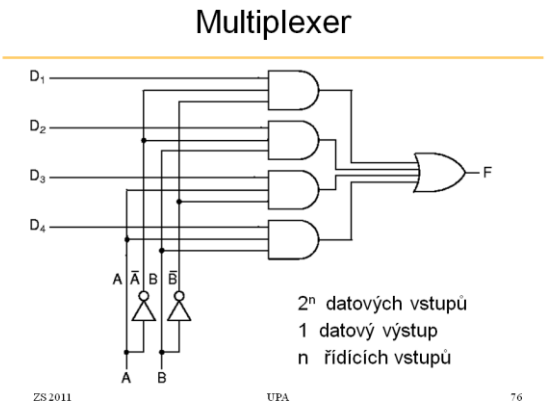
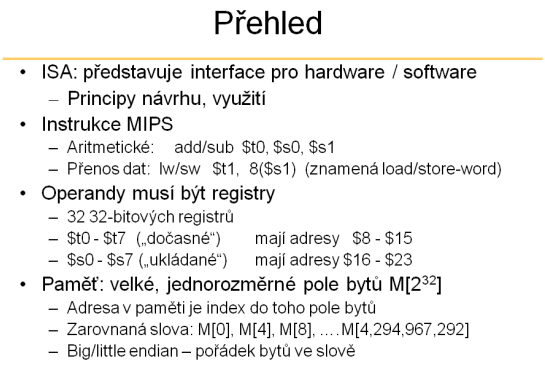
RISC - instrukce jsou prováděny přímo hardwarem
- jednoduché instrukce
- přístup do paměti jen instrukcemi load/store
- velké množství registrů
- pipelining
- počítače s redukovaným souborem intrukcí

Multiplexer



Benchmarky (měření výkonnosti počítačů, zkušební úlohy)
- nejjednodušší měření výkonu se zakládá na měření doby výpočtu
- doba CPU – doba, kterou procesor věnuje zpracování jedné úlohy
  - doba CPU uživatele – čas věnovaný dané úloze
  - doba CPU systému – doba, kdy procesor provádí akce operačního systému, týkající se uživatelské úlohy
- **jediný adekvátní parametr pro měření výkonu je - *čas CPU***
- hodnocení výkonu počítače je založeno na propustnosti a době výpočtu
- vyhodnocení výkonu procesoru je založeno na době výpočtu *výkon = 1 / doba výpočtu*
- výpočet výkonu *CPU time = IC * CPI * doba cyklu*
  - *IC…počet instrukcí*
  - *CPI…střední hodnota počtu cyklů na program*
- *MIPS = počet_instrukcí / doba_výpočtu x 10^6*
- *MFLOPS = počet_FP_opreací / doba_výpočetu x 10^6*
- Amdahlův zákon, Linpack Benchmark, Intelský iCOPM Index, SPEC 2000

ISA (Instruction Set Architecture)
- interface mezi software a hardware
- popis organizačních, funkčních principů procesoru
- z pohledu programátora je to seznam dostupných mechanismů pro programování
- model procesoru může obsahovat:
  - seznam instrukcí
  - datových typů



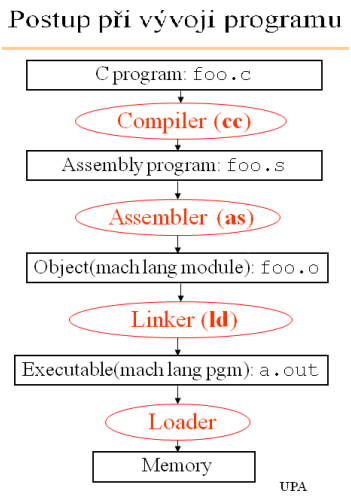
- seznamu registrů

- Tři formáty instrukcí MIPS : R (aritmetické), I (immediate), J (skoky)
- Rozhodovací instrukce – používají *jump* (goto)

PC (Program Counter) – speciální registr, obsahuje adresu právě prováděné instrukce

Změna PC:

- Zvyšuje se při načtení instrukce
- Při skoku a větvení
- Při volání podprogramu
- Při návratu z podprogramu



Pointer (ukazatel) – proměnná, která obsahuje adresu jiné proměnné

- Úsporný kód, úsporný kód
- Zdroj chyb v software

Kompilátor – konvertuje HLL soubor do jednoho souboru

Assembler - čte a používá direktivy

- *Direktivy, neprodukující strojní kód:*
  - .align n //zarovnání na 2^n hranici
  - .text //uložení další položky do textového segmentu
  - .data //uložení další položky do datového segmentu
  - .asciiz str //uložit řetězec *str* do paměti
- nahrazuje makroinstruce → pseudoinstrukce
- generuje stojní jazyk
- vytváří objektový soubor (*.o)

Linker - sestavuje objektové soubory (.o) a vytváří spustitelný soubor – program

- umožňuje nezávislou kompilaci
- edituje „odkazy“ ve skokových instrukcích, vyhodnocuje reference do paměti

Postup:

- Slučuje kódové segmenty všech .o souborů
- Slučuje datové segmenty všech .o souborů a připojuje je na konec kódových segmentů
- Vyhodnocuje reference a prochází relokační tabulku (doplní všude absolutní cesty)

Loader - máme program na disku → loader natáhne program do paměti a spustí

- je součástí operačního systému

Činnost:

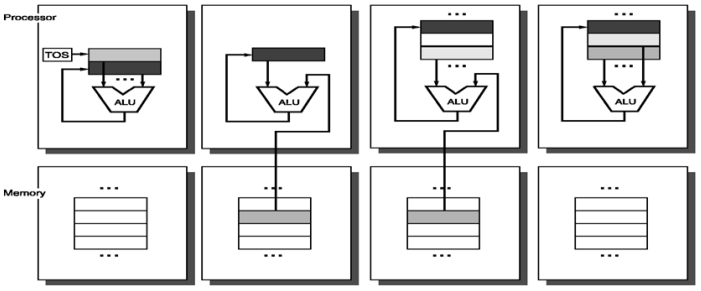
- Čte hlavičku
- Vytvoří adekvátní adresní prostor pro program
- Kopíruje instrukce a data ze souboru do paměti
- Inicializuje registry
- Nastavuje registr PC
- Exit – ukončení programu

Realizace daných výrazů v Stack-machine, Accumulator-machine, Load-Store, Memory-memory

Ilustrace typů architektur (ISA) 1

Assembler pro C:=A+B:

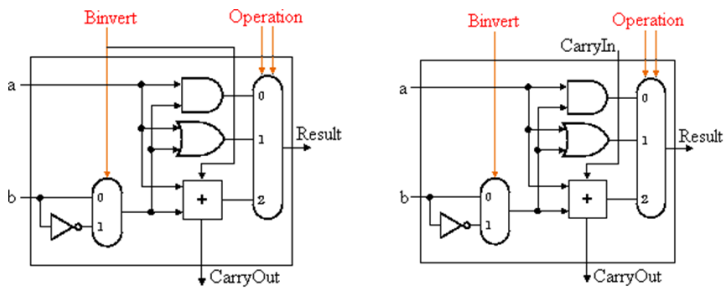
Stack	Accumulator	Register (register-memory)	Register (load-store)
Push A	Load A	Load R1,A	Load R1,A
Push B	Add B	Add R1,B	Load R2,B
Add	Store C	Store C,R1	Add R3,R1,R2
Pop C			Store C,R3



ALU (aritmeticko logická jednotka) - jádro procesoru
- problémem je přetečení (ošetření výjimek – přerušení)

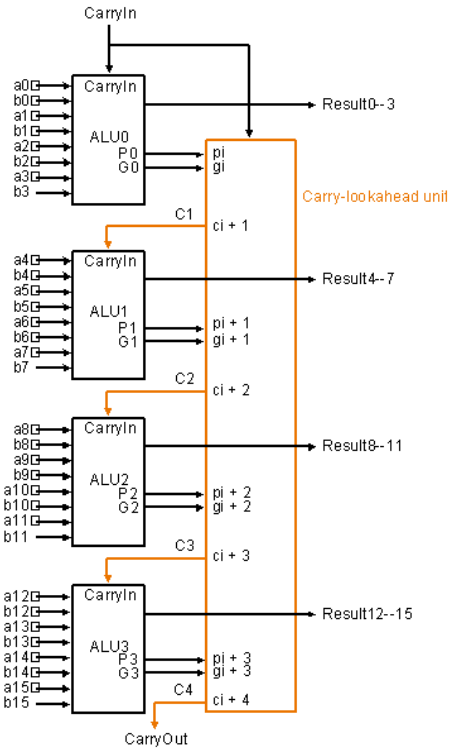
Jednobitová ALU (sčítačka)

Jednobitová ALU



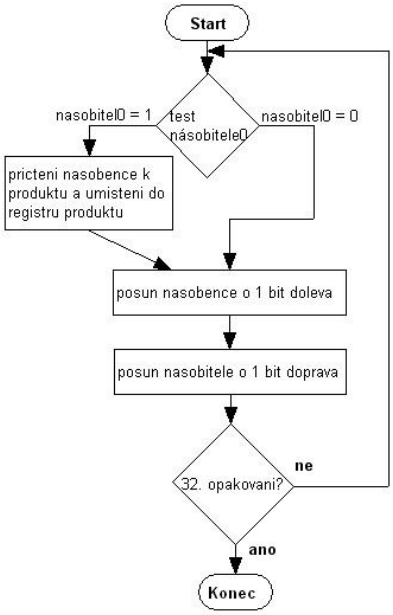
Bit s nejnižší váhou Ostatní bity

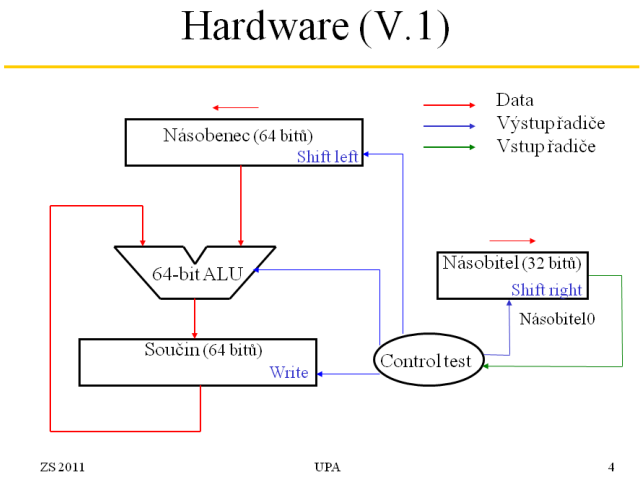
Paralelní binární sčítačka



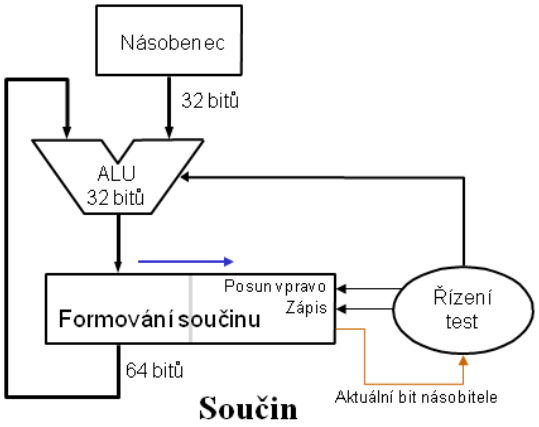
- N-bitovou sčítačku nelze sestavit z 1-bitové (velká a složitá)
- Lze ji sestavit za použití 4-bitových CLA sčítaček
- CLA – Carry-Lookahead Adder
- ***Lépe – použít princip CLA opakovaně***

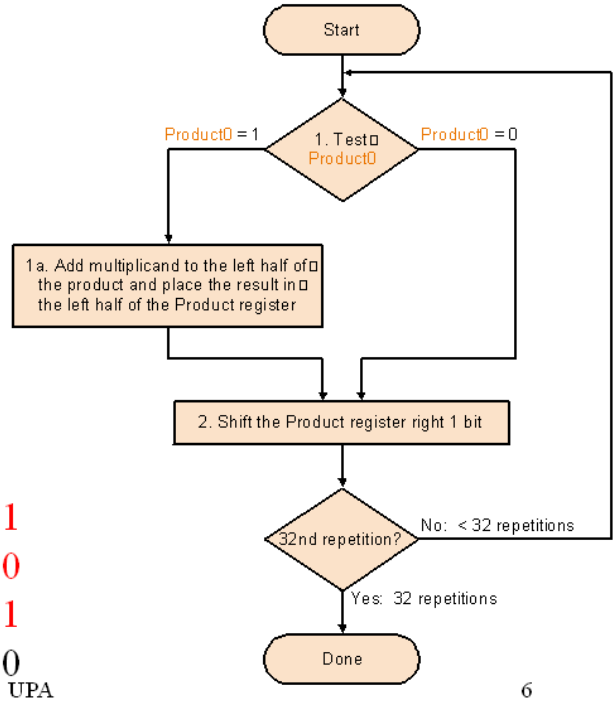
Operace násobení binárních čísel



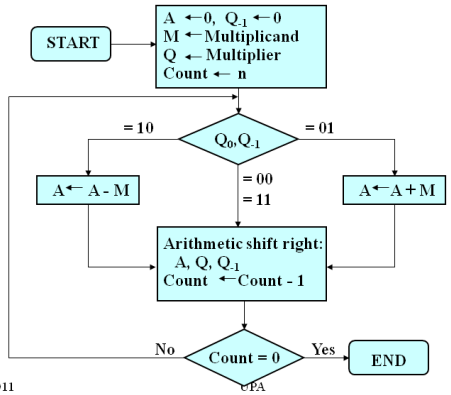


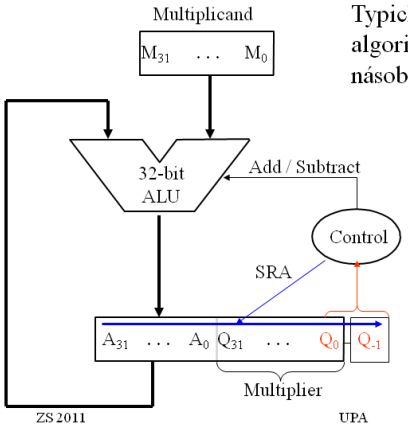
Při násobení dvou 32 bitových čísel vzniká 64 bitový výsledek. Naše varianta řešení využívá pro výpočet 3x64 bitové registry, lze použít 3x32 bitové registry. Násobenec a násobitel jsou uloženy do 32 bitových registrů. Zároveň tvoří poslední volný registr a registr, kde je uložen násobitel celkový výsledek. Pak se vždy testuje 0. bit násobitele a výsledek se přičítá do 3. registru. Po každém cyklu se posune registr násobitele a 3. registr o jeden bit doprava vypadnutý bit, ze 3. registru se doplní do registru násobitele na nejvyšší bitovou pozicí. Celkový výsledek je pak uložen ve dvou 32 bitových registrech.





Boothův algoritmus

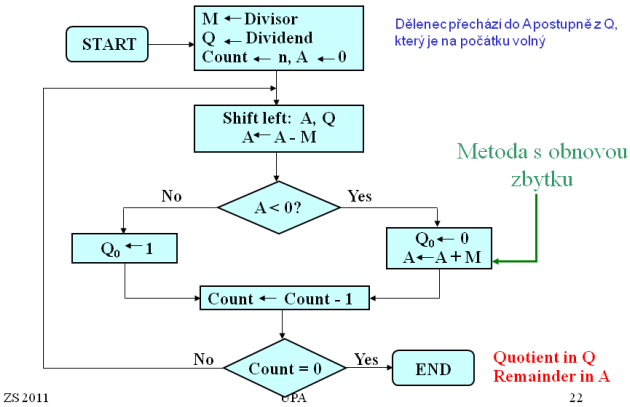


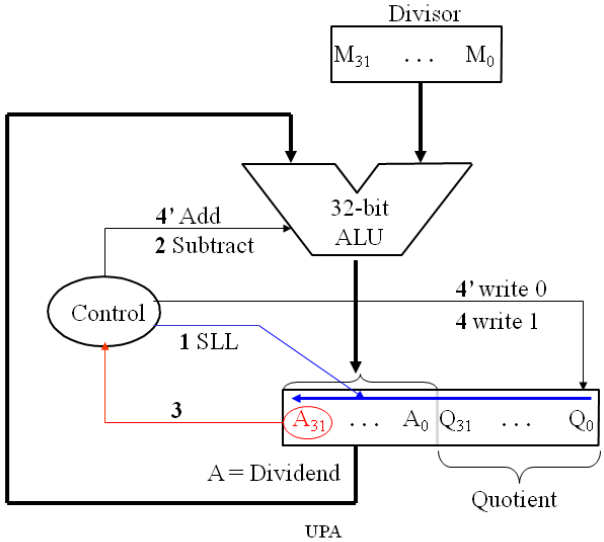


Typickým znakem Boothova algoritmu je test dvou bitů násobitele a postup po jednom bitu

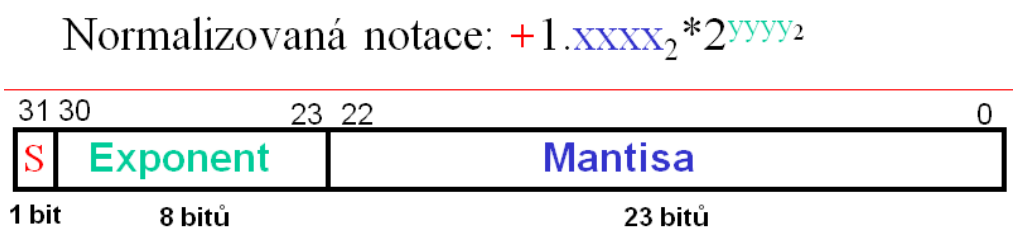
- Odečte, nalezne-li kombinace 1-0
- Přičte, nalezne-li kombinace 0-1

Dělení bez znaménka





Čísla v pohyblivé řádové čárce



- Čísla FP pouze aproximují reálná čísla
- Použití standardu IEEE 754, používá se od roku 1980
- Rozsah 2.0 x $10^{\pm 38}$

Multicyklová jednotka – 1cykl/krok:

- Instrukce je rozdělena na více cyklů, aby se dal aplikovat pipelining, poté ty části, které momentálně nic nedělají, začínají dělat další instrukce

- Implementace pomocí PLA, ROM
- Načtení instrukce
- Dekódování instrukce/čtení dat
- Operace ALU/Provádění instrukcí formátu-R
- Dokončení instrukcí formátu-R
- Dokončení přístupu do paměti

CPI pro R-formát = 4 cykly
CPI pro Store = 4 cykly
CPI pro Load = 5 cyklů
CPI pro Podmíněný skok = 3 cykly
CPI pro Jump = 3 cykly

Jednocyklový procesor

- Všechny instrukce se musí dokončit během jednoho cyklu, tzn. jeho délka, musí vyhovět i nejpomalejší instrukci
- Např. budeme mít dobu cyklu 8ns, potom každá instrukce bude trvat 8ns, i když by tolik času nepotřebovala

Pipelining

- Paralelní zpracování
- Zvyšuje propustnost, efektivitu
- Každá instrukce využívá pouze část hardware v každém kroku, proto lze instrukce ve zpracování překrývat
- Každý proces je rozčleněn na části, každá se provádí ve specializované jednotce
- Moderní procesory podporují multiprocessing i pipelining
- Ideální pipeline → k – 1 + N, kde k je hloubka pipeline, N je počet instrukcí

Pipelining u MIPSu

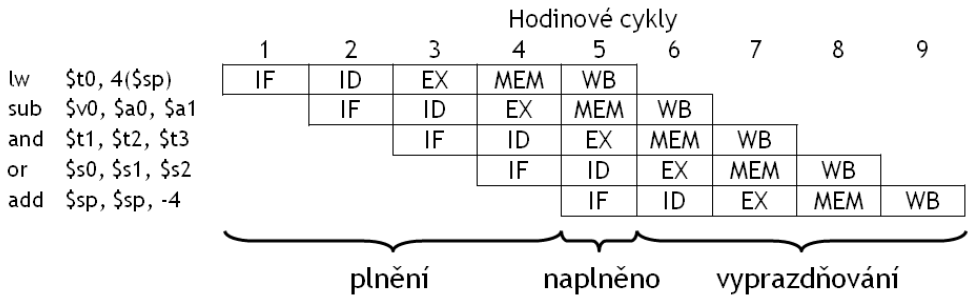
- Provedení instrukce MIPS může zabrat 5 kroků

Krok	Jméno	Popis
Instruction Fetch	IF	Čtení instrukce z paměti
Instruction Decode	ID	Čtení zdrojových registrů a generace řídících signálů
Execute	EX	Výpočet výsledku R-typu popř. větvení
Memory	MEM	Čtení nebo zápis do datové paměti
Writeback	WB	Uložení výsledku do cílového registru

- Všechny instrukce nepotřebují všech 5 stupňů…

Instrukce	Vyžadované kroky				
beq	IF	ID	EX		
R-type	IF	ID	EX		WB
sw	IF	ID	EX	MEM	
lw	IF	ID	EX	MEM	WB

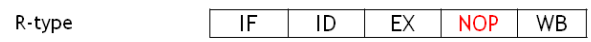
…jednocyklová verze ale musí zvládnout všech 5 stupňů během *jednoho* taktu

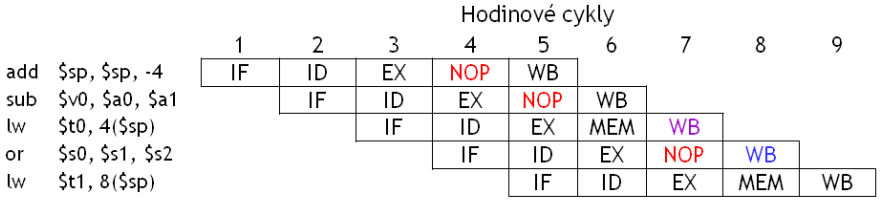


- Hloubka pipeline je počet stupňů, zde pět
- V prvních čtyřech probíhá plnění, jsou zde nevyužité funkční jednotky
- V cyklu 5 je pipeline plná. Probíhá zpracování pěti instrukcí současně, jsou využity všechny hardwarové jednotky
- V cyklech 6-9 se pipeline vyprazdňuje

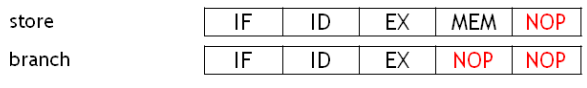
Řešení: Vložení NOP

- Vynucení uniformity
  - —Navrhnout tak, aby všechny instrukce trvaly 5 cyklů.
  - —Navrhnout se stejným počtem stupňů, ve stejném pořadí
    - některé stupně budou neaktivní pro některé instrukce





- Zápisy a větvení mají NOPy …



Hazardy

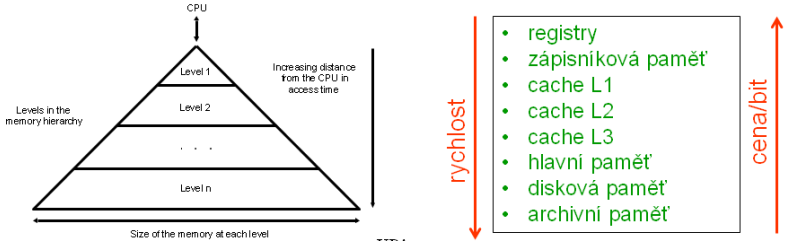
- Brání provedení následující instrukce v příslušném taktu
  - ***Datové hazardy***
    - Instrukce je závislá na výsledku předchozí instrukce
    - Pozastavení – vložení třech bublin (no-ops) do pipeline
    - <u>Řešení:</u> ***forwarding*** – zasílání dat také do dalšího stupně
      - změna pořadí instrukcí, aby se omezilo pozastavování
    - ***RAW (Read after Write)*** – instrukce chce číst novou hodnotu dříve, než ji instrukce může zapsat
    - ***WAR (Write after Read)*** – instrukce chce zapsat výsledek dříve, jež druhá instrukce přečte hodnotu
    - ***WAW (Write after Write)*** – instrukce chce zapsat výsledek dříve než druhá instrukce
  - ***Strukturní hazardy***
    - různé instrukce se snaží používat současně stejné funkční jednotky (paměť, registrovou sadu)
    - <u>Řešení:</u> duplikovat hardware
  - ***Řídící hazardy (větvení)***
    - Cílová adresa je známá až na konci třetího cyklu → POZASTAVENÍ
    - <u>Řešení:</u> predikace (statická a dynamická): smyšky
      - opoždění instrukcí větvení (branches)

Branch delay slot

- Provedeme-li instrukci větvení, žádná instrukce za skokem není provedena náhodně
- Ať se koná větvení nebo ne, instrukce ležící bezprostředně za skokem, se provede vždy (nyzývá se ***branch delay slot***)

Technologie pamětí

- V současné době se ke stavbě hierarchického paměťového systému používají hlavně tři typy pamětí…
  - Hlavní paměť používá technologii DRAM (dynamická RAM).
  - Úrovně blíže k CPU (L1/L2/L3 cache) jsou statické RAM (SRAM). Jsou mnohem dražší, ale také mnohem rychlejší.
  - Největší kapacitu, ale také nejmenší rychlost mají paměti, používané na nejnižší úrovni. (např. harddisky). Tato technologie je velmi levná, ale doba přístupu je dlouhá.



Fyzická paměť – instalována v počítači

Virtuální paměť – disková paměť, chová se jako hlavní paměť

Cache paměť – rychlá paměť pro dočasné umístění dat nebo instrukcí

Statická RAM (SRAM)

- Informace je uchovávána pomocí dvojce invertujících hradel
- rychlé, dražší
- nejsou potřeba zotavovací obvody
- cache

Dynamická RAM (DRAM)

- informace je uchována jako náboj na kondenzátoru
- pomalejší a levnější než SRAM
- potřeba zotavovací obvody
- adresní linky – zápis, čtení

Obě ztrácí informaci po vypnutí napájení.

Hierarchie paměťového systému

- registry – rychlá interní paměť v procesoru, rychlost = 1 cykl hodin CPU
- cache – rychlá paměť interní nebo externí (k procesoru), rychlost = několik cyklů hodin CPU
- hlavní paměť – externí vzhledem k procesoru, rychlost 1-10 hodin CPU
- disková paměť – velmi pomalá – doba přístupu = 1 -15 ms, použití pro odkládání dat (instrukcí) z hlavní paměti

Lokalita

- Tento příklad znázorňuje ***princip lokality*** – programy přistupují v daném krátkém časovém intervalu k relativně malé části adresního prostoru (stejně jako vy jste přinesli jen malou část knih).
- Existují dva typy lokality …
  - *Časová lokalita* – je-li položka referencována, má tendenci být referencována v krátkém časovém úseku opět. (knihu, kterou jste právě odložili na stůl pravděpodobně zase brzy vezmete do ruky).
  - *Prostorová lokalita* – Je-li položka referencována, budou patrně referencovány i s ní sousedící položky (naleznete-li knihu, je pravděpodobné, že vás mohou zajímat i knihy, ležící v jejím těsném okolí).

Cache paměť - pojmy

- **Blok:** Skupina slov **Set:** Skupina bloků
- Hit ☺
  - *Je-li hledán blok B v cache a je nalezen*
  - Hit Time: čas potřebný k nalezení bloku B
  - Hit Rate: četnost přístupů, kdy B je nalezen v cache
- Miss ☹
  - *Je-li hledán blok B v cache a není nalezen*
  - Miss Rate: četnost přístupů, kdy B je hledán v cache a není nalezen

Strategie zápisu bloků

1) Write-Through:

- Zápis dat (a) do cache *a současně* (b) do bloku v hlavní paměti
- **Výhoda:** Případ „Miss" je jednodušší & levnější, protože není třeba zapisovat blok zpět do nižší úrovně
- **Výhoda:** Snazší implementace, je třeba pouze zápisový buffer

2) Write-Back:

- Zápis dat *pouze* do bloku cache. Zápis do paměti pouze při výměně bloků
- **Výhoda :** Zápisy omezeny pouze rychlostí zápisu cache
- **Výhoda :** Podporovány zápisy více slov najednou, efektivní je pouze zápis celého bloku do hlavní paměti najednou