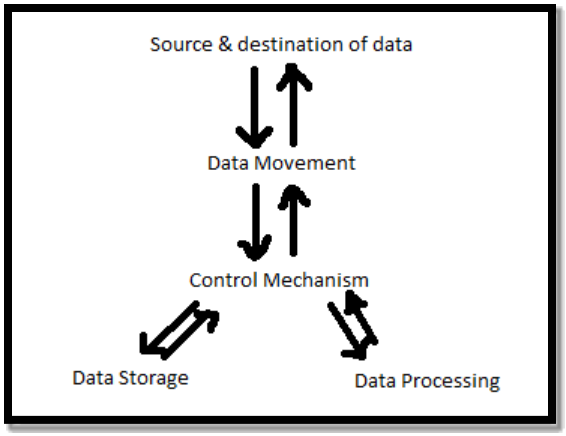
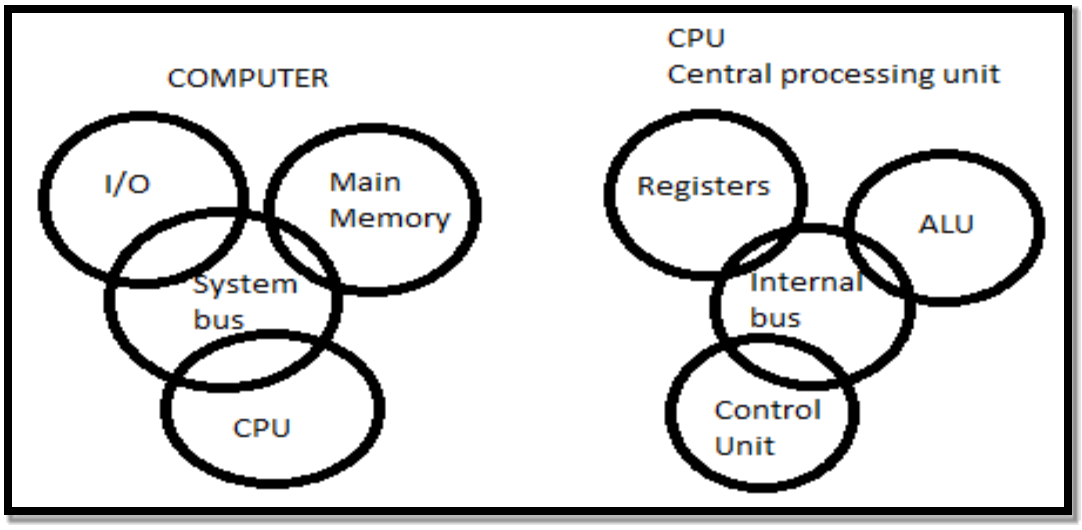
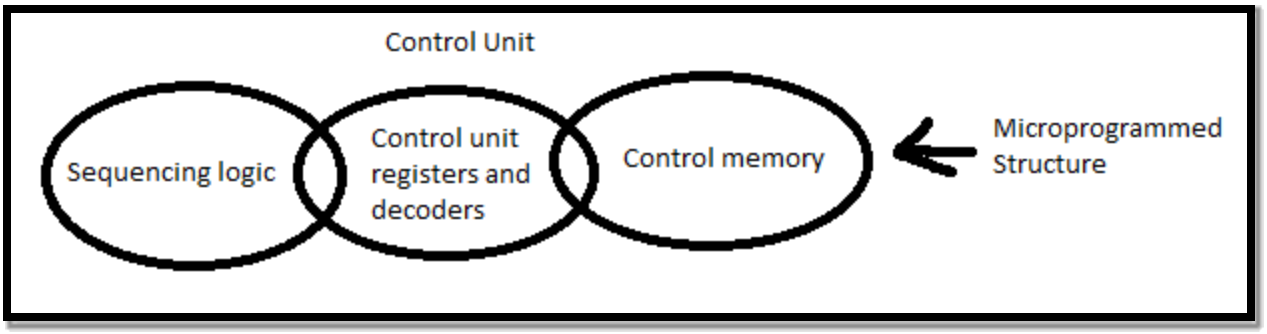
Компјутерски Архитектури – Теорија

1. Вовед

- Компјутерска архитектура – Атрибутите на системот кои се видливи за програмерот односно оние атрибути кои директно влијаат на логичката ексекуција на програмата
  - Пр. Инструкциско множество, број на битови за променливи, 1/0 механизми, техники за адресирање на меморија.
- Компјутерска организација - како се имплементирани архитектурните карактеристики.
  - Пр. Контролни сигнали, интерфејси, технологија на меморија (хардверски детали невидливи за програмерот)
  - Пр. Intel X86 и IBM System/ 370 фамилиите имаат иста основна архитектура, а организацијата се разликува помеѓу различните верзии. Ова овозможува компактибилност на кодаб (софтверот)
- Компјутерски систем – комплексен систем од хиерархиски поврзани компоненти
- Структура – начинот на кој компонентите се поврзани
- Функција – Работата на секоја компонента како дел од целокупната структура
- Функции на компјутерот :
  - Обработка на податоци
  - Складирање на податоци
  - Преместување на податоци
  - Контрола



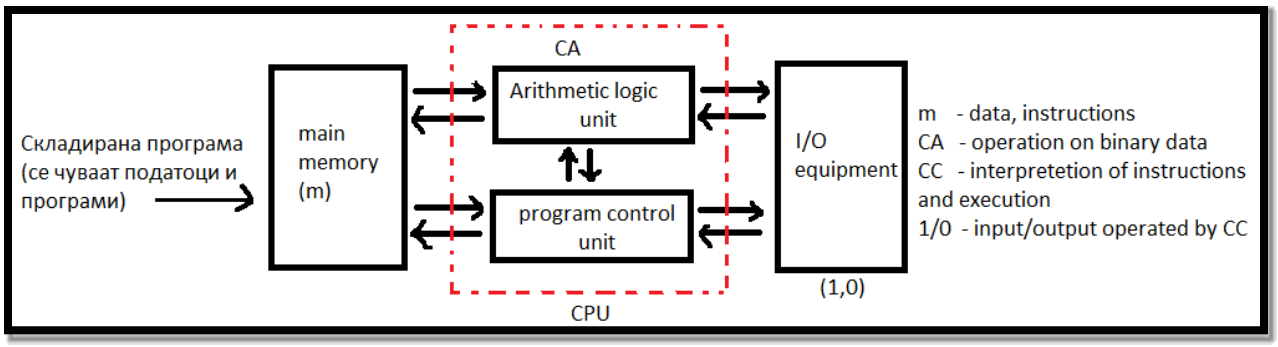




- Компјутер
  - Central processing unit (CPU) – процесор – ги контролира операциите на компјутерот и ги обработува податоците
  - Меморија – чува подаотци
  - I/O – преместува податоци мегу компјутерот и надворешната околина
  - System Interconnection (BUS) – Механизми со кои комуницираат процесорот, меморијата и I/O(жици)
- CPU
  - Control unit – ги контролира операциите на процесорот, а со тоа и на компјутерот
  - Arithmetic & logic unit (ALU) – ги обработува податоците
  - Регистри – создава интерна меморија на процесорот
  - CPU Interconnection – механизам за комуникација мегу CU, ALU и регистрите

2. Компјутерска Еволуција и перформанси
   - 1 ГЕНЕРАЦИЈА – ВАКУУМ ЦЕВКИ
- ENIAC (Electronic Numerical Integrator and Computer)
  - Направен заради потребите на BRL (Balistics Research Laboratory) на армијата на САД за WW2
  - Направен со вакуум цевки (18.000)
  - Тежина – 30 тони , 120м^2, 140 kW потрошувачка
  - 5000 Собирања во секунда
  - Декадна машина (не бинарно)
  - Меморија, 20 акумулатори кои чуваат 10 цифрен број
  - Се програмира рачно со прекинувачи
  - Направен од Mauchly и Eckert

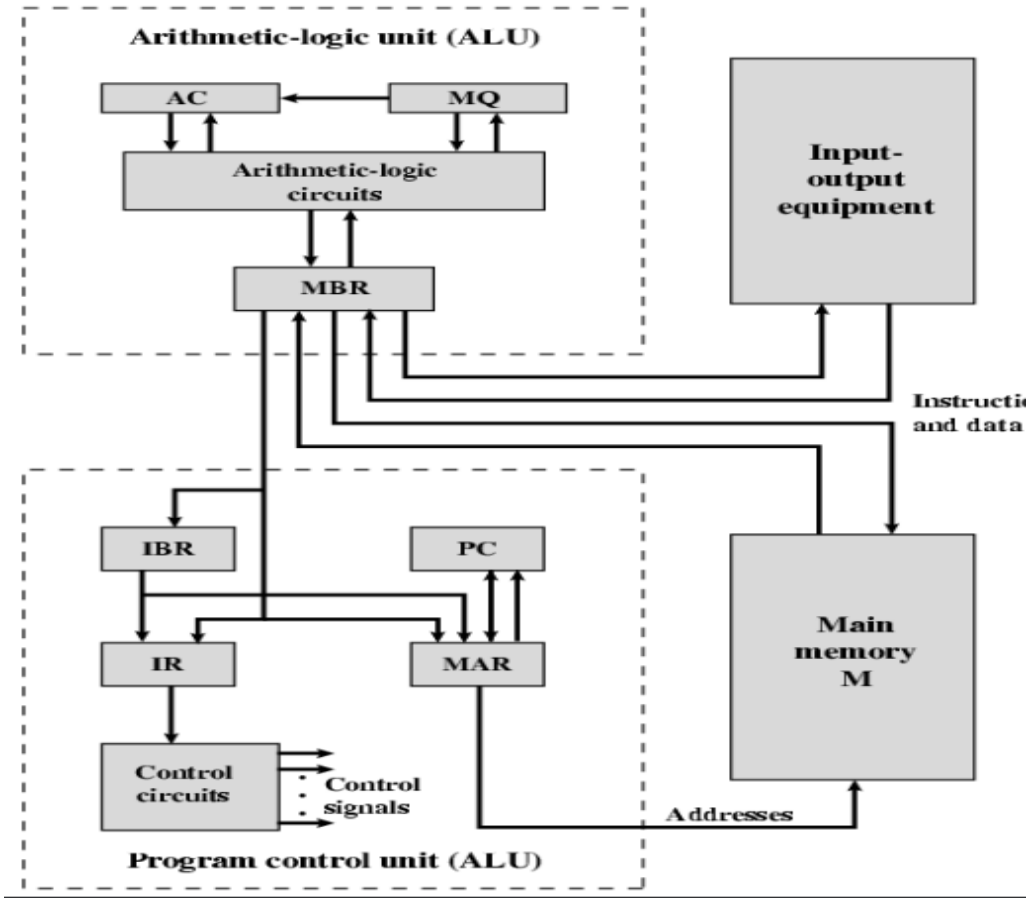
- Von Neumann / Turing Machine
  - Aka IAS Computer (Princeton Institute for Advanced Studies)
  - Структура :



  - Меморија – 1000 линии (зборови) составени од 40bits
  - Броевите и инструкциите се претставени во бинарна форма

Број – sign bit (0)| value (1 – 39)

Instruction Word	0 -7	8 – 10	20 – 27	28 - 39
	Op. code	Address	Op. Code	Address
	Left Instruction		Right Instruction	



- Memory buffer register (MBR) - до него доаѓаат зборови од I/O и меморијата
- Memory address register (MAR) – ја специфицира адресата на зборт кој што ке се впише во MBR
- Instruction register (IR) – содржи 8-bit opcode инструкциите
- Instruction buffer register (IBR)- привремено ја чува десната интрукција од зборот.
- Program counter (PC) –ја содржи адресата на следната инструкција

- Accumulator (AC) * multiplier quotient (MQ) –привремено чуваат операнди и резултати
  - IAS има 21 инструкција
    - Пренос на податоци : меморија – ALU или >ALU
    - Unconditional branch : за повторувачки операции
    - Conditional branch : како if структура
    - Arithmetic : ги врши ALU
    - Address modify :

- Комерцијални компјутери
  - Eckert – Mauchly Computer Corporation
    - UNIVAC – Universal Automatic Computer
      1. Се користи за пресметки од US Bureau of the Census
      2. Станува дел од Sperry – Rand Corporation
      3. Опсег на задачи : Алгебарски пресметки ,статистички проблеми, логистички проблеми, billings за компанија за осигурување.
    - UNIVAC II (Следна 1100 серија)
      1. Поголема меморија, побрз, backward compatible
- IBM – произведување опрема за обработка на дупчени картички
    - 701
      1. Прв компјутер со складирано програма на IMB
      2. ЗА научни пресметки
    - 702
      1. Погоден за бизнис апликации
      2. 700/7000 серија на компјутери

- 2 Генерација – Транзистори
  - Помали, поефтини , помало диспација на топлина
  - Solid – state device , направени од силикон
  - Пронајден од Bell Labs (1947)
  - NCR, RCA – први почнале да проиузведуваат мали транзисторски машини ; IMB – 7000 серијата
  - Карактеристични се програмските јазици и појавата на системски софтвер
  - DEC – Digital Equipment Corporation – PDP – 1 го започнале трендот на нижи компјутери

- IBM 7094
  - Поголема меморија 2k – 32k 2^10 36- bit words
  - Memory cycle time 3ms -1,4ms
  - Број на операции 2m – 185
  - Додадени се data канали – посебни I/O модули со сопствен процесор и инструкциски сет
  - Multiplexor – централа терминална точка мегу data каналите, CPU и меморијата (schedules access)
- 3та генерација – интегрирани кола
  - Самоодржувачки транзистор – дискретна компонента
- Микроелектроника
  - Компјутерот е составен од 2 фундаментални компоненти :
    - Порти – уреди кои имплементираат едноставна булова или логичка функција
    - Мемориски ќелии – флип флопови
    - Меѓуповрзувања
  - Може да се произведуваат на полупроводник (сицилиумска плочка)
  - На почетокот само мал број на проти и флип флопоови можеле да се спакуваат заедно – small – scale - interogation

- МУРОВ ЗАКОН – Гордон Мур = ко-основите на Intel

- бројот на транзисторите на чип ќе се дуплира секоја година (секои 18 месеци во 1970тите)
- последици од Муров закон
- цената на чипот останува скоро непроменета
  -пократки електични патеки ---> поголема брзина (подобри перформанси)
  -помала големина на компјутерот ---> поголема флексибилност
  -намалена моќ и потреба за ладење
  -Меѓуповрзувањата на чиповите се посигурни

- IMB system /360 $100k

- некомпатибилни со 7000 серијата
- прва планирана „фамилија“ компјутери
  -слични или идентични инструкциски множества
  -слични или идентични оперативни системи
  -зголемена брзина
  -зголемен број на терминали (I/0 порти)
  -зголемена меморија
  -зголемена цена

- DEC DDP – 8 $16k

- првиот мини комјутер
- доволно мал да се стави на лабораториска лупа
- не барал климатизирана соба
- вградливи апликации
- структура на магистрала

omnibus – 96 сигнални патеки контролирани од CPU, флексибилна архитектура

Понови генерации:

- LSI – large scale integration (1000+ компоненти/чип)

- VLSI – very large scale integration (10,000+ компоненти/чип)
- ULSI – ultra large scale integration (1,000,000+ компоненти/чип)

- Полупроводничка меморија

- Fairchild -1970
- Големина на 1 јадро = 1 bit од магнетното јадро
- може да содржи 256 бита меморија
- побрза меморија
- Не-деструктивно читање
- 4 пати поголем капацитет за секоја нова генерација

- Микропроцесори

* Intel 4004
- прв микропроцесор
- ги содржел сите компоненти на CPU на еден чип
- можел да собира 2 4-bit броеви и множел со повторувачко додавање

* Intel 8008
- првиот 8-bit микропроцесор
- 4004 и 8008 биле дизајнирани за специфични апликации

* Intel 8080
- првиот микропроцесор за општа намена
- 8-bit микропроцесор, побрз, поголем инструкциски сет
- дизајниран да биде CPU за микрокомпјутер за општа намена

16-bit микропроцесори- Intel 8086

32-bit микропроцесори – Bell Labs & Haolett Packard – Intel 80386

- Брзина на микропроцесори (техника на забрзување)

- branch prediction = процесот предвидува кои групи на инструкции би можеле да бидат следни

- анализа на текот на податоците = кои инструкции зависат од кои резултати, закажува извршување на инструкциите

- Шпекулативно извршување = предвремено извршување на инструкции

- pipelining = во текот на извршувањето се преклопуваат повеќе инструкции

- ◆ Балансирање на перформанси

- зголемена брзина на процесот и капацитетот на меморијата

- брзината на меморијата заостанува зад процесорот

- решенија:

  - зголемен број на битови кои се превземаат во единица време (DRAM да е поширок, а не подлабок)

  -промена на DRAM интерфејсот со помош на кеш

  -намалување на фрекфенцијата на пристап до меморијата (покомплексен и чип кеш)

  -зголемен пропустен опсег на меѓуповрзувањата (магистрали)

- ◆ I / 0 уреди

- периферии со интезивни I / 0 барања

- барања за голема пропусност на податоци

- проблем со преместувањето на податоци

- решенија:

  -кеширање

  -баферирање

  -магистрали со големи брзини

  -посложена структура на магистралите

  -конфигурации со повеќе процесори

- ◆ Балансирање на провесорски компоненти, главна меморија, I / 0 уреди и меѓуповрзувачки структури

- ◆ Подобрувања во организацијата и архитектурата на чипот

- Зголемување на хардверската брзина на процесорот

- намалување на големината на логичките порти

- зголемена фрекфенција на такт сигналот

- Зголемена големина и брзина на кешот

- кеш во чиповите, времето за пристап до кешот значајно се намалува

- Промена на организацијата и архитектурата на процесорот

- зголемување на брзината на извршување

- паралелизам

- Проблеми со брзината на тактот и густината на портите

-моќност

-зголемување на густината на логиката и брзината на тактот = зголемување на густината на моќноста

-проблем со топлина

- RC доцнење

-брзината на движење на електроните во транзисторите е ограничена со отпорноста и капацитативноста на металните жици

-пораст на RC производот = поголемо доцнење

-потенки жици = поголема отпорност

-поблиски жици = поголема капацитативност

- Латентност на меморијата

-меморијата заостанува зад процесорот во брзина

-решение: поголемо внимание на организација и архитектура

- Зголемување на капацитетот на кешот

- 2/3 нивоа кеш меѓу процесорите и главната меморија

- повеќе кеш меморија на чипот

пр. Pentium – 10% од чипот е кеш

Pentium 4 – 50% од чипот е кеш

- Покомплексна логика на извршување

- pipelining = овозможува паралелно извршување на инструкциите (проточна цевка)

- superscalar = повеќе проточни цевки на 1 процесор (инструкциите се независни, паралелно извршување)

- DIMINISHING RETURNS

- комплексна организација на процесорот

  -голем паралелизам

  -идните зголемувања ќе бидат скромни

- придобивките од кешот доаѓаат до крајна граница

- зголемената брзина на тактот влегува во проблем со дисипација на моќноста (од физички аспект)

- MULTUCORE / MULTICORE – повеќе јадра

- повеќе процесори на еден чип (голем споделен кеш)

- во процесорот, зголемување на перформансите е пропорционално на квадратен корен од зголемување на комплексноста

Ако софтверот користи повеќе процесори, 2x процесори скоро ги дуплира перформансите

2 поедноставни процесори на чипот наместо еден многу комплексен (поголем кеш, помала моќност)

- Intel x86 еволуција

- 8080

  -first general purpose microprocessor

  -8bit data path

  -used in the first personal computer – Altair

-8086

  -16bit machine

  -instruction cache prefetch instructions ← 1MB

  -8088: in the first IBM PC

-80286

  -16MB adressable memory

-80386

  -32bit machine

-first to support multitasking

-80486

-sophisticated powerful cache & instruction pipelining

-build in math coprocessor

-Pentium

-superscalar techinique

-multiple instructions executed in parallel

- Pentium Pro

-increased supercalar organization

-agressive register renaming

-tranch prediction

-data flow analysis

-speculative execution

- Pentium II

-MMX technology

-processing video, audio & graphics data

- Pentium III

-additional floating point instructions for 3D graphics

- Pentium 4

-additional floating point, multimedia enhancements

- Core

-first microprocessor with dual core

- Core 2

-64bit architecture

- Core 2 Quad – 3GHz

-4 processors

-820 milion transistors

->embedded system architecture

->instruction set evolved with backwards compatibility

->500+ instructions in the instruction set

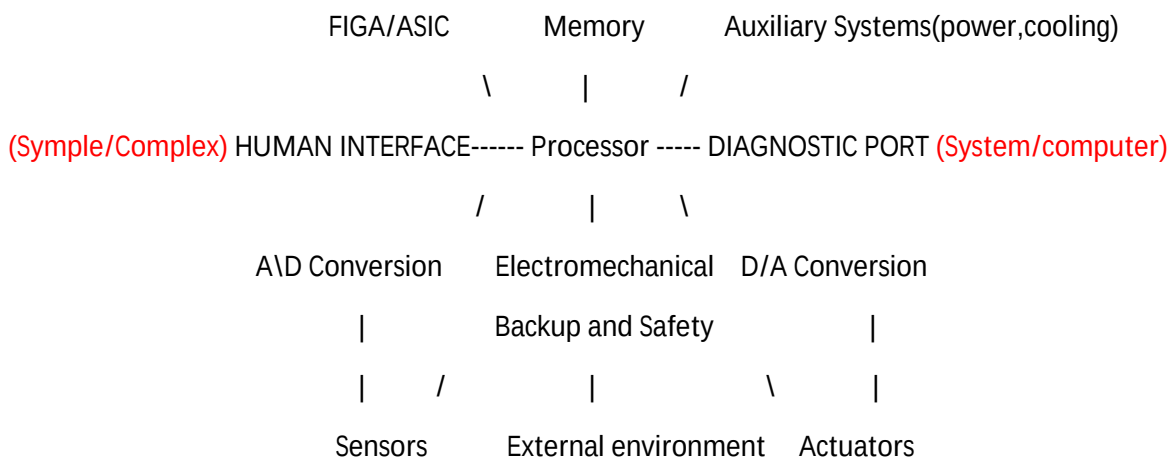
*ARM

Дефиниција: Embedded system:Комбинација на хардвер и софтвер,дизајниран да извршува одредена функција.Најчесто се делови од поголеми системи или производи.(не е компјутер за општа намена)

->различни големини(различни ограничувања,оптимизација,повторна употреба)

->различни барања:безбедност,надежност,real time,флексибилност,животен век,услови за средината,статичен<->динамичен товар,бавни<->брзи пресметки<->В/И барања,дискретни<->континуални.

***Организација на** embedded system*



*FPGA - Special purpose field programmable

*ASIC – application specific

RISC = reduced instruction set computer

*ARM **еволуција**

-RISK:based microprocessors and microcontrollers(ZNG);

-мал ,голема брзина ,мала потрошувачка;

-се користат во PDAs,рачни играчки конзоли,телефони(iPad,iPhone);

-origin:Acorn Computers company-GB;

-ARM1 and ARM2:истражување и напредок и копроцесор во BBC машината;

-ARM3:уште подобар;

-ARM ltd. – Acorn,VLSI,Apple Computer;

-Категории на ARM системи (ги задоволуваат овие системи)

->вградливи системи во реално време(системи за меморија,апликации);

->апликациски платформи (уреди на отворени платформи Unix,Palm OS,Symbian OS,Windows CE)

->безбедносни апликации.

***Одредување на перформанси**

-клучни параметри:перформанси,цена,големина,безбедност,надежност,потрошувачка на моќ.

-Брзина на такт сигнал(clock speed)-Hz

->на сигналите им треба време да се стабилизираат на 1 или 0 и тие можат да се менуваат со различни брзини;

->операциите мора да бидат синхронизирани

->инструкцијата се извршува во дисркетни чекори (fetch,decode,load,store,arithmetic and logical operations)потребни се повеќе такт циклуси;

->проточност = повеќе инструкции едновремено.

перформанси ≠ перформанси на такт сигнал

*Брзина на извршување инструкции

- CPU – cycles per instruction

$$CPU = \frac{\sum_{i=1}^{n}(CPIi * Ii)}{Ic}$$

I_c=Број на извршувања

- Processor time

$T = I_c * CPI * T$

$T = Ic*(p+mk)*\tau$

p=**број на тактови за декодирање и извршување**

m=broj na memoriski referenci

$$K=\frac{memory\ cucletime}{processor\ cycletime}$$

- MIPS-million instructions/second

$$MIPS = \frac{Ic}{T * 10^6} = \frac{f}{CPI * 10^6}$$

- MFLOPS = Million floating-point operations/second

$$MFLOPS = \frac{N\ of\ executed\ floating\ -\ point\ operations}{Execution\ time * 10^6}$$

- Зависни од инструкциско множество,дизајн на компајлер,имплементација на процесор,кеш и мемориска хиерархија.

*BENCHMARK

- Карактеристики
  ->програми дизајнирани за тестирање перформанси;
  ->напишани во јазик на високо ниво(портабилни);

  ->стилови на работење (системски,нумерички,комерцијални);

  ->лесно мерливи;

  ->широко дистрибуирани.

- SPEC BENCHMARKS = SYSTEM PERFORMACE EVALUATION CORPORATION
  ->SPEC CPU 2006-за пресметковни можности
  -17 floating point програми(C, C++, Fortran);
  -12 integer програми(C, C++);
  -3000000 линии код.
  ->**Други SPEC пакети**
  -SPECJUM 98 = Хардвер и софтвер;
  -SPECJbb2000 = Java Business Benchwark;
  -SPECweb99 = www servers;
  -SPECmail2001 = работи како mail server.
- SPEC метрика за брзина = извршување на 1 задача
  ->основно работење на секој benchwark со користење на референтна машина.
  ->однос од референтното време и системското време.

$$r_i = \frac{Trefi}{Tsuti}$$

Trefi = **време за извршување на** benchmark I **на р.м.**

Tsuti = -----| |------ **на тестираниот систем**

  ->целокупните перформанси се просек од едно сите на сите benchmarks(геометриска средина)

$$r_G = \left(\prod_{i=1}^{n} r_i\right)^{\frac{1}{n}}$$

- SPEC метрика за пропусност-брзина;
  ->мери пропусност/брзина на машината за изведување одреден број задачи;
  ->Повеќе копии на benchwarks работат истовремено(колку што има процесори)

$$r_i = \frac{N * T_{refi}}{T_{suti}}$$

N=**број на копии кои работат**;

Tsuti = **поминато време од почетокот на извршување до крајот на сите копии.**

\+ **Геометриска средина**

***Амдалов закон**-Grene Amdahl

- Потенцијално забрзување со користење повеќе процесори;

->кодот треба да може да се парализира;
->забрзување е ограничено поради diminishin returns на повеќе процесори.

- Зависат од задачата
  ->серверите добиваат со одржување врски на процесорите;
  ->базите се делат во паралелни задачи.
- Програма на 1 процесор
  ->f е бесконечно паралелизабилен без товар за распоредување
  -> 1-f е сериски;
  -> T е време на извршување на програма на 1 процесор;
  ->N е број на процесори кои ги користат паралените делови од кодот.

$$\text{Speed up} = \frac{T * (1 - f) + Tf}{T * (1 - f) + \frac{Tf}{N}} - \frac{1}{(1 - f) + \frac{f}{N}}$$

$$\text{Speed up} = \frac{\text{Time to execute program on 1 processor}}{\text{Time to execute program on N parallel processors}}$$

- f мало,паралелните процеси имаат мал ефект N->∞ ограничено забрзување $\frac{1}{1 - f}$ (Diminishing returns за повеќе процесори).

3.)Глобален преглед на функцијата и поврзаноста на компајлерот

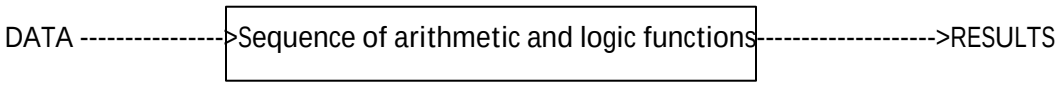
***Главни концепти на** Von Neumann architecture

- податоците и инструкциите се чуваат во една меморија (read-write);
- кон податоците од меморијата се пристапува преку мемориски адреси;
- егзекуцијата има секвенцијална природа

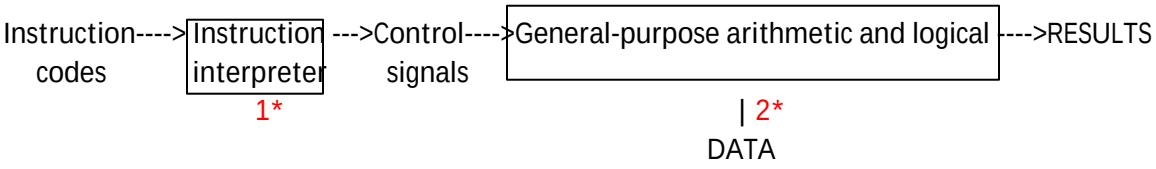
*Hardwired program = ''програма'' која настанува како резултат на поврзување на различни компоненти за да се изврши одредена операција.

- Ваквите системи не се флексибилни;
- Концепт за хардвер со општа намена која прави различни работи во зависност од контролните сигнали(софтверско програмирање)

Хардверско програмирање



Софтверско програмирање



***Програма**

- Низа пд чекори,во секој чекор се изведува аритметичка или логичка операција(со различно множество контролни сигнали)

1* + 2* = CPU \

= компоненти на компајлер

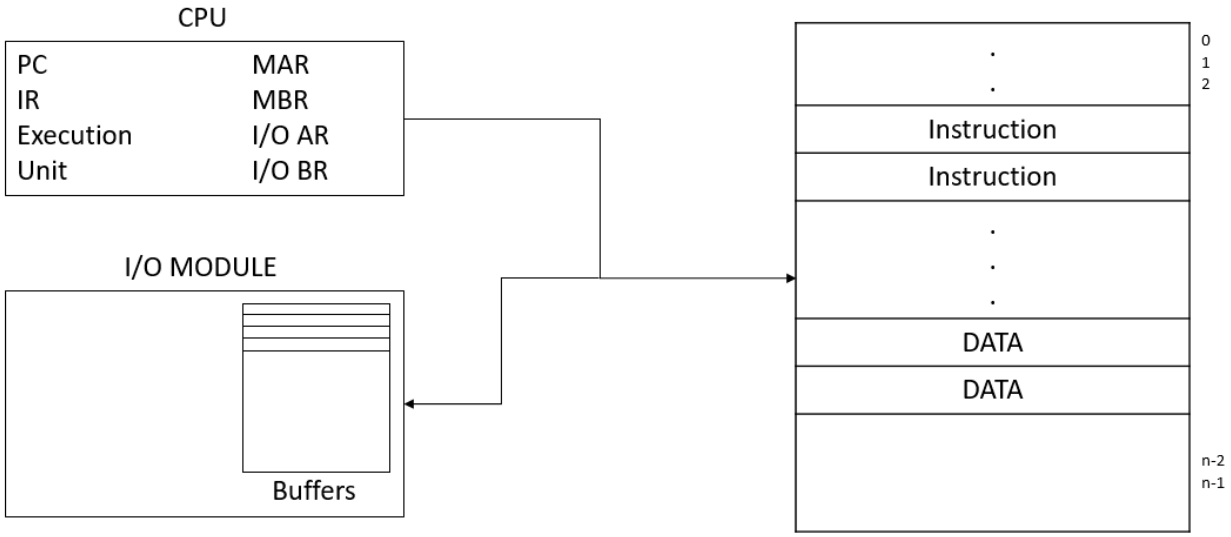
1/0 components/

(main)memory

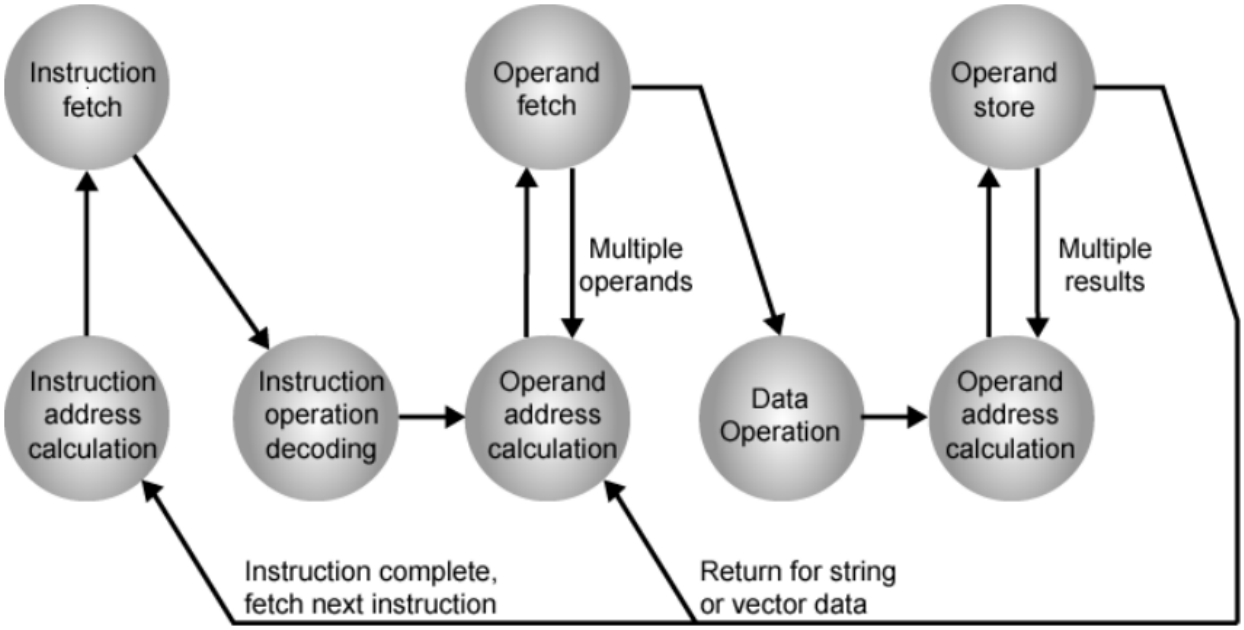
- ❖ Секоја операција има уникатен код (пр. ADD, MOVE)
- ❖ Прифаќање на кодот –> контролни сигнали

- Функција на компјутерот

Глобален преглед на компјутерските компоненти



- o Инструкциски циклус (2 чекори)
  1. fetch (земи) + decode (декодирај)
  2. execute (изврши)
- o Fetch циклус
  - ➢ Програмскиот бројач (PC) ја чува адресата на следната инструкција која треба да се земе
  - ➢ Инструкцијата се зема од мемориската локација во PC, а PC се инкрементира за (освен ако не е кажано поинаку)
  - ➢ Инструкцијата се запишува во инструкциски регистар (IR), се интерпретира и зе изведуваат бараните акции
- o Execute циклус
  - ➢ процесор – меморија = трансфер на податоци меѓу CPU и главната меморија
  - ➢ процесор - I/O = трансфер на податоците меѓу CPU и I/O
  - ➢ обработка на податоци = аритметичко/логичка операција врз податок
  - ➢ контрола = промена на текот на низата операција (eg. jump)



- Прекини (Interrupts)
  - механизам со кој други модули (пр. В/И) може да ја нарушат нормалната низа на обработка
  - за подобрување на ефикаснота на процесорот
  - видови на прекините:
    - програмски (overflow, делење со 0)
    - тајмер – генериран од внатрешен процесорски тајмер, се користи во pre-emptive multitasking
    - В/И – од В/И контролер
    - хардверски пад (грешка за парност на меморијата)
- Циклус на прекини – додаден е на инструкцискиот циклус
  - процесорот проверува дали има прекин(преку сигнал за прекин
  - ако нема прекин, ја зема следната функција
  - ако има најавен прекин:
    - се прекинува и запишува програмата
    - PC се поставува на почетната адреса на процедурата за обработка на прекинот
    - прекинот се обработува
    - се враќа содржината и се продолжува со прекинатата програма
- Справување со повеќе прекини (2 начина)
  1. Оневозможување на прекини
     - процесорот игнорира други орекини додека обработува прекин
     - прекините остануваат во чекање додека не се заврши со обработка на првиот прекин
     - прекините се обработуваат во низа како што се јавуваат
  2. Дефинирање на приоритети

        - прекини со понизо приоритет може да се прекинат од некој со повисок приоритет
        - по обработување на прекинот со повисок приоритет, процесот се враќа на претходниот прекин
- Поврзување
    - Сите единици мора да се поврзани
    - постојат различни типови на врски за различни единици (меморија, I/O, CPU)
    - поврзување со меморија = добива и се испраќа податоци, добива адреси (локации) и контролни сигнали READ, WRITE, TIMING
    - I/O поврзување:
        - излез: прими податок од компјутер, испрати на периферија
        - влез: прими податок од периферија, испрати на компјутер
        - прими контролни сигнали од компјутер
        - испрати контролни сигнали кон периферија
        - прими адреса од компјутер (пр. бр. на порта)
        - испрати игнал за прекин (контрола)
    - CPU поврзување = прима инструкции и податоци, запишува податоци по обработка, испраќа контролни сигнали кон други единици, прима и реагира на прекини
    - побрзувања:
        - memory <—> processor
        - I/O <—> processor
        - I/O <—> memory
- Магистрали = структура која поврзува повеќе уреди одеднаш
    - само 1 уред од магистралата може да пренесува информации во дадено време
    - единечна или повеќекратна магистрала (според бр. на битови)
    - системска магистрала = магистрала која ги поврзува главните компјутерски компоненти (CPU, memory, I/O)
        - 50 – 100 линии, секоја со свое значење и улога
            - податочни, адресни, контролни линии (магистрали)
- Податочна магистрала
    - пренесува податоци (не прави разлика меѓу податок и инструкција)
    - ширината е клучна за одредување на перформансите пр. 8, 16, 32, 64 bit (линии со вредност0 или 1)
- Адресна магистрала
    - го одредува изворотили одредиштето на податокот
      пр. CPU треба да прочита инструкција/податок од некоја локација во меморијата
    - ширината на магистралата о одредува меморискиот капацитет на системот
    - се користи за адресирање на В/И порти
- Контролна магистрала
    - го контролира пристапот и употребата на податочните и адресните магистрали (timing)
        - меморија читај/запиши сигнал
        - В/И читај/запиши

        - барање за пристап до магистрала
        - барање за прекин (interrupt)
        - clock – такт сигнал (за синхронизирање операции)
        - Reset
- Работа на магистрала
    - модул праќа податоци до друг модул
        1. право на користење на магистралата
        2. пренесување податоци
    - модул добива податоци од друг модел
        1. право на користење на магистралата
        2. пренесување на барање на податоци
        3. вториот модул да ги прати податоците

  Физички изглед на магистралите:

  паралелни линии на електронските плочки, плоснати лентести кабли (Ribbon cables), strip конектори на матичната плоча (пр. PCI), множество од жици

- Проблем со единечна магистрала (со многу уреди)
    - пропагациско доцнење
        - долги податочни патеки водат кон координација на магистралата => лоши перформанси
        - насобраниот податочен трансфер се приближува до капацитетот на магистралата
    - повеќето системи користат повеќе магистрали (решение)
    - мезанин архитектура = брза магистрала којаштое интегрирана со останатиот дел од системот (за појаки В/И уреди)
- типови магистрали
    - Dedicated (специјално наменета) – одвоени податочни и аредсни линии
    - мултиплексирана – споделени линии+контролна линија која кажува дали е податок или адреса
    - предност = помалку линии
    - недостаток = покомплексна контрола, перформанси
- Арбитрација на магистрала
    - повеќе од модул ја контролира магистралата
    - централизирана арбитрација
        - еден хардверски уред го контролира пристапот до магистралата – КОНТРОЛЕР НА МАГИСТРАЛАТА (АРБИТЕР)
        - може да е дел од CPU или посебен модул
    - дистрибуирана арбитрација
        - секој модул може да ја присвои магистралата
        - контролна логика на сите модули
- Timing
    - координација на настаните на магистралата
    - синхроно

        - настаните се одредени според текст сигналите
        - контролната магистрала ја вклучува и clock линијата
        - едно 1-0 е циклус на магистралата
        - синхронизирани на растечка ивица и 1 циклус по настан
    - асинхроно
        - еден настан на магистралата зависи од друг претходен настан
- Тип на трансфер на податоци
    - Read
    - Write
    - Read – modify – write
    - Read – after – write
    - Block

4 Инструкциско множество – карактеристики и функции

- Инсатрукциско множество = комплетна колекција од инстукции кои ги разбира еден CPU
  - машински код, бинарен, обично претставен преку асемблер.
- Елементи на инстукција
  - Операциски код(opcode) = направи го ова
  - Референца кон изворен операнд = операндите
  - Референца кон резултатен операнд = резултатот
  - Референца кон следна инструкција = потоа направи го ова

Инструкциски формат

Opcode	Operand reference	Operand reference
4 bit	6 bit	6 bit = 16 bit

- Операндите се наоѓаат во:
  - главна меморија (или виртиуелна или кеш)
  - CPU регистри
  - Непосредно
  - В/И уреди
- Претставување на инструкции
  - машински код: секоја инструкција има уникатна низа од битови
  - за човечко разбирање: симболичко претставување

    пр. ADD, SUB, LOAD, MUL, DIV, STOR
  - и операндите може да се претставата вака:

    пр. ADD A, B (додај го B на A)
- Типови на инструкции
  - Обработка на податоци
    - аримтеички и логички операции
    - најчесто податоците се наоѓаат во регистри

- Складирање на податоци
  - пренос на податоци од/во регистри и главна меморија
- Пренос на податоци
  - В/И инструкции
- Контрола на текот на програмата
  - тестрирање и гранење

- Број на адреси
  - 3 адреси (ADD B, C, A A=B+C)
    - операнд 1, операнд 2, резултат
    - може да има и 4та-следна инструкција (обично е имплицитен -во PC) – не е вообичаено
  - 2 адреси (ADD A, B A=A+B)
    - една адреса претставува операнд и резултат
    - намалување на должината на инструкцијата
    - потребно е дополнителна работа, привремено складиште за дел од резултатите
  - 1 адреса (карактеристично за раните машини)
    - имплицитно втора адреса
    - обично е регистар (акумулатор AC)
  - 0 адреси
    - сите адреси се имплицитни
    - користи магацин (stack) кој работи на last-in-first-out база
  - Повеќе адреси-покомплексни инстукции
    - повеќе регистри
    - операциите меѓу регистри се побрзи
    - помалку инструкции во програмата
  - Помалку адреси-поедноставни инструкции
    - повеќе инструкции во програма
    - побрзо земи/изврши на инструкциите
- Главни прашања при дизајнирање на инструкциски сетови
  - Репертоар на операции
    - Колку операнди ?

- Што можат да направат ?
- Колку се комплексни ?

- Податочни типови
  - инструкциски формати
  - должина на полето за opcode
  - број на адреси
- Регистри
  - број на достапни регистри во CPU
  - кои операции во кои регистри
  - Режим на адресирање
  - RISC vs CISC

- Типови на операнди
  - Адреси-тип на податоци
  - Броеви
    - најчести нумерички типови (binary integer/binary fixed point, binary floating point, decimal)
    - цели/со подвижна запирка
    - знаци: 1100(+), 1101(-) - SIGN DIGIT
  - Знаци(карактери)
    - ASCII-American Standard Code for Information Interchange (8 bit)
    - IRA-International Reference Alphabet (8 bit)
    - EBCDIC-Extended Binary Coded Decimal Interchange Code (IBM mainframes, 8 bit)
  - Логички податоци
    - податоците се расцепкуваат на 1 bit
    - битови и знаменца
- x86 Податочни типови
  - репрезентација на податоци:

8 bit = Byte, 16 bit = word, 32 bit = double word, 64 bit = quadword, 128 bit = double quadword

- RISC = Reduced Instruction Set Computer

  CISC = Complex Instruction Set Computer
- Адресирањето е преку 8 bit единица

- Зборовите не мора да се подредени на парни адреси
- 32-bit на податочна магистрала = единици од double word на адреси деливи со 4
- little-endian style = најнезначајниот бит е зачуван на последната адреса
- negative integers → two's complement
- SIMD (single-instruction-multiple-data) data types
  - packed byte and packed byte integer
  - packed word and packed word integer
  - packed doubleword and packed doubleword integer
  - packed quadword and packed quadinteger
  - packed single-precision floating-point and packed double-precision floatig-point (4x32→128 2x64→128)

- ARM податочни типови
  - 8 (byte), 16 (halfword), 32 (word) bits
    - Halfword и word – треба да се подредени
    - постои можност и за неподреден пристап
  - за сите типови е подржана интерпретација на цел број без знак и цел број со знак во двоен комплимент
  - повеќето имплементации немаат хардвер за броеви со подвижна запирка
    - се имплементираат софтверски
    - штеди батерија и простор
    - опционален ко-процесор за оваа намена кој работи со IEEE 754 со единечна и двојна прецизност
  - Endian подршка
    - E-bit во сисетемскиот контролен регистар
    - под програмска контола

- Типови на операции
  - Пренос на податоците
    - мора да се дефинира: извор, одредиште, количество податоци
    - може да има различни инструкции за различни преноси (IBM 390) или една инструкција и различни адреси (VAX)
  - Ариметички

- собирање, одземање, множење, делење (за цели броеви со знак и понекогаш за броеви со подвижна запирка)
- може да вклучува: апсолутна вредност, негација, инкрементација, декрементација

- Логички
  - операции врз битовите (NOT, AND, OR, XOR ...) - Shift Rotate
- Претворба

  пр. декадно во бинарно
- Влезно/излезни
  - може да бидат специјални инструкции или инстуркциите да се користат за пренос на податоци (мемориска мапирање)
  - може да се прави со одвоен контролер (DMA)
- Системска контрола
  - привелигирани инструкции (CPU треба да е во специјална состојба kernal режим)
  - резервирани за употреба на оперативниот систем
- Пренос на контрола
  - гранење (jump инструкција, if clause)
  - скокни (skip, обично за правење јамки)
  - повик на процедура (функции C++)
  - повик за обработка на прекин

- Редослед на бајтите
  - во кој редослед се читаат броевите кои зафаќаат повеќе од 1 бајт
  - Endian концепт
    - Big endian - најзначајниот бајт е на најмалата адреса
    - Little endian – најмалку значајниот бајт е на најмалата адреса

Неможе да се каже кој endian е супериорен систем

little endian: Pentium (x86), VAX, Internet

big endian: IBM 370, Motorola 680x0 (Mac), RISC

5. Инструкциско множество : режим на адресирање и формати

- Режими на адресирање

● Непосредно адресирање

> операндот е дел од инструкцијата

Пр. ADD5 (додади 5)

> нема мемориска референца за земање податок (брзо, ограничен опсег)

● Директно адресирање

> адресното поле содржи адреса на операндот

Пр. ADD A (додади ја содржината на A)

>1 мемориска референца за пристап до податок

>Ограничен адресен простор

● Индиректно адресирање

> Индиректно адресирање мемориска ќелија кон која покажува адресното поле чија содржина е адресата на (покажувачот кон) операндот

Пр. ADD A (додади ја содржината на ќелијата кон која покажува содржината на A)

>голем адресен простор (2^n , n = должина на зборот)

>може да е вгнездено [EA = (((A)))]

>повеќе мемориски пристапи до операндот => побавно

● Регистерско адресирање (директно) ER = R

> операндот се чува во регистар кој се именува во адресно поле

> ограничен број на регистри, потребно е помало адресно поле (пократки инструкции, побрзо земање инструкции)

> нема мемориски пристап , брзо извршување , ограничен адресен простор

> бара добро асембирско програмирање или компајлер

● Регистерско индиректно адресирање ER = (R)

> операндот е во мемориска ќелија кон која покажува содржината на регистерот

> голем адресен простор (2^n) , -1 мемориски пристап во споредба со индиректно адресирање

● Адресирање со поместување EA = A + (R)

> Адресното поле има 2 вредности

A = основна вредност – База

R = регистар кој го содржи поместувањето

- Релативно адресирање

> верзија на адресирање со поместување

> EA = A + (PC)

Земи операнд од A ќелијата почнувајќи од локацијата каде моментално покажува Program Counter

- Базно – регистерско адресирање

> A го чува поместувањето , R го чува покажувачот кон базната адреса (може да е експлицитна или имплицитна)

- Индексно адресирање EA = A + R

> A = база , R ≠ поместување

> Добро за пристап до поле

Postindex : EA = (A) + (R)

Preindex : EA = (A+(R))

- Магацинско адресирање (stock)

> опреандот е (имплицитно) на врвот на магацинот

- Скоро сите моредни компјутерски системи подржуваат повеќе начини на адресирање

x86 Режим на адресирање

- Виртуелна или ефективна адреса е поместување во сегмент

SHORT ADRESS + DISPACEMENT = LINEAR ADDRESS

- Режими (SR – segment register , I – index register , B – base register , S – scaling factor)

> Immediate Operand = A

> Register Operand LA = R

> Displacement LA = (SR) + A

> Base LA = (SR) + (B)

> Base w/ Displacement LA = (SR) + (B) + A

> Sealed Index w/ Displacement LA = (SR) + (I) xs + A

> Base w/ Index and Displacement LA = (SR) + (D) + (B) + A

> Base w/ Scaled Index & Displacement LA = (SR) + (B)

ARM режими на адресирање вчитај/запиши

- само инструкции кои референцираат меморија
- индиректно преку базен регистар + поместување
- Preindex и Postindex
- Базниот регистар се однесува како индексен регистар
- Поместувањето е или непосредна вредност или друг регистар (ако е регистар можно е и скалирање)

Инструкциски формати

- изглед (поделба) на низата битови во инструкцијата
- вклучува opcode и операнд (имплицитен/експлицитен)
- вообичаено има повеќе од 1 инструкциски формат во едно инструкциско множество

Должина на инструкција

- под влијание но и влијае врз :

  > големината и организацијата на меморијата

  > структурата на магистралата

  > комплексноста на процесорот и неговата брзина

- балансирање помеѓу репертоар на моќни инструкции и заштеда на простор

Алокација на битови (Фактори за алокација)

- број на режими на адресирање
- број на операнди
- регистри наспроти меморија
- број на регистерски множества
- опсег на адреси
- грануларност на адреси

Асемблер

- машината запишува и разбира бинарни инструкции
- напорно и подложи на грешки

1. ● се користи хекса наместо бинарно

> код како серија од линии (хекса адреси и мемориски адреси)

> треба да се преведе автоматски со помош на програма

2. ● се додаваат симболички имиња и мнемоници за инструкции

> при полиња по мениа (адреса на локацијата , opcode со 3 букви ако има мемориска референца : adresa)

> потребна е покомплексна програма за превод

● симболички адреси

1 > првото поле (адреса) сега е симболичко

2 > мемориската референца во трето поле е симболичка

1 + 2 = асемблерски јазик (кој се преведува од асемблер)

● Асемблерите се користат за некои видови системско програмирање (компајлер, I/O процедури)

6. MIPS Instruction set

Вовед

● инструкции = зборовите во компјутерскиот јазик

● инструкциско множество = компјутерскиот речник (сите валидни зборови)

● инструкциско множество развиено од MIPS Technologies

● популарни инструкциски множества

> ARM v7 (слично со MIPS) 32 bit adress size

> Intel x86

> ARM v8 (слично со MIPS) 64 bit adress size

- comments ! – 1 instruction per time

PROGRAM ->(COMPILER)-> ASSEMBLY CODE ->OS LOADER-> MEMORY

MIPS инструкции

● формат со 3 операнди

Пр. op destination , src1 , src2

Design Principle 1 : Едноставноста ја сака регуларноста !! REGISTER SIZE IN MIPS : 32 BIT (1 WORD)

- броеви со знак -> двоен комплемент

  > sign extend – преостанатите битови до 32 се дополнуваат со најзначајниот

Design Principle 2 : Smaller is faster! max 32 registers

$S0, $S1.. – REGISTERS TO VARIABLES

$t0, $t1.. – TEMPORARY REGISTERS

- типови инструкции

  > data operations (аритметички – add , sub/ логички – and, or, not, xor)

  > data transfer (Load, Store)

  Memory ←→Register

  > sequencing (branch – условен , < , > , == / Jump – безусловен , goto)

! MIPS E big-endian

- непосредни операнди (константи)

  > наместо уште една lw инструкција, има алтернативни immediate инструкции

$zero – register 0

Design Principle 3 : Честите случаи направи ги да бидат брзи

- формати на инструкции

Design Principle 4 : Добар дизајн бара добри компромиси

  > сите инструкции да бидат со иста должина и да имаат ист формат

  > компромис : повеќе инструкциски формати за различни инструкции

Regular	● R – format : 3 регистри како операнди
Data transfer/Immediate	● I – format : 2 регистри + непосредна вредност / адреса
Jump	● J – format : константна адреса

6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
op	rs	rt	rf	shamt	funct
op	rs	rt	Adress/Immediate (16)		
op	target adress (26b)				

Columns – 1● R – format
2● I – format
3● J – format

Rows - ● аритметички/логички операции
● Load/Store , Branch , Immediate
● Jump

$S0 - $S7 - 16 – 23 register respectively

$t0 - $t7 - 8 – 15 register respectively

35 lw , 32 – add , 24 – sub , 43 – sw , 8 – addi = funct

легенда :

- op – операција и формат на инструкцијата
- rs – прв регистар – операнд
- rf – втор регистар – операнд
- rd – дестинациски регистар (се сместува раз.)
- schmt – shift
- funct – функција (варијантата не op)

- *логички операции*

  - операции кои овозможуваат работа со битови (инаку обично се работи со цели зборови)
  shift left – sll – множење со 2 (целобројно)
  shift right – srl – делење со 2
  (AND) и – and, andi
  (OR) или – or, ori
  (NOT) не – nor

- *донесување одлуки*

  if : beq r1,r2,L1 #if(r1=r2) geto L1;
  !if: bne r1,r2,L1 #if (r1!=r2) geto L1;
  for/while: Loop: j loop инструкција (целов прво) #j e jump

- *инструкции за споредување*

  set on (<) slt : slt $t0, $s3, $4 #$t0=1 ако $s3<$S4
  slti : slti $t0, $2, 10 #$t0=1 ако $s2<10
  sltu : sltu $t0, $s3, $s4 #исто како slt ама unsigned
  switch :
  - може да се имплементира со up if then наредби
  - се користи jumptable – низа од адреси, оттаму се вчитува во регистар адресата
  - JR – безусловно се скока до таа локација

- *поддршка за процедури во хардверот*

  процедура - извршува одредена задача заснована врз внесеите параметри
  при внесување на процедурата, програмата следи 5 чекори:
  1 - сместување на параметрите таму каде што процедурата може да се пристапи
  2 – префрлање на контролата на процедурата
  3 – доделување ресурси за складирање
  4 – извршување на посакуваната задача
  5 – сместување на вредноста на резултатот
  6 – враќање на контролата кон точката на повикување

  $a0 - $a3 – 4–7 registers аргументи

$U0 - $U1 – 2-3 register резултати (вредности)
$ra - 31 register повратна адреса

- повик на процедура jal Адреса_на_процедура > jump-and-link
  1 - во $ra ја запишува адресата од каде што е повикана процедурата
  2 – скока до адресата на процедурата > враќање до повикувачко место jr $ra

- Program Counter (PC) – регистар каде се чува адресата на инструкцијата што во моментот се извршува
  - jal во $ra ја запишува вредноста PC+4
  $t8 + $tg – 24-25 registers
  $sp – stack pointer – 29 registers

- користење на повеќе регистри
  stack – податочна структура во која се чуваат регистрите
  - редица од зборови , last-in-first-out
  - stack покажувач кон последната адреса ($sp-29th)
  - складирање во stack – PUSH (вметни)
    вадење од stack – POP (извади)
  - stackот расте од повисоките адреси кон пониските

- вгнездени процедури
  повикувачка процедура (A)
  - се грижи за регистрите $a0 $a3 и $t0 - $t9
  - доколку и се потребни ги прелева во stack
  повикувачка процедура (B)
  - се грижи за регистрите $s0 - $s7 и $ra

  $gp – global pointer (for static variables) 28 reg

- *алокација на големи податоци*
  Stack се користи и за чување на големи податочни објекти (низи и структури)
  - активациски запис / рамка (frame) – сегмент каде се чуваат овие објекти (објекти) заедно со регистрите (рамка од процедурата)
  $fp – frame pointer (покажувач кон рамка) 1 збор на почетокот (додека $sp покажува на крајот) 30 reg
  текстуален сегмент – сегмент од UNIX објектната датотека која содржи код во машински јазик за рутини во изворната датотека
  - ако процедуратра очекува повеќе од 4 параметри, тие треба да бидат сместени над $fp

- Организација на меморија и извшување на инструкции

- Пресметување на наредната адреса
  I – format : PC + 4 + 16b immediate
  J – format : PC (first 6 bits) 26bit target address

- MIPS адресирање
  - immediate addressing : операндот е константа
  -register addressing : операндот е регистар
  -base/displacement addressing : операндот е сума од регистар и константа
  -PC – relative addressing : адресата е 6pc bits и 26bits target address (J – format)

- Организација на меморијата
  -малку податоци во регистрите
  lw, sw – инструкции за трансфер
  -за да се пристапи до некој збор мора да се знае адресата
  -адресите се деливи со 4 (порамнување)
  -најпотребните променливи се сместуваат во регистри (бидејќи се побрзи и со поголема принудена моќ)
  -останатите ги префрла во меморијата
  $at – assembler temporary (1 register)
  инструкции:
  add, sub, add, wult, div, mfhi (move from high), mflo (move from low), and, andi, or, ori, xor, nor, sll, srl
  ! HIGH : % LOW : / - при делење
  32bit(o) 32bit(p) – при множење

7. Структура и функција на процесорот

- Што прави процесорот ?
  1. Земи инструкција
  2.Интерпетирај ја инструкцијата
  3.Земи податок
  4.Обработи го податокот
  5.Запиши податок
  -За сето ова потребно е привремено зачувување на податоци – има потреба од мала внатрешна меморија (регистри)
- Организација на регистрите
  -кориснички видови регистри
  -може да се референцираат со асемблер
  -категории:
  -регистри за општа намена (general purpose)
  -податочни регистри (data)
  -адресни регистри (сегменти – segment pointers; индексни – index registers ; магацин stack

pointer)
-целовни кодови (знаменца – flags)

- *Контролни и статусни регистри*
  -ги користи контролната единица
  -Program counter (PC) – contains the address at the instruction to be fetched
  -Intstruction register (IR) – contains the instruction from was recently fetched
  -Memory address register (MAR) – contains the address of a location of memory (ADDRESS BUS)
  -Memory buffer register (MBR) – contains a word of data to be written to memory of the word most recently read. (DATA BUS)
  MBR <-> REGISTERS
  -Program status word (PSW) : sign, zero, carry, equal, overflow, interrupt (enable/disable), supervisor
- *Инструкциски циклус*
  indirect cycle – при индректното адресирање кај operand fetch има loop
  IR determines if the operand is using indirect addressing, if so the rightwest N bits of the MBR are transfered to MAR. The CU registers a memory read and the desired address is put into NBR.
- *Протечност (pipeline) на инструкциите*
  pipelining – it is worked on various products simulataneously, new inputs are accepted at one end before previously accepted inputs appear as outputs at the other end
  -intruction prefetch / fetch overlap
  проблеми при дуплирање на execution rate
  -времето на извршување е обично подолго од времето на земање инструкција (извршувањето вклучува читање и запишување операнди)
  -појава на условен скок (branch)
  секогаш се прави земање на следната инструкција, па ако условот за скок не е исполнет не се губи време.
  поделба на повеќе фази
  - FETCH INSTRUCTION (FI) : Read the next instruction into a buffer
  -DECODE INSTRUCTION (DI) : determine opcode $ operands
  -CALCULATE OPERANDS (CO) : одреди ефективна адреса на операндите
  -FETCH OPERANDS (FO) : од меморија
  -EXECUTE INSTRUCTION (EI) : execute $ store result
  -WRITE OPERAND (WO) : store result in memory

8. **Внатрешна Меморија**

❖ Карактеристики на меморија

Локација	Перформанси
Капацитет	Физички тип
Единица на пренос	Физички карактеристики
Метод на пристап	Организација

- Локација
  - CPU,внатрешна,надворешна
- Капацитет
  - Големина на збор=природна единица на организација
  - Број на зборови (бајти)
- Единица на пренос
  - Внатрешна
    - Обично одредена според ширината на податочната магистрала
  - Надворешна
    - Обично блок кој е многу поголем од збор
  - Адесибилна единица
    - Најмалата локација која уникатно може да се адресира
    - Внатрешен збор
- Метод на пристап
  - Секвенцијално
    - Почни од почеток и читај по ред
    - Времето на пристап зависи од локацијата на податоците и претходната локација(пр.Лента)
  - Директно
    - Индивидуални блокови имаат уникатни адреси
    - Пристапот е со скокање во близината плус секвенцијално пребарување
    - Времето на пристап зависи од локацијата на претходната локација
  - Случајно
    - Локациите точно се идентификуваат со индивидуални адреси
    - Времето на пристап е независно од локацијата или претходниот пристап (пр.RAM)
  - Асоцијативно
    - Податокот се лоцира со споредба на содржината со складиштето
    - Времето на пристап е независно од локацијата и претходниот пристап (пр.Кеш)
- Перформанси
  - Време на пристап – латентност
    - Времето помеѓу давање адреса и добивање на точен податок
  - Време на мемориски циклус
    - Потребно е време мњморијата да се “опорави” пред следниот пристап

- Брзина (рата) на пренос
  - Брзината со која може да се премести додаток

➢ Физички типови
- Полупроводнички – RAM
- Магнетни – диск и лента
- Оптички- CD и DVD
- Други – меур, холограм

➢ Физички карактеристики
- Decoy- Губење на информацијата
- Volatility- можност да се брише
- Потрошувалка на енергија

➢ Организација
- Физичко распоредување на битовите во зборови
- Не е секогаш очигледно (пр. Преплетено)

❖ Крајна Линија
- Големина – капацитет , брзина – времето е пари
  - Побрза-→ поскапа по бит
  - Поголема -→ поефтина по бит, поспора

❖ Хиерархиска листа
1. Регистри
2. L1 кеш
3. L2 кеш
4. Главна меморија
5. Дисков кеш
6. Диск
7. Оптички медиуми
8. Лента

❖ Мемориска Хиерархија
- Регистри (во CPU)
- Внатрешна(главна)меморија – RAM
  - Може да се вклучи 1 или повеќе нивоа на кеш
- Надворешна меморија
  - Складиште за поддршка и чување копии

❖ Полупроводна меморија
➢ RAM
- Лошо име бидејќи сите полупроводни мемории се со случаен пристам (Random)
- Читање/запишување/volatile (работи само на струја)
- Привремено складиште
- Статистичка/динамичка
- Се користат електорнските сигнали

➢ Dynamic RAM (DRAM) – tendency to look away
- Битови се запишуваат како полнеж во кондензатор
- Полнежите слабеат, има потреба од освежување дури и кога има напојување

- Поедноставна градба, помала по бит, поефтина, побавна
- Потребни се кола за освежување
- Главна меморија
- Аналогна= новото на полнеж ја одредува вредноста
- РАБОТА:
  - Адресираната линија е активна кога битот се чита или запишува (затворен транзисторски прекинувач- проток на струја)
  - Запишување: 1) напон на бит линијата (1- high, 0-low)
    2) потоа сигнализација на адресната линија (пренесува полнеж во кондензаторот)
  - Читање: 1)избрана адресна линија (се вклучува транзиторот)
    2) полнеж од кондензаторот преку бит линијата кон засилувачот (се споредува со референтната вредност за дали е 0 или 1)
    3) мора да се освежи полнежот на кондензаторот

➢ Static RAM (SRAM)
- Користи исти логички елементи како процесорот
- Битовите се зачувани како вклучени/исклучени прекинувачи
- Нема слабеење на полнеж, покомплексна градба
- Поголема по бит,поскапа , побрза, кеш
- Нема потреба од кола за освежување
- Дигитална= користи флоп – флопови

❖ SRAM vs DRAM

➢ Непостојани- потребно им е напојување за чување на податоците
➢ Динамитски келии:
- Поедноставни за градба, помали
- Погусто спакувани, поевтина
- Имаат потреба од освежување
- Поголеми мемориски единици (pr. main memory)

➢ Статички келии:
- Побрзи
- Пр. Кеш

❖ ROM

- Перманентно складиште= постојана (nonvolatile)
- Само за читање не може да се запишува
- За микропрограмирање
- Библиотечни процедури
- Системски програми (BIOS)
- Функциски табели

Important applications

- Се запишува во текот на производството, многу скапо за малку користење

➢ Типови ROM
- Програмерски ROM – PROM
  - Потребна е специјана опрема за програмирање

- Бришлив програмерски ROM- EPROM
  - Read & written electrically, must be erased before a write operation.
  - Бришење со UV светлина
- Електрично бришлив ROM – ERPROM
  - Многу подолго се запишува од чита
- Флеш Меморија – Flash Memory
  - Uses electrical erasing technology (entire & blocks)
  - High density

Read-mostly memory

- Организација
  - 16Mbit чип може да е организиран како 1M од 16 битови зборови
  - Бит по чип систем има 16 групи од 1Mbit чипови со 1 на секој збор во чип 1 итн.
  - 16Mbit чип може да се организира како 2048X
    - 2048 X 4bit поле
    - Го намалува бројот на адресни пинови
    - Мултиплексирана адреса на ред и колона
    - 11 пина за адресирање (2^{11}=2048)
      - Со додавање на уште еден пин се зголемува опсегот на вредности на x4 капацитетот
- Преплетена Меморија –
  - Колекција од DRAM чипови групирани во мемориски банки
  - Банките независно опслужуваат барања за читање/запишување
  - К Банки може да опслужуваат К барања едновремено
- Корекција на грешка
  - Хардверска грешка – перменантен дефект
  - Софтверска грешка – случајна, не деструктивна, нема перманентно уништување на меморијата
    - Детектирање со Хакингов код
- **Напредна** DRAM **организација**
  - Основен DRAM ист уште од првите RAM чипови
  - Подобрен DRAM- содржи мал SRAM (Ја чува последно прочитаната линија како КЕШ!!)
  - Кеш DRAM - поголема SRAM компонента, се користи како кеш сериски бафер
  - Понапредни DRAM верзии: SDRAM, DDR-DRAM, RDRAM
  - Традиционалниот DRAM е асинхрон
- **Синхрон** DRAM (SDRAM)
  - Пристапот се синхронизаира со надворешниот такт
  - Адресата се дава на RAM кој го наоѓа податокот (CPU чека конвенционален DRAM)
  - Бидејки SDRAM ги претставува податоците со системскиот такт, CPU знае кога податокот ќе биде спремен
  - CPU не мора да чека, може да прави нешто друго
  - Burstmode (Режим на изливи) – овозможува SDRAM да постави проток од податоци и да излие во блок
- RAMBUS DRAM
  - Прифатено од Intel Dentium&Itanium, главен соперник на SDRAM
  - Вертикалното пакување – сите пингови на една страна

- Размена на податоци преку 28 жици ≤ 12см голжини
- Магистралата адресира до 320 RDRAM чипа со 1.6 bps

- DDR SDRAM (Double-data rate)
  - Обичен SDRAM може да испраќа податок само еднаш по такт, додека DDR SDRAM може двапати по такт (еднаш на растечка, еднаш на опаѓачка ивица)
  - Desktop computers & servers
  - DDR2 – односот помеѓу внатрешен и надворешен такт е 2,4 пати побрз
  - DDR3 – 2 пати побрз од DDR2, 4 пати побрз такт
  - DDR4(2014) – поголема густина на пакување, повисока фреквенција за работа
    - Double data rate 4-genera(нешто) synchronous dynamic RAM
    - DIMMs со капацитет до 512GB – dualin-line memory module

Одлуки при дизајн на проточност

1. Во секоја фаза има одреден overhead за преместување податоци и припрема:
   - ова треба значајно да го зголеми времето потребно за извршување на инструкцијата
   - посебно е значајно за секвенцијалните инструкции кои се логички зависни
2. Контролната логика потребна за работа со мемориските и регистреските зависности за оптиматизација на проточност енормно расте со бројот на фази.

- Ова може да доведе до ситуација кога контролната логика е покомлексна од самите фази кои ги контролира

Pipeline Hazard(Pipeline bubble)

- Occurs when the pipeline must stall because conditions don't allow continued execution
  1. Хазард на ресурски(resource hazard)
     - Две или повеќе инструкции во цевката користат исти ресурси(инструкциите мора да се извршуваат сериски наместо паралелно)
     - Structural hazard
  2. Податочен хазард(data hazard)
     - Конфликт во пристап на локација не операнд(инструкцијата зависи од операнд од претходната уште незавршена инструкција.
     - Типови:
       - Read after write-вистинска зависност
       - Write after read-антизависност
       - Write after write-излезна зависност
  3. Контролен хазард(control hazard)
     - хазард на гранење(branch hazard)
     - се прави погрешна одлука при предвидување на гранењето

Справување со гранење

1. Multiple streams

- Да се земат во предвид два можни исходи во посебни протоци

2. Prefetch branch target

- Се презема и следната инструкција и целта на скокот

3. Loop buffer

- Содржи н последно земени инструкции во низа
- Посебно добро за јамки што може да ги собере во баферот

4. Branch prediction

Статички(не зависат од историјата):

- Predict never taken
- Predict always taken
- Predict by opcode

Динамички(врз основа на стари исходи):

- Taken/not taken switch
- Branch history table

5. Delayed branch

Паралелизам на ниво на инструкција, Суперскаларни процесори

Што е superscalar?

- Вообичаени инструкции(аритметички,прочитај/запиши,условен скок) може да се иницираат и извршат независно
- Еднакво применливо и во RISC и во CISC
- Why superscalar?
    - Повеќето инструкции се на скаларни вредности
    - Со подобрувањето на овие фракции се добива севкупно подобрување

Суперпротогеност-superpipelined

- Многу фази во цевката траат помалку од половина такт
- Со дуплирање на внатрешната брзина на тактот се добиваат 2 задали по надворешен такт
- Суперскаларите овозможуваат паралелни земи и изврши инструкции

Ограничувања

- Паралелизам на ниво на инструкција=можност за извршување на повеќе инструкции паралелно
- Оптимизации на ниво на компајлер,хардверски техники
- Ограничени од:
    1. Вистинска податочна зависнот WAR (втората инструкција зависи од првата)- не се елиминира
    2. Процедурална зависност(инструкциски зависни од условен скок)
    3. Конфликти на ресурси(потреба од истиот ресурс кај 2 или повеќе инструкции,може да се елиминира)
    4. Излезна зависност
    5. Антизависност

Дизајн

- Паралелизам на ниво на инструкција
    - Инструкциите во низа се независни
    - Може да се преклопи извршувањето
    - Под влијание на податочната и процедуралната зависност, кои пак зависат од инструкциското множество
- Мапински паралелизам
    - Можност да се искористи паралелизам на ниво на инструкции
    - Под влијание на бројот на паралелни проточни цевки

Суперскаларна имплементација

- Едновремено земање повеќе инструкции
- Логика за одредување на вистинските зависности кои вклучуваат регистерски вредности
- Механизам за комуникација на овие вредности
- Механизми за иницирање на инструкции паралелно
- Ресурси за паралелно извршување на повеќе инструкции
- Механизми за извршувањ на процесната состојба во точен редослед

КЕШ

Мемориска хиерархија

1. Inboad memory
   -registers,cache,main memory
2. Outboard storage
   -magnetic disc,cd,dvd
3. Off-line storage
   -magnetic tape,MO,WORM

Помала цена по бит
Поголем капацитет
Поголемо време за пристап
Помала фреквенција за пристап од CPU

Локалност на референцирање = за време на извршување на програмата, мемориските референци се обично кластерирани пр.јамки

Кеш

- Мало количество на брза меморија помеѓу нормалната главна меморија и CPU
- Може да е лоцирана на CPU чип или модул

Организација

n-адресни линии, 2^n зборови(меморија)

1. Главна меморија се дели на M блокови со по K зборови, $M=2^n/K$
2. Кешот се состои од : m блокови(линии) кои содржат K зборови + таг($m<<M$). Ако се прочита збор од главната меморија, тој блок каде што се наоѓа зборот се референција во една линија во кешот. Со тагот се означува за кој блок стнаува збор.

Работа на кеш

- CPU бара содржина на мемориска локација
- Најпрво се преоверува кешот
  - Ако е таму, се зема од кешот=погодок
  - Ако не е таму, се чита потребниот блок од меморија=промашување
- Кешот вклучува ознаки за точна идентификација на блокот од главната меморија

- Дизајн на кеш
  - Адресирање
  - Фолемина
  - Функција на мапирање
  - Алгоритам за замена
  - Полиса на запишување
  - Големина на блок
  - Број на кешови
- Кеш адресирање
  - Локација меѓу процесорот и единицата за управување со виртуелна меморија

    2 меѓу MMU и главна меморија
- Логички кеш (виртуелен кеш) чува податоци со користење на виртуелни адреси
  - Прочесорот пристапува до кешот директно
  - Пристапот до кешот е побрз,пред MMU превод на адресата +(good thing)
  - Виртуелните адреси го користат истиот адресен простор за различни апликации,мора да се испразни кешот на секоја промена на контекстот (bad)
  - Физичкиот кеш ги чува податоците со користење на физичките адреси на главната меморија
- Големина
  - Чена =поголем кеш е скап
  - Брзина = поголем кеш е побрз (до дадена точка) проверка на кешот за податок одзема време
- Функција на мапирање
  - Директно мапирање
    - Секој блок од клавната меморија се мапира во само 1 кеш линија , ако блокот е во кешот ,тој мора да е на едно одредено место
    - Адресата е во 2 дела
    - Најмалку значајно W бита го одредуваат уникатниот збор
    - Најзначајнита S бита одредуваат еден мемориски

    Address length =STW ,No of blocks = 2^s ,Size of tag=s-r , m=s^r ,s^{r+w}
- For d aganst b
  - Едноставно евтино
  - Фиксна локација за даден блок = ако програма повторувачки пристапува од 2 блока кои се малираат во иста линија ,бројот на промашување во кешот е многу голем
- Жртвен кеш (victim cache)
  - Помала казна за промашување

- Памти што било отфрлено ,се искоритува повторно со помала казна
- Целосно асоцијативен,и-16 кеш линии
- Меѓу директно мемориран L1 кеш и следното меморирано ниво

○ Асоцијативно меморирање
- Блок од главната меморија може да се вчита во било која кеш меморија
- Мемориската адреса се инпретира како ознака избор
- Бребарувањето на кешот станува скапо

○ Меморирање со асоцијативни множевства (директно+асоцијативно)
- Кешот с едели на даден број множевства со даден број линии
- Даден блок се меморира во било која линија во дадено множевство

Директно

Ознака S-r 8	Slot r 14	Збор w 2

Асоцијативно

Ознака 22	Збор 2

Асоцијативно множевство

Ознака 9	Множевство 12	Збор 2

○ Алгоритми за замена (при полн кеш)
- Директно меморирање :нема извор,секој блок се меморира во само 1 линја и таа се заменува
- Асоцијативно и асоцијативни множевства
  - Хардверски имплементиран алгоритам
  - LRU=Last Recontly Used
  - FIRO =First in First Out
  - LFU = Last frequently used

○ Полиси за запишување
- Не смее да се препише блок во кешот ако главната меморија не е освежена со промените
- Повеќе CPU меже да шират кешови
- I/O може директно да ја адресира главната меморија

Решенија:
- Запиши со пренос-Write throught
  - Сите запишувања одат и во главната меморија и во кешот

- Повеќе CPU можат да ја игнорираат главната меморија за да ја чуваат локацијата (за CPU)кеш :Точен
- Многу собрќај се забавува запичувањето

- Запишување наназад
  - Промените инуцијативно се прават само во кешот
  - Се користи Update bit за кеш слотот за да е означи постоење на промена
  - Ако блокот треба да се замени ,запиши во главната меморија сако ако битот е 1
  - Другите кешови не се синхронизирани
  - I/O мора да пристапува до главната меморија преку кешот
  - 15% од меморијата референции се запишување

- Кохерентност на кешот
  - Сите кешови (за сите CPU) да испиша т валидни податоци

← следење на магистралата за запишување со пренос

Approaches ← хардверска транспортација

← следена меморија без кеш

- Големина на линија
  - Запиши го не замо баранит збор туки и дадениот број на соседни зборови
  - Зголемена големина на блокот првично го зголемува % на податоци = принцип на локалност
  - % на податоци ќе се намали како што блокот станува поголем
  - Веројатност аз акористење на новоземаната информација останува помалку од веројатноста за повторно управување употреба на земаната
- Поголеми блокови
  - Намали го бројот на блокови кој ги собира кешот
  - Податоците се пребришуваат кратко откако ќе се заврши
  - Секој дополнителен збор е помалку локален па помалку веројатно е дека е потребен
  - Нема дефинирано оптимална вредност (8-64 в
- Број на кешови
  - Кешови на повеќе нивоа
    - Големата густина на логиката овозможува кеш на чипот
    - Побрзо од пристапот на магистралата
    - Ја ослободуба магистралата за други преноси
    - Вообичаено се користи кешот не и кеш одвоен од чипот

- Л1 на чип ,Л2 одвоен статички RAM
- Л2 пристап е многу брз од DRAM или ROM
- Л2 чисто користи одвоено податочна патека
- Л2 сега може да е и не чип

- Унифицирани наспроти поделени кеш мемории
  - Еден кеш за податочен и инструкции или 2 ,еден за податоци и еден за инструкции
  - Предности за унифицирање
    - Повисока рата на погодок
    - Ја балансира отовареноста на земање инструкции и податоци
    - Само еден кеш за дизајни и имплементации

10. Виртуелна меморија

1. Оперативен систем е програма која ги контролира апликациите и работи како медијатор меѓу корисникот и компјутер (како хардвер).

- Употребливост: ОС го прави компјутерот поедноставен за употреба.

- Ефикасност: ОС овозможува подобра употреба на компјутерските ресурси.

2. Услуги на оперативниот систем

- Дизајн на програми = editors & debuggers to assist the programmer in creating the pograms.

- Извршување на програми = ги извршува сите инструкции

- Пристап до В/И уреди

- Контролиран пристап до датотеки = protection mechanism

- Пристап до системот

- Детекција на грешка и одговор

- Водење статистички информации = runtime, response time

3. Управување со меморија

- Uni – program

*Меморијата се дели на два дела

*Еден за оперативниот систем(монитор)

*Еден за програмата која тековно се извршува

- Multi – program

*„Корисникот“ дел се дели и споделува помеѓу активните процеси

4.Проблем се појавува бидејќи 1/0 уредите се многу бавни во споредба со CPU т.ш. дури и во мулти-програмирачки систем, CPU може да „врти во празно“ голем дел од времето.

*Решенија:

- Зголемување на main memory

- Swapping

5. Swapping

*Долгорочна редица од процеси запишана на диск

*Процесите се „заменуваат“ со појава на слободен простор

*Кога процесот ќе заврши тој се вади од главната меморија

*Ако ниту еден од процесите во меморијата не се спремни (пр. В/И блокирани)

- Замени блокиран процес во средната редица

- Стави спремен процес или нов процес

- Swapping е I/O процес, може проблемот да се влоши

6. Partitioning – партиципирање

*Поделба на меморијата во делови за алокеција за процеси (вклучувајќи го и ОС) со еднаква или различна големина.

*Партиции со фиксна големина

- Немора да значи дека сите партиции се еднакви

*Партиции со динамичка големина

Релокација – решение

- Наша гаранција дека процесот ќе се вчита во истото место во меморија
- Инструкциите содржат адреси (локации на податоци и адреси на инструкции(пратење).
- Логичка адреса = релативно на почетокот на програмата

- Физичка адреса = вистинската локација во меморија
- Автоматска конверзација од логика во физичка со користење на базна адреса.

7. Bagging – страничење

*Поделба на меморијата во мали делови од еднаква големина = рамки за страници (frames)

*Поделба на програмите (процесите) во еднакви големи мали делови = страници (pages)

*Алокација на потребниот број рамки на процес

*ОС одржува листа од слободни рамки

*Процесот не бара последователни рамки

*Се користи табела на страници за вадење сметка

8.Виртуелна меморија

*Страничење по потреба – demand paging

- Нема потреба сите страници на процесот да бидат во меморија
- Страниците се носат по потреба

*Page fault

- Бараната страница ја нема во меморија
- ОС мора да ја донесе бараната страница
- Може да треба да исфрли страница за да направи место
- Избери страница за исфрлање врз основа на скорешната историја

*Thrashing

- Премногу процеси во премалку меморија
- ОС го троши целото време на замена
- Се прави малку или никаква вистинска работа
- Светилката за дискот е постојано вклучена
- **Решенија:**
  - добри алгоритми за замена на страници(LRU(eg))
  - намалување на бројот на процеси кои работат
  - поголема меморија
- Нема потреба целиот процес да биде во меморија за да работи.
- Може да се носат страници по потреба
- Може да работат процеси кои се поголеми од вкупната главна меморија
- Главната меморија = реална меморија
- Корисникот/програмерот гледаат многу поголема меморија – виртуелна меморија

9.Translation lookaside Buffer

*Секоја виртуелна мемориска референца предизвикува две пристапи до физичката меморија

- Земи запис од табелата со страници
- Земи податок

*Се користи специјален кеш за табелата за страници = TLB – Translation Lookaside Buffer

- contains page table entries which have been most recently used

10. Сегментација

*Страничењето не е (вообичаено) видливо за програмерот

*сегментацијата е видлива за програмерот

*обично различни сегменти се алоцирани за програма и податоци, сегментите се динамички

*Може да има повеќе програмски и податочни сегменти

11. Предност на сегментацијата

*Ја поедноставува работата со растечки податочни структури

*Овозможува програмите да бидат променети и независно да се компајлираат без повторно линкување и повторно вчитување

*Се нуди за споделување меѓу процесите

*Се нуди за заштита

*Некои системи комбинираат со segmentation со paging