APO

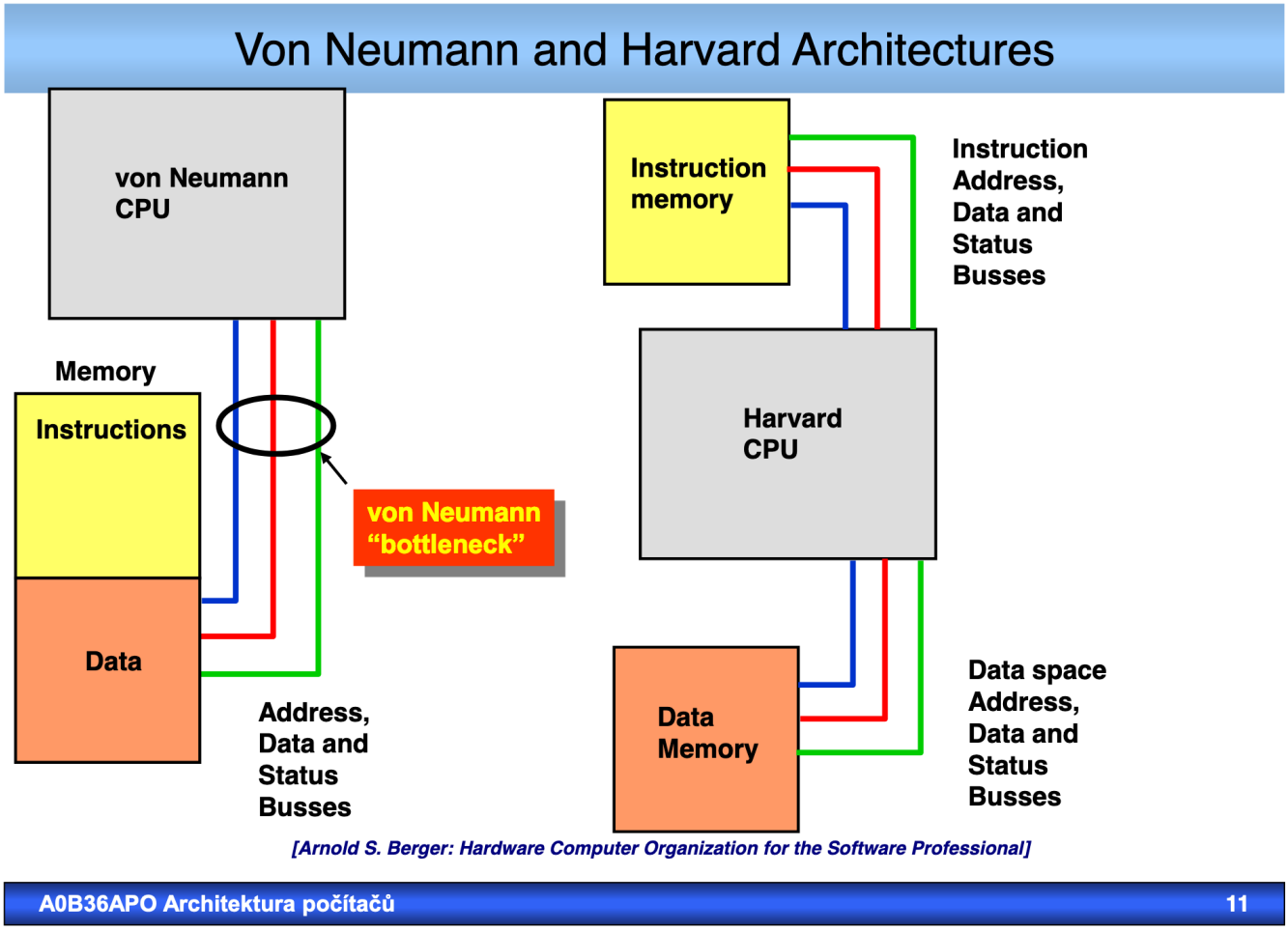
7. Architektura počítače; CPU, paměti, subsystémy.

---

https://cw.fel.cvut.cz/wiki/courses/b35apo/start

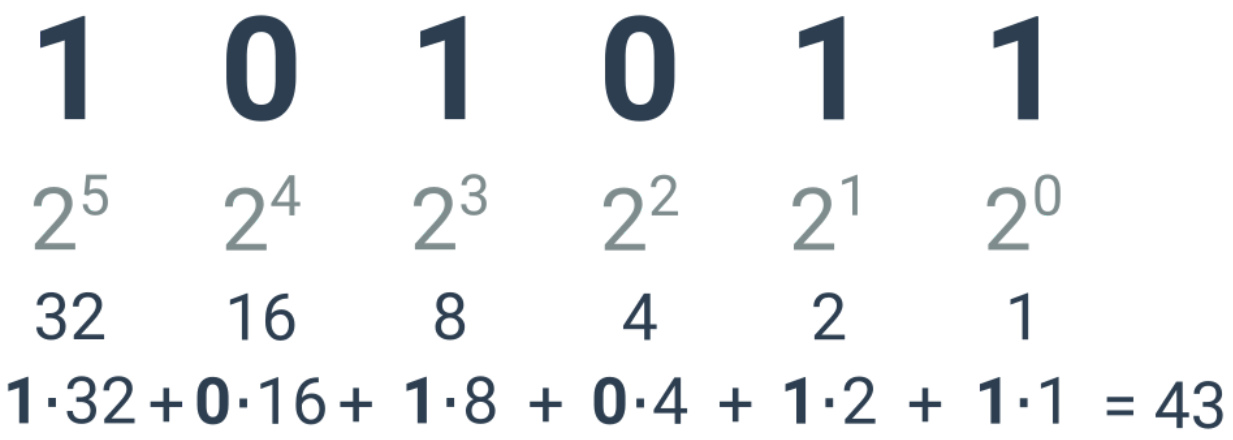
Architektura počítače

Von Neumanova - jednodušší, pomalejší, možnost rozdělit paměť, používá se dnes
data/instrukce dle potřeby **Harvardská** - paměť pro instrukce a data fyzicky oddělena Sběrnice propojující paměti se skládá ze tří sběrnic: adresní, datové, řídící



Reprezentace čísla v počítači

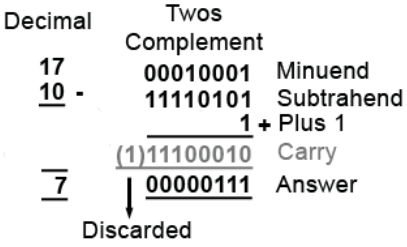
Dvojková soustava -

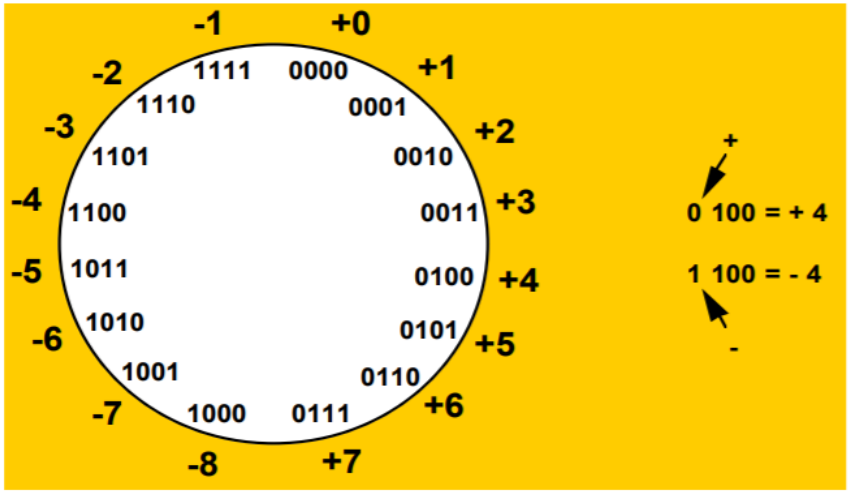


bit - 1,0
nible - 4 bity, hexa, 0-15
byte - 8 bitů, 0-255

Two’s complement (dvojkový doplněk)

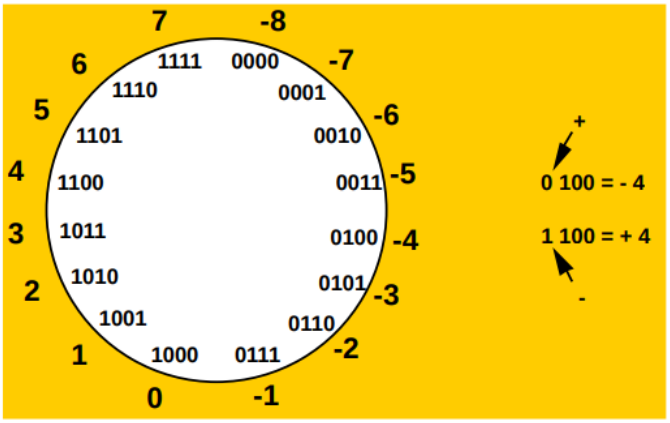
Pro byte je rozsah -128 až 127





Excess-K (aditivní)

Exponent reálných čísel, K = $2^{a-1} - 1$, kde "a" je počet bitů



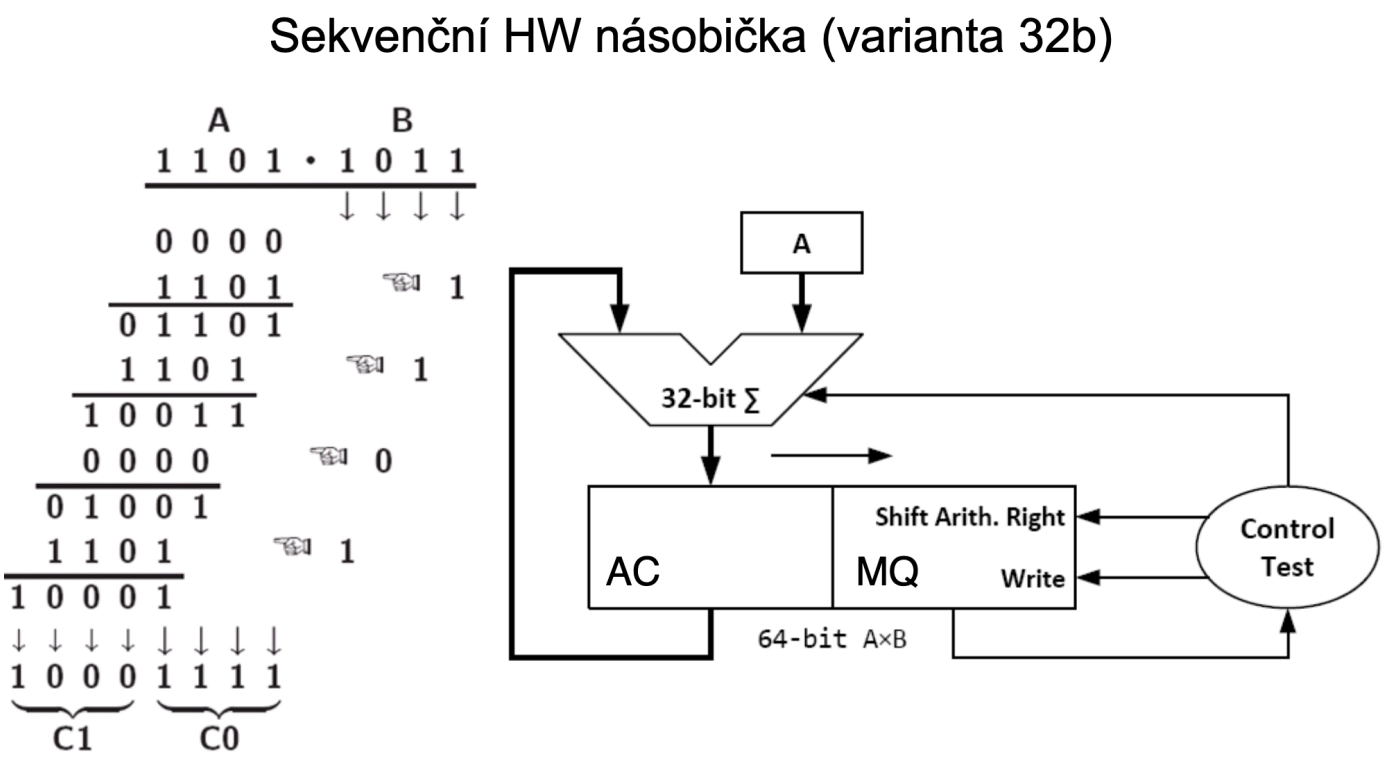
Endianita

- Little endian: V tomto případě se na paměťové místo s nejnižší adresou uloží nejméně významný byte, intel
  32bitové číslo `0x4A3B2C1D` se na adresu `100` uloží takto:

	100	**101**	**102**	**103**	
...	1D	2C	3B	4A	...

- Big endian: V tomto případě se na paměťové místo s nejnižší adresou uloží nejvíce významný byte, motorola
  32bitové číslo `0x4A3B2C1D` se na adresu `100` uloží takto:

	100	**101**	**102**	**103**	
...	4A	3B	2C	1D	...

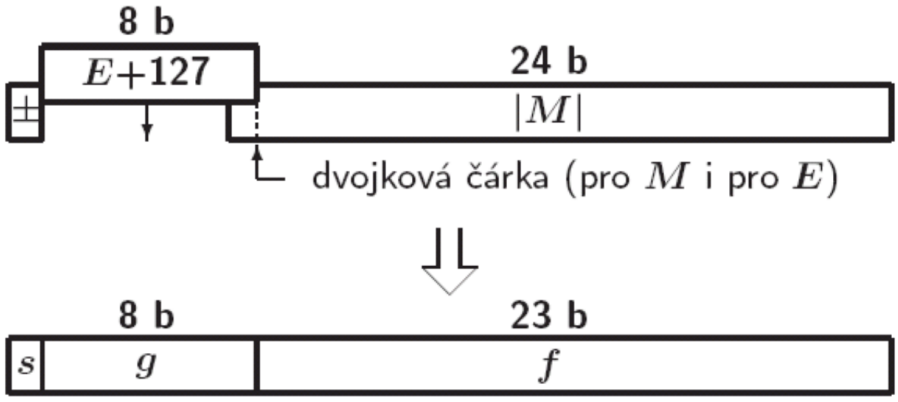


Reálná čísla

M - mantisa (zlomková část), v přímém kódu
E - exponent, v aditivním kódu
S - znaménko +- (1=záporné)
Normalizované - skrytá 1, například číslo $1.00101001_2 e^{100_2}$ se uloží bez první 1
Nenormalizované - čísla, co jedničku neobsahují, exponent samé 0 nebo 1

- |0|11111111|0...0| - nekonečno (nenormalizované), lze i mínus inf
- |0|11111111|ne samé 0| - NaN not a number, nedefinované číslo
- |0|00000000|0...0| - nula (nenormalizované), lze i mínus 0

ANSI/IEEE Std 754-1985 — jednoduchý formát — 32b



ANSI/IEEE Std 754-1985 — dvojnásobný formát — 64b

g . . . 11b *f* . . . 52b

Převod čísla na binární reprezentaci

- Převeďte -12.625$_{10}$ IEEE-754 float formát.
- Krok #1: Převeďte -12.625$_{10}$ = -1100.101$_2$ = 101 / 8
- Krok #2: Normalizujte -1100.101$_2$ = -1.100101$_2$ * 2^3
- Krok #3:

Vyplňte pole, znaménko je záporné -> S=1.

Exponent + 127 -> 130 -> 1000 0010 .

Úvodní bit 1 mantisy je skrytý ->

1 1000 0010 . 1001 0100 0000 0000 0000 000

- Převeďte -12.625_{10} IEEE-754 float formát.
- Krok #1: Převeďte $-12.625_{10} = -1100.101_2 = 101 / 8$
- Krok #2: Normalizujte $-1100.101_2 = -1.100101_2 * 2^3$
- Krok #3:

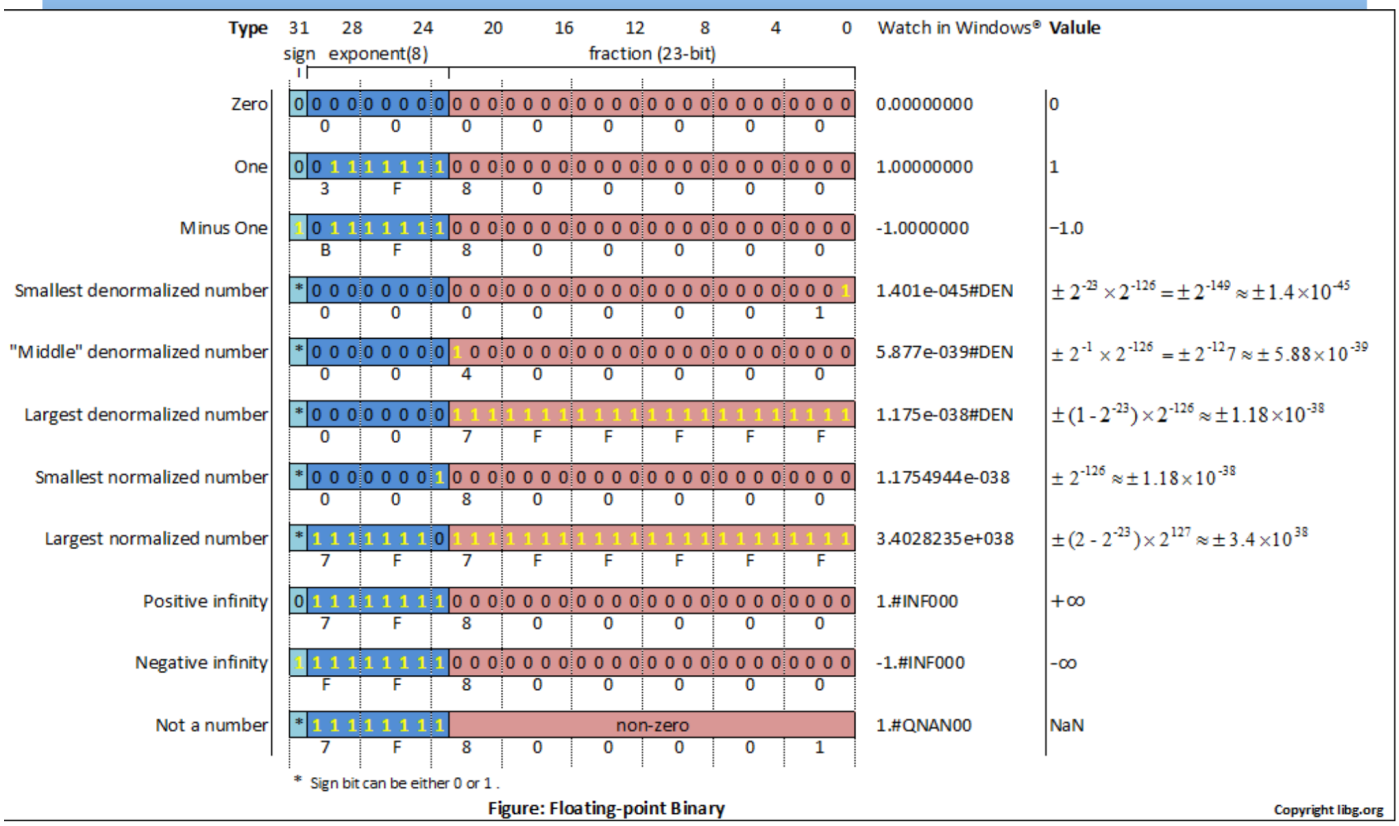
Vyplňte pole, znaménko je záporné -> S=1.

Exponent + 127 -> 130 -> 1000 0010 .

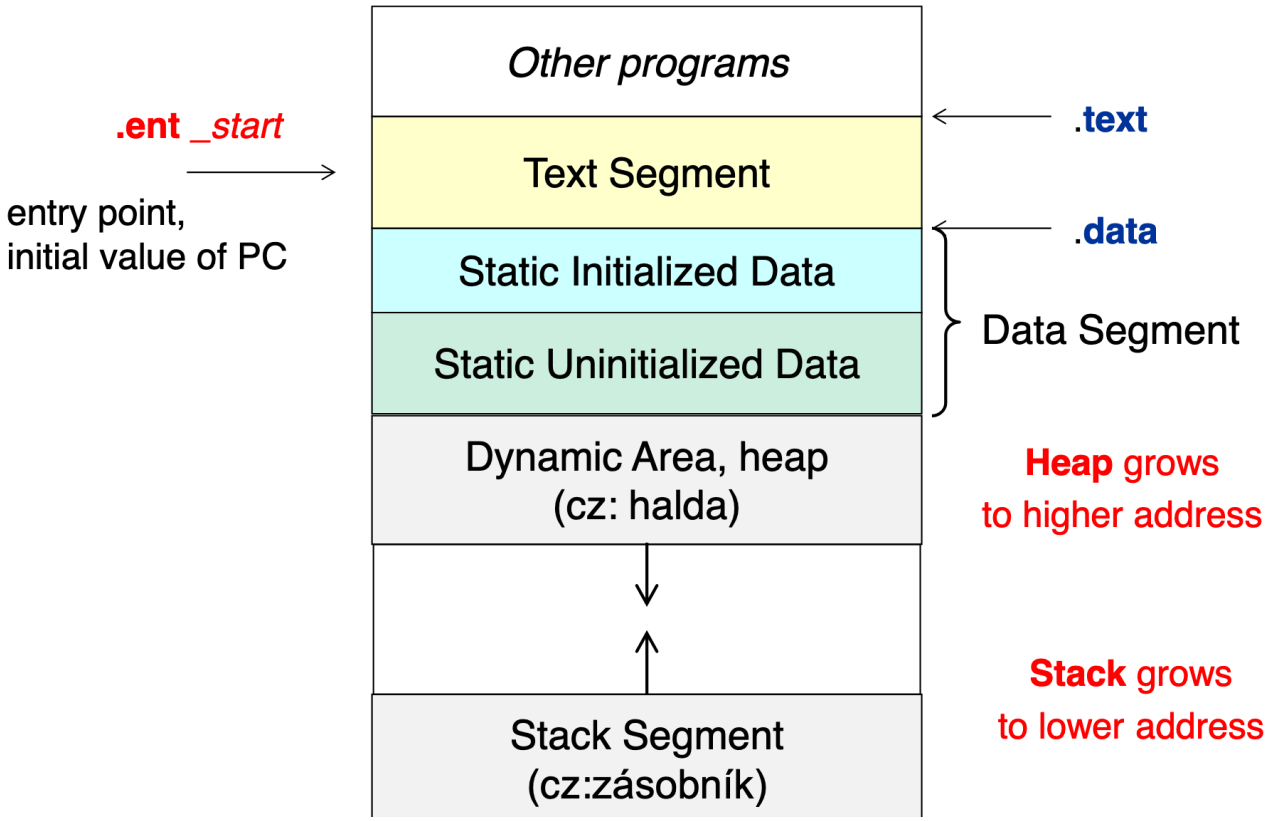
Úvodní bit 1 mantisy je skrytý ->

1 1000 0010 . 1001 0100 0000 0000 0000 000

Širší přehled



Layout of program in memory



Architektura CPU

---

https://cw.fel.cvut.cz/wiki/_media/courses/b35apo/lectures/04-n-memory.pdf

Struktura CPU

řídící jednotka (řadič) zajišťuje součinnost jednotlivých částí CPU
Registry - uchovávají data pro další výpočty nebo důležitá data jako PC nebo i konstanty jako 0, velmi rychlá paměť

- *datové* - ukládání hodnot
- *adresní* - uchovávají adresy odkud mají být data načítána / kam ukládána
- procesor může vykonávat aritmetické/logické operace pouze nad daty v registrech
  ALU (aritmeticko-logická jednotka) (může jich být i více) zajišťuje všechny aritmetické a logické výpočty
  PC (program counter) - čítač instrukcí - uchovává stav paměti, ukazuje na instrukci, co se má vykonávat, přičítá se po délce instrukce

Propojen s RAM pomocí sběrnic (adresní část - určuje adresu paměti, datová část - přenáší data, řídící část - read/write a kdy)

Ukazatel zásobníku - stack pointer - SP

- uchovává adresu posledního záznamu uloženého na zásobníku
- obvykle “růst dolů” na nižší adresy: při push se *SP* snižuje, při pop se *SP* zvětší
- zásobník pracuje na principu LIFO (last in, first out)
- používá se pro uchování návratové adresy při volání funkcí, nebo zároveň pro ukládání parametrů pro volanou funkci
- sekundární využití je pro ukládání proměnných programu, ke kterým nepřistupujeme instrukcemi PUSH a POP, ale běžnými ukazateli

Typy instrukcí
CISC complex instruction set computer

- instrukce se skládá z několika kroků ⟶ zpomalování procesoru
- různě dlouhé, různě trvající instrukce
- Intel/AMD

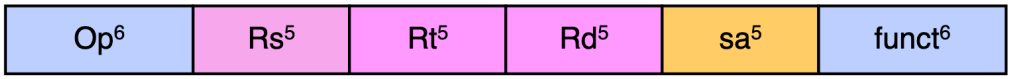
RISC reduced instruction set computing

- 32 bit vždy -> jednoduše se inkrementuje PC
- snížení počtu instrukcí, snaha dosáhnout 1 takt=1 instrukce
- MIPS/ARM
- Instruction format - R (aritmetické), J (skokové na dané adresy), I (podmínkové skoky, použití immediate hodnoty (konstanty))
  - rs - register source
  - rt - register target
  - rd - register destination

All instructions are 32-bit wide.

Register (R-Type)

Register-to-register instructions, Rx are numbers of registers
Op: operation code specifies the format of the instruction,
funct- sub-function, control codes
sa - used with the shift and rotate instructions,



Immediate (I-Type)

16-bit immediate constant is a part of the instruction

Jump (J-Type)

Used by jump instructions only

Upper indexes specify a bit width of fields .

Které z následujících tvrzení je pravdivé?

- a) Pro architekturu CISC je typické větší množství identických univerzálních registrů
- b) Architektura RISC podporuje i složité adresovací módy
- c) Výsledný kód je delší pro architekturu RISC
- d) Jedna instrukce architektury CISC se typicky vykonává v jediném hodinovém taktu

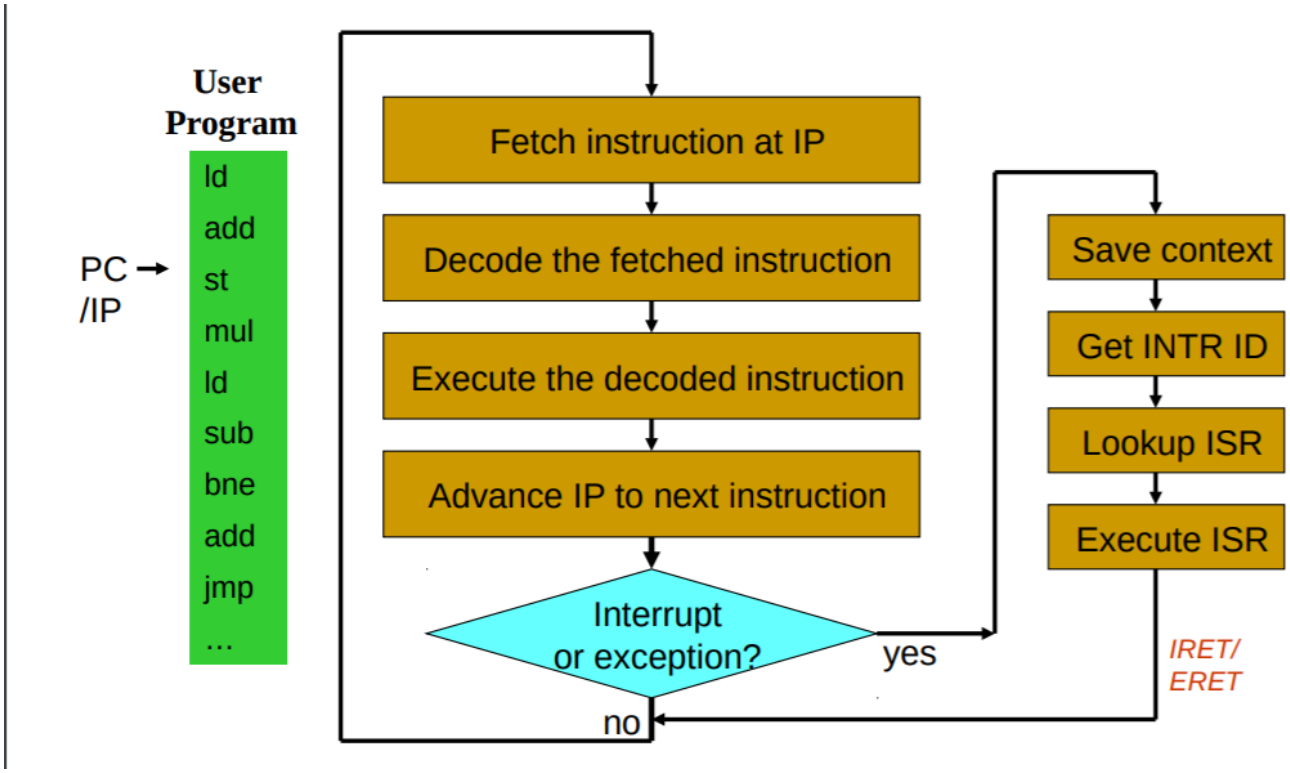
> ! podle tohoto https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc/ c)

Jedno cyklové CPU

Nejdříve se vykoná jedna instrukce a když jsou všechny její části (IF, ID, EX ...) hotové, tak se načte další. Tento přístup je pomalý. Většinou se alespoň separátně načítají a vykonávají instrukce (zde je použit delay slot - po každém skoku je načtena jiná instrukce, většinou to je ale NOP aneb žádná instrukce).

- počáteční nastavení (zejména PC)
- čtení instrukce
  - PC → adresa Hlavní paměti (HP)
  - čtení obsahu
  - přečtená data se uloží do IR (Instruction register)
  - inkrementace PC o délku instrukce
- přečtení opcode (kód instrukce - typicky několik prvních bitů podle velikosti instrukční sady)
- provedení instrukce - zahrnuje i například čtení operandu
- pokud došlo k přerušení (např. IO), nebo výjimce (invalid instruction, dělení 0, chyba zápisu/čtení) tak se zpracuje

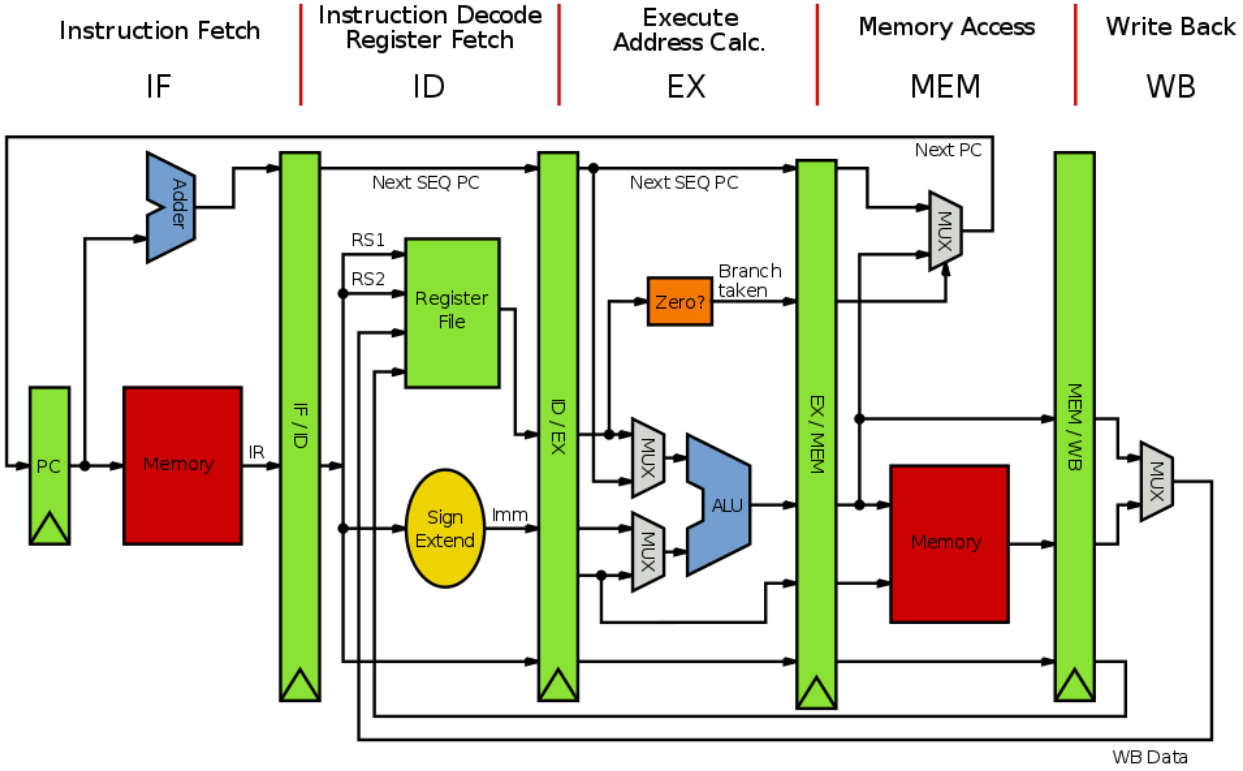
- opakuj od 2. bodu



Pipelining

Pipelining, zřetězené zpracování či překrývání strojových instrukcí je způsob zvýšení výkonu procesoru současným prováděním různých částí několika strojových instrukcí. Základní myšlenkou je rozdělení zpracování jedné instrukce mezi různé části procesoru a tím i umožnění zpracovávat více instrukcí najednou. Fáze zpracování bývá rozdělena na následují úseky:

1. IF - nastavení PC a načtení instrukce do paměti, PC += instr. size
2. ID - dekóduje se opcode instrukce a přečtou se data z registrů specifikované instrukcí, zde se vykonávají podmínky
3. EX - vykonání instrukce, poslání hodnot registrů do ALU
4. MEM/ME - read/write do hlavní paměti
5. WB - zápis zpět do registru



Mělo by pak platit (až na výjimky na začátku atd.), že jeden takt procesoru = jedna dokončená instrukce.

Problém s ukládání do registrů:

- Pokud budeme vždy načítat instrukce dopředu, tak u uložení do registru (WB) může být problém, že následující instrukce použije starou hodnotu registru.
- **Hazard unit** - Pokud nastane problém popsaný výše, tak provede hazard unit STALL - vloží mezi problémové instrukce, jednu instrukci NOP nebo zablokuje všechny brány vlevo od EX. Toto pozastaví na jeden takt procesor a nebudou probíhat nové instrukce, během čehož se dodělá práce předešlé instrukce. STALL je řešení, ale zpomaluje procesor -> řešení FORWARDING
- **Forwarding** (v tomto případě)- Přeposlání potřebné hodnoty z WB nebo už z ME rovnou do EX, kde se nepoužijí hodnoty získané z ID. S tímto je ale stále potřeba dát STALL nebo NOP instrukci před instrukci lw (load word z paměti v ME fázi), pokud hned po ní používáme její hodnotu.

Problém podmíněnými skoky

- Pokud budeme vždy načítat instrukce dopředu u skoku ve vyhodnocování podmínky zda skočit (if statement, probíhá v ID) se může použít stará hodnota registru. Pokud probíhá výpočet hodnoty v podmínce v předešlé instrukci je potřeba použít STALL, jinak vše zařídí forwarding (ME -> ID).
- Další problém je, že pokud se v podmínce bude skákat, tak všechna načtená data a instrukce se musí zahodit. Řešení jsou predikce skoků (další kapitola).

Superskalární architektura
Superskalární architektura (superskalarita) je v informatice jedním ze způsobů zvyšování výkonu procesoru. Superskalarita umožňuje v jednom taktu zpracovat více strojových instrukcí zároveň, protože některé části procesoru jsou duplikovány, například matematický koprocesor (FPU) nebo aritmeticko-logická jednotka (ALU). Od více jádrových procesorů se superskalarita liší tím, že je zvětšen počet pouze některých částí procesoru.

Čeho se vlastně dosáhne aplikací superskalární architektury nebo pipeliningem?

- Jedná se o dvě různá označení jedné techniky zvýšení výkonu CPU
- Jde o dvě různé techniky zvýšení výkonu CPU
- Jde o dvě různé techniky ke snížení příkonu CPU
- S CPU tyto pojmy nesouvisí

> ! Jde o dvě různé techniky zvýšení výkonu CPU

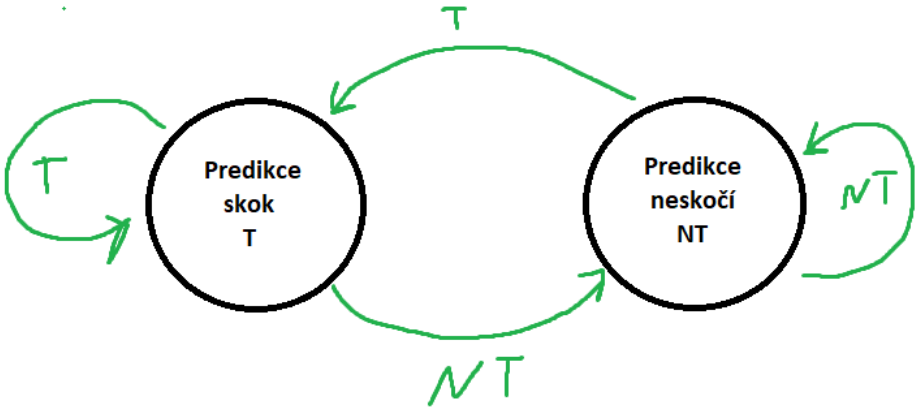
Predikce skoků

Spekulativní vykonávání instrukcí je metoda u podmínek (if statement), kde si na základě nějakého kritéria (prediktory), vybereme zda se začne vykonávat následující instrukce jako kdyby podmínka byla pravdivá, či nikoliv.

Nejednoduší možnost je mít pevnou predikci, vždycky předpokládat, že se skočí (taken, T) nebo neskočí (not taken, NT), to má však úspěšnost jen kolem 67% (if), 99.99% (for, while).

Jednobitový prediktor

- Postupné přecházení mezi dvěma stavy.



Dvoubitový prediktor

- Postupné přecházení mezi čtyřmi stavy. Lepší než jednobitový.
- ↻Strongly taken ⇄ Weakly taken ⇄ Weakly not taken ⇄ Strongly not taken↺
- Modifikace hysterezí - ze stavu weakly T/NT špatnou predikcí rovnou na strongly NT/T.

Perceptron

- Použit v moderních CPU, model neuronu, vstup je BHR (branch history register) neboli registr s historií několika "n" posledních skoků, učí se vzor chování programu, pro každý ze vstupů (jednotlivé poslední skoky) si pamatuje a upravuje jeho váhu
- Vylepšení -> 2 perceptrony mezi kterými se vybírá podle úspěšnosti například 2 bitových prediktorem

To jaký prediktor použít pro danou instrukci ze zjistí podle adresy PC. Vezme se spodních "n" bitů adresy, pokud máme 2^n prediktorů.

Architektura paměti

Struktury a hierarchie pamětí

- Rychlé a malé paměti jsou umístěny blízko CPU.
- Větší a pomalejší paměti jsou dále od CPU.
- Čím jsou paměti blíže tím se jejich poměr cena/velikost a rychlost zvětšuje
- Princip cachování nejčastěji používaných dat v rychlých pamětech - zrychlení přístupu k datům.
- od nejrychlejší a nejmenší po největší a nejpomalejší:
  - on-chip L1 caches zpoždění kolem 4 cyklů CPU
  - off-chip L2 caches (SRAM) zpoždění kolem 12 cyklů CPU
  - L3 společná pro více vláken (logických CPU) zpoždění kolem 36 cyklů CPU
  - hlavní paměť (DRAM) zpoždění kolem 300 cyklů CPU
  - vedlejší paměť (pevný disk)

SRAM vs DRAM

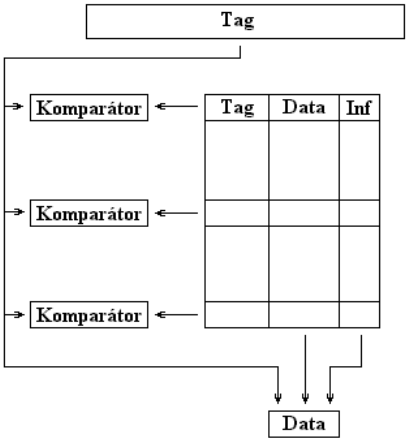
- **SRAM**: Statické paměti uchovávají informaci v sobě uloženou po celou dobu, kdy jsou připojeny ke zdroji elektrického napájení. Paměťová buňka SRAM je realizována jako **bistabilní klopný obvod**, tj. obvod, který se může nacházet vždy v jednom ze dvou stavů, které určují, zda v paměti je uložena 1 nebo 0.
- **DRAM**: V paměti DRAM je informace uložena pomocí **elektrického náboje na kondenzátoru**. Tento náboj má však tendenci se vybíjet i v době, kdy je paměť připojena ke zdroji elektrického napájení. Aby nedošlo k tomuto vybití a tím i ke ztrátě uložené informace, je nutné periodicky provádět tzv. **refresh**, tj. oživování paměťové buňky. Tuto funkci plní některý z obvodů čipové sady.

Cache paměti (chtějí to prý nakreslit)

Je to rychlejší ale menší paměť.
Je rozdělená na bloky dat (16, 32, 64 bytů). Každý blok má TAG, ten udávají nejvyšší bity adresy v RAM. Také má příznakový bit, kde je 1 (ok) nebo 0 (prázdná paměť). Tento celek je základní stavební jednotka cache.

Pokud je plná, tak se data na vyhození mohou vybrat podle různých kritérií. Například: nejstarší data, nejméně používaná data, nejdéle nepoužívaná data, náhodně

Plně asociativní cache

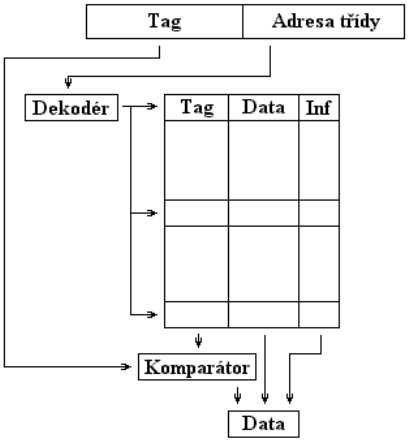


U plně asociativní cache paměti je celá adresa, ze které se budou číst data (popř. na kterou se budou data zapisovat), brána jako tag. Tento tag je přiveden na vstup komparátorů (zařízení realizující porovnání dvou hodnot) společně s tagem v daném řádku tabulky. Pokud některý z tagů v tabulce je shodný se zadaným tagem na vstupu, ohlásí odpovídající komparátor shodu (HIT) a znamená to, že požadovaná informace je v cache paměti přítomna a je možné ji použít. Pokud všechny komparátory signalizují neshodu (MISS), je to známka toho, že požadovaná informace v cache paměti není a je nutné ji zavést odjinud (externí cache paměť, operační paměť).

Výhody : ze všech typů nejrychlejší

Nevýhody: velké množství komparátorů, HW náročná, mnoho obvodů, plýtvání místem

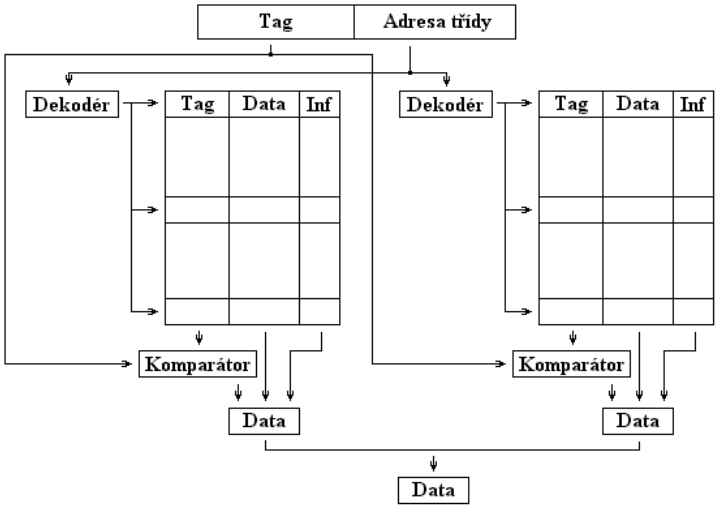
Přímo mapovaná (jedno cestná) cache



Přímo mapovaná cache paměť je speciální případ n-cestně asociativní cache paměti pro n=1. Zadaná adresa je rozdělena na tag a adresu třídy (množinu). Adresa třídy je přivedena na vstup dekodéru, kde se vybere jeden řádek v tabulce. Tag na tomto řádku je následně porovnán se zadaným tagem, čímž se rozhodne o přítomnosti resp. nepřítomnosti informace v cache paměti.

Přímo mapovaná cache ve srovnání s n-cestně asociativní cache pamětí vykazuje nižší výkon (velké množství přepisů), a proto její použití není dnes příliš časté.

N-cestně asociativní



Adresa třídy je přivedena na n dekodérů (zařízení, které na základě vstupní hodnoty vybere jeden ze svých výstupů, na který umístí hodnotu log. 1, a na ostatní výstupy umístí hodnotu log. 0), které v každé tabulce vyberou jeden řádek, neboli množinu. Z těchto řádků se potom vezmou příslušné tagy a komparátorem se porovnají se zadaným tagem. Podobně jako u plně asociativních cache pamětí pokud jeden z komparátorů signalizuje shodu, je informace v cache paměti přítomna. V opačném případě je nezbytné informaci hledat jinde.

N-cestně asociativní paměti částečně eliminují nevýhody (mnoho komparátorů) plně asociativních cache pamětí a v současnosti jsou nejpoužívanějším typem cache pamětí.

Počet cest * velikost data bloku * počet množin = velikost cache

hledám data když je hit L1 tak OK když MISS tak:
- L2, zde když je hit tak OK a kopíruju data do L1, když MISS tak:
- L3, zde když je hit tak OK a kopíruju data do L1 a L2 + synchronizace jader, když MISS tak:
- atd.

Podle způsobu práce při zapisování dat lze cache paměti ještě rozdělit do dvou skupin:

- **write-through**: cache paměti, u kterých v případě zápisu procesoru do cache paměti dochází okamžitě i k zápisu do vyšších úrovní. Procesor tak obsluhuje jen zápis a o další osud dat se stará cache paměť.
- **write-back**: cache paměti, u nichž jsou data zapisována do operační paměti až ve chvíli, kdy je to třeba, a nikoliv okamžitě při jejich změně. K zápisu dat do operační paměti tedy dochází např. v okamžiku, kdy je cache zcela zaplněna a je třeba do ní umístit nová data. Tento způsob práce cache paměti vykazuje oproti předešlému způsobu vyšší výkon, ale vzniká problém konzistence dat.

Způsoby adresace/Stránkování

Používá se k tomu a každý proces měl svoji paměť a neovlivňovaly se navzájem. Paměť procesu je rozdělen na jednotlivé stránky. Ty jsou pak vloženy do RAM. Každý proces bude mít PT (page table), kde pro každou stránku je adresa v RAM.

Máme 2 typy adres. Logickou a fyzickou adresu. Logická adresa je adresa, se kterou pracuje proces. Tato adresa je překládána na fyzickou adresu, což je konkrétní adresa v RAM paměti. Přes logickou adresu se dá přistupovat k datům sekvenčně, tak jak jdou za sebou.

Výhody - CPU vidí unifikované adresy - pro přístup do hlavní paměti i k IO zařízením. Data v hlavní paměti nejsou poskládána tak jak jdou za sebou, mohou být zpřeházena - optimalizace využití prostoru paměti a fragmentace.

Velikost logického adresního prostoru je dána architekturou procesoru - kolik adres je procesor schopen generovat. Např. 32 bitový procesor generuje 2^32 adres. Velikost fyzického adresního prostoru je dána nainstalovaným hardware počítače - fyzicky dostupnou pamětí.

Souvislý LAP (Logický Adresní Prostor) není zobrazován jako jediná souvislá oblast FAP (Fyzický Adresní Prostor). FAP se dělí na úseky zvané rámce, LAP se dělí na úseky dané stránky (stránka = rámec).

Struktura logické adresy
Logická adresa se skládá ze dvou částí. První část je index v tabulce stránek (Page table) - díky němu se v tabulce stránek vyhledá fyzická adresa stránky. Druhá část je offset - posunutí ve stránce.

Převod logické adresy na fyzickou (pomocí MMU (Memory Management Unit))

- K logické adrese je v tabulce stránek nalezena fyzická adresa dané stránky/rámce.
- K adrese rámce je přičten offset a výsledek je požadovaná fyzická adresa.

Mapování, stránkování, segmentace

Stránkování
LAP je rozdělen do úseků, které na sebe navazují - stránky. Logická adresa odkazuje na adresu stránky, která odkazuje na adresu rámce FAP. Rámce v FAP na sebe nemusí navazovat.
Segmentace
LAP je rozdělena na segmenty. Segmenty jsou části programu - mají logický význam (hlavní program, procedura, funkce,...), jsou různě dlouhé - nízká vnitřní fragmentace. Výhody - lze určit přístup do nepovolené části paměti (segmentation fault), se segmenty v paměti lze libovolně hýbat, lze nastavovat práva k přístupu do segmentu. Nevýhoda - externí fragmentace.
Stránkování a segmentace najednou
Výše uvedené metody lze kombinovat. Segmentace vybírá části LAP, stránkování zobrazuje LAP do FAP - LAP je dělena na segmenty, které jsou stránkovány.
https://youtu.be/Z4kSOv49GNc

Přerušení a výjimky

Přerušení

- cílem je zlepšení účinnosti systému
- je potřeba provést jinou posloupnost příkazů jako reakci na nějakou „neobvyklou" událost
- přerušující událost způsobí, že se pozastaví běh procesu v CPU takovým způsobem, aby ho bylo možné později znovu obnovit, aniž by to přerušený proces „poznal"
- využití např. při IO operacích
- testování, zda je voláno přerušení, se koná alespoň po dokončení každé instrukce
- přerušení bývá často voláno programem
- Maskovatelná, lze je zakázat ve stavovém řídícím slovu CPU, případně řízení priorit (periferie, čítače, časovače)
- Nemaskovatelná - ošetření HW chyb, hlídací obvod (Watch Dog)

Výjimka

- Výjimka – ošetření zvláštních situací, které brání dalšímu vykonávání instrukcí (exception)
  - Matematické přetečení (výsledek instrukce s kontrolou saturace přetekl)
  - Načtena nedefinovaná instrukce (neznámý operační kód instrukce typu IR, nebo neznámá funkce instrukce typu R)
  - Systémové volání (instrukce syscall)

Periferie

- Pravidelně dotazovány jestli posílají data (polling) - pomalé
- Používá se přerušení - při příjmu znaku se vyvolá na CPU přerušení a zpracuje se - středně rychlé
- Ukládají do paměti RAM - ve fyzické paměti na danou adresu se ukládá/čte hodnota pomocí DMA - velmi rychlé

Direct Memory Access - přímý přístup do paměti. Využívá se při přenosu velkého množství dat - data nemusí jít přes procesor a nevytěsňují tak data z cache.

- Program/OS nastaví parametry přenosu
- Procesor nastaví adresy do DMA řadiče, ten na konci přenosu vyvolá přerušení

Point-to-Point - linka mezi dvěma zařízeními
Sériový port (vycházím z videa 7 od Štěpána s tím, že tím je asi myšlena Sériová linka) - komunikace point-to-point, full-duplex, komunikační kanál, ze začátku signál jde z stop bit na start bit, má vodič Tx na vysílání a Rx na přijímání dat
Sériová sběrnice - data se přenáší bit po bitu
Paralelní sběrnice - data se přenáší paralelně po více vodičích (8 vodičů = 8 bitů za jeden hodinový cyklus), problém přeslechu z jednoho vodiče do druhého, nutná synchronizace na konci (řešení je pomalejší rychlost)

Half-duplex - komunikace probíhá jen jedním směrem
Full-duplex - umí přijímat i vysílat současně
PCI sběrnice - paralelní sběrnice (32-64 vodičů), half-duplex, vodič na clock, frame (zda se vysílá paket nebo ne)
PCI-express - sériová, full-duplex, paralelizace sériové sběrnice

Zdroje přerušení, přerušovací vektory

Určení zdroje výjimky přerušení
Softwarové vyhledání (polled exception handling)

- Veškerá přerušení a výjimky spouštějí rutinu od stejné adresy – např. standardní MIPS, adresa 0x00000004

- Rutina zjistí důvod ze stavového registru (MIPS: cause registr)

Vektorová obsluha přerušení

- Již hardware CPU zjistí příčinu/číslo zdroje
- V paměti se nachází na pevné/řídicím registrem specifikované (VBR) adrese tabulka vektorů přerušení
- Procesor převede číslo zdroje na index do tabulky
- Z daného indexu načte slovo a vloží ho do PC

Nevektorová obsluha více pevně určených adres podle priorit/důvodu

- Často jsou přístupy kombinované, např. výjimky mají oddělené cílové adresy skoků, využívají tabulku atd., ale veškerá vnější přerušení končí pouze na jednom z vektorů