га у циљни 16 битни регистар, сегментни регистар, или 16 битну меморијску локацију, док микропроцесори *80386 Pentium* избављају 32 битни податак из магацина и користе 32 битну адресу. *POP imm* инструкција није доступна. *POPF* (*pop flags*) инструкција избавља 16 битни број из магацина и смешта га у маркер регистар, а *POPFD* избавља 32 битни број из магацина и смешта га у *EFLAGS* регистар. Инструкција *POPA* (*pop all*) избавља 16 бајтова и смешта их у регистре. Код *80386 Pentium*-а инструкција *POPAD* пуни 32 битне регистре из магацина.

### 5.2.4. Сегментација - подела програма

Ту су и четири 16 битна сегментна регистара који омогућавају *8086* процесору за приступ једног *megabyte*[104] - мегабајта меморије на неуобичајен начин. Уместо груписања сегментних регистара са адресним регистром, као и у већини процесора чији је адресни простор премашио своју регистарски величину, *8086* помера 16 битни сегмент само четири бита пре него што дода у 16 битни померај (16 × сегмента + померај ), дакле производњу за 20 битну екстерну адресу из 32 битног сегмента: *offset* - померање пара. Као резултат тога, свака спољна адреса може бити прослеђен до 212=4096 другачијих сегмената: померај парова. 16 бајтно раздвајање између сегмента база (због 4 битне смене) се зове пасус-*paragraph*. Иако се сматра компликован и гломазан од стране многих програмера, ова шема такође има своје предности; мали програм (мање од 64 KB) може да се учита са почетком у фиксни померај (као што је 0) у свом сегменту, избегавајући потребу за пресељење - *relocation*[105], са по највише 15 bytes усклађености отпада.

Слика 33. 1.44 MB дискете могу да ускладишти 1,474,560 bytes података. MB у овом контексту значи 1.000 × 1.024 bytes

Компајлери за *8086* породице често подржавају два типа показивача, близу и далеко. *Near* - близу показивачи су 16 битна компензација имплицитно повезане са кôдом програма и/или података сегмента и тако се може користити само у оквиру појединих делова програма довољно малих да стане у једном сегменту. *Far* - далеки показивачи су 32 битни сегмент: *offset* - померање пара решавање на 20 битним спољним адресама. Неки компајлери подржавају велике показиваче, који су као далеки показивачи осим аритметичког показивача на *huge* - огромном показивачу, третира га као линеарни 20 битни показивач, док се аритметички показивач на *far* - далеке показиваче обавија око из свог 16 битног помераја без додиривања сегментног дела адресе.

Да бисте избегли потребу да се опише *near* и *far* на бројним показивачима, структуре података и функција, компајлери подржавају меморијске моделе подразумеване величине којима се прецизира показивач. *Tiny* - сићушан (max 64K), *small* - мали (max 128K), *compact* - компактни (data > 64K), *medium* - средње (code > 64K), *large* - велики (code,data > 64K), и *huge* - огроман (individual arrays > 64K модели покривају практичне комбинације *near, far,* и *huge* показивача за кôд и податаке. *Tiny* модел подразумева да кôд и подаци се деле у једном сегменту, као и у већини 8 битних процесора заснованих, и може се користити за изградњу *.com-files*[106] за пример.

[104] Је више од јединица byte за складиштење дигиталних информација
[105] Је процес додељивања адреса оптерећења различитим деловима програма и подешавање кода и података у програму да одражавају додељене адресе
[106] Је врста извршне датотеке, име потиче од типа датотеке са наставком .COM

### 5.2.5. Преформансе

Иако делимично у сенци других дизајнерских избора у овом конкретном чипу, мултиплексирани *bus* је ограничен перформансама незнатно; трансфери 16 битне или 8 битне количине су урадили у четири такта приступа меморије (што је брже на 16 бита, мада спорије на 8 битним количинама, у односу на типичне савремене "8 битне процесоре"). Као инструкције, варира од једног до шест бајтова, проналажење и извођење је истовремено (као што остаје у *x86* данашњим процесорима): *bus* интерфејс јединица храни инструкцијску струју до извршења јединице преко 6 битних припрема реда (облик лабаво повезано снабдевање), убрзава рад на регистрима и константама - *immediates*[107], док меморија операције нажалост постаје спорија; четири године касније, овај преформанс проблем је фиксиран са *80186* и *80286*. Међутим, пуна (уместо парцијална) 16 битна архитектура са пуном ширином *ALU* значи да 16 битне аритметичке инструкције сада могу бити изведене са једним *ALU* циклусом (уместо два, преко преноса), знатно убрза такве инструкције. У комбинацији са *orthogonalizations*[108] пословања насупрот операнд - типова и модова који се баве, као и друга побољшања, те перформансе добите преко *8080* или *8085* су прилично значајне, без обзира на случајеве у којима старији чипови могу бити бржи.

***Време извршења за типичне инструкције (у циклусима)***

| **instruction** | **register-register** | **register immediate** | **register-memory** | **memory-register** | **memory-immediate** |
|---|---|---|---|---|---|
| mov | 2 | 4 | 8+EA | 9+EA | 10+EA |
| ALU | 3 | 4 | 9+EA, | 16+EA, | 17+EA |
| jump | *register* => 11 ; *label* => 15 ; *condition,label* => 16 | | | | |
| integer multiply | 70~160 (depending on operand *data* as well as size) plus EA | | | | |
| signed integer divide | 80~190 (depending on operand *data* as well as size) plus EA | | | | |

- EA = време за рачунање ефективне адресе, у распону од 5 до 12 циклуса,
- Временски период у најбољем случају зависи од статуса, припрема инструкција и других фактора.

Као што се може видети из ове табеле, операције на регистарима и константама су били брзи (између 2 и 4 циклуса), док меморија-операнд инструкције и скокови су били прилично спори; скоковима су требали више циклуса него на простом *8080* и

[107] Идентификатор чија је придружена вредност не може обично бити мењана од стране програма током његовог спровођења
[108] Је процес проналажења скупа ортогоналних вектора који обухватају одређени потпростор

*8085*, и *8088* (коришћени у *IBM PC*-у) који је додатно отежан ограниченим *bus*-ом. Разлози зашто је већина меморијских повезаних инструкција била спорија троструко:

- Слабо спрегнути курс и извршне јединице су ефикасне за добављање инструкција, али не и за скокове и податаке случајног приступа (без посебних мера),
- Не наменски обрачун адреса потпуни сабирач је пружен, а *microcode* рутине су морале да користе главну *ALU* за ово,
- Адреса и *buses* подаци су били мултиплексирани, приморавајући је нешто дуже (33 ~ 50%) него у *bus* циклусу типичних савремених 8 битних процесора.

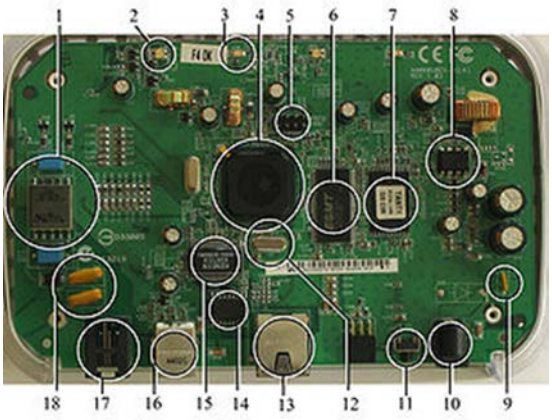
Међутим, меморијски приступ преформансама је драстично побољшан са следећим генерацијама *Intel's* чипова. *80186* и *80286*, оба су повећали адресу хардвер израчунавања, меморисање великог броја циклуса, а *80286* је имао посебану не мултиплексирану адресу и *buses* податке.


Слика 34. Intel 80186 Microprocessor


Слика 35. AMD 80286 (16 MHz version)

## 5.2.6. Верзија чипа

Фреквенције је првобитно ограничена на 5MHz (*IBM PC* је користио 4,77MHz, 4/3 стандарда *NTSC Colorburst*[109] - боја узастопне фреквенције), али су последње верзије у *HMOS*-у су одређене за 10MHz. *HMOS-III* и *CMOS* верзије су произведене за дужи временски период (бар док је у 1990) за *embedded system*[110] - уграђене системе, иако његов наследник, *80186/80188* (који укључује неке *on-chip* периферије), је више популаран за уграђену употребу .




Слика 36. Слика интерног ADSL модем / рутера. Модеран пример уграђеног система. Саставни делови су: microprocessor (4), RAM (6), и flash memory (7).

[109] Је аналогни видео, композитни видео сигнал генерисан од видео сигнала генератора коришћен да задржи синхронизовани сигнал боје у боји телевизијског сигнала

[110] Је компјутерски систем дизајниран за одређене контролне функције у оквиру већег система, често у реалном времену рачунарских ограничења

### 5.2.6.1. Деривати и клонови

Слика 37. NEC V20, 8MHz.

Компатибилани, у многим случајевима, побољшане верзије су произведене од стране *Fujitsu, Harris/Intersil, OKI, Siemens AG, Texas Instruments, NEC, Mitsubishi*, и *AMD*. На пример, *NEC V20*[111] и *NEC V30* пар били су хардвер компатибилан са *8088* и *8086*, респективно, али је уградио сет инструкција од *80186* заједно са неким од *80186* побољшања брзине, обезбеђујући *drop-in* (убацивање битова) способност да надогради сет инструкција и брзину обраде без потребе произвођача да измењују своје дизајне. Такави релативно једноставани и ниске потрошње *8086* компатибилни процесори у *CMOS* се још увек користе у уграђеним системима.


Слика 38. K1810BM86

Електронска индустрија Совјетског Савеза била је у могућности да реплицира *8086* и кроз индустријску шпијунажу и обрнутог инжењеринга. Добијени чип, *K1810BM86*[112], био је бинарни и пин компатибилан са *8086*, али није механички компатибилна јер је коришћен у метричка мерења.

*8088* и *8086* су била одговарајућа језгра совјетских производа *PC* компатибилних *ES1840* и *ES1841*[113] рачунара. Међутим, ови рачунари имају значајне хардверске разлике из својих изворних прототипова, а подаци/*bus* адреса струјно коло дизајнирано је независно од интелових производа. *ES1841* је први *PC* компатибилни рачунар са динамичким *bus* димензионисањем. Касније су неки од принципа *ES1841* усвојени у *PS/2* и неке друге машине.

[111] Процесор обрнутог инжењеринга, пински компатибилана верзија Intel-а 8088 са сетом инструкција компатибилних са Intel-ом 80186
[112] Совјетски клон Intel 8086 microprocessor-а
[113] Совјетски клонови IBM PC-а 1980.

## 5.3. Произвођачи микропроцесора

Постоји већи број произвођача микропроцесора, а међу њима се истичу:

- Intel, Integrated Electronics Corporation,


- IBM, International Business Machines Corporation,


- AMD, Advanced Micro Devices,


- Multinational telecommunications company,


- Transmeta.

# 6. Закључак

Гледано кроз историју, процесори су превазилазили многе промене и иновације у техници. Први процесори су били механички и практично нису били засебан део, затим електромеханички (релејни), па на бази електронских вакуумских цеви и били су јако велики. До значајног смањења димензија и повећања перформанси дошло је употребом транзистора (минипроцесори) и у другој половини 20. века интегралних кола (микропроцесори).

Микропроцесор је интегрисано коло великог степена интеграције компонената. Функционише као централна процесорска јединица рачунара на једном интегрисаном колу или највише неколико интергисаних кола. Микропроцесор је свестрано употребљив програмабилни уређај који прихвата дигиталне податке као улаз, обрађује их према инструкцијама смештеним у меморији дајући резултат на излазу. Обрађује бројеве и симболе представљене бинарним бројним системом.

Најзначајније разлике међу микропроцесорима су у броју регистара, њиховој намени и међусобним везама. Број регистара и њихова организација директно утичу на листу наредби. Произвођачи су настојали да нађу добар компромис између броја регистара и листе наредби како би програми микропороцесора били максимално ефикасни. Велики број микропроцесора показује да у томе нису били увек успешни јер је развој микропроцесора увек био испред производње терајући произвођаче да стално избацују нове, напредније микропроцесоре.

Три основне карактеристике микропроцесора су:

- Сет инструкција: листа наредби које микропроцесор може да извршава.
- Ширина магистрале за податке: Број битова који се обрађује једном наредбом.
- Брзина рада: одређује колико наредби у секунди процесор може да изврши. Даје се у мегахерцима (MHz).

У сваком случају, основни задатак процесора је у извршавању сетова инструкција рачунара. Ограниченост њихових функција раније је представљала уједно и тежњу за њиховим константним унапређивањем. Та тежња је, пре свега, базирана на томе да процесори, уопштено говорећи, брже извршавају тражене наредбе, а да у исто време, могу обављати што више функција истовремено.

Микрорачунари (базирани на микропроцесорима) се праве и користе не само као радне станице и сервери, већ и у роботима, телекомуникационим уређајима, сателитима, аутомобилима, инструментима, мобилним телефонима, камерама, кућним уређајима као што су машине за прање рубља, микроталасне рерне и кућним пекарама хлеба, као микроконтролери.

Дуго су рачунари били у употреби само на универзитетима, већим предузећима и државним установама, пре свега јер су били јако гломазни (биле су потребне читаве просторије и спратови) и скупи. Али открићем микропроцесора од компаније Интел (*Intel Corporation*) све се то променило.

Појава јефтиних рачунара са интергисаним колима трансформисала је модерно друштво. Микропроцесори опште намене у персоналним рачунарима користе се за израчунавања, обраду текста, приказ мултимедијалних садржаја и комуникацију преко интернета. Много више микропроцесора је у саставу уграђених система (embedded system) обезбеђујући дигиталну контролу на милионе објеката, од кућних уређаја до аутомобила, мобилних телефона и контролера индустријских процеса.

Микропроцесори данас спадају у најкомпликованија интегрална кола. Мале димензије им омогућавају смањену потрошњу у односу на претходнике, много веће брзине рада и, можда најважније, могућност употребе практично у било ком уређају. Све ово је учинило микропроцесор једним од највећих достигнућа 20. века.

# 7. Литература


- Стојчев. М. (1999), „Архитектура и програмирање микрорачунарских система заснованих на фамилији процесора 80x86”, Универзитет у Нишу, Електронски факултет, Ниш.
- Хајдуковић. М. (2004), „Архитектура рачунара“, Факултет техничких наука, Нови Сад.
- http://sr.wikipedia.org/ јун 2012. год.
- http://www.laynetworks.com/ јун 2012. год.
- http://whatis.techtarget.com/definition/ јул 2012.год.
- http://segaretro.org/ јул 2012.год.
- http://www.itcitaliste.rs/ јул 2012.год.
- http://www.answers.com/topic/ август 2012.год.
- http://commons.wikimedia.org/ август 2012.год.