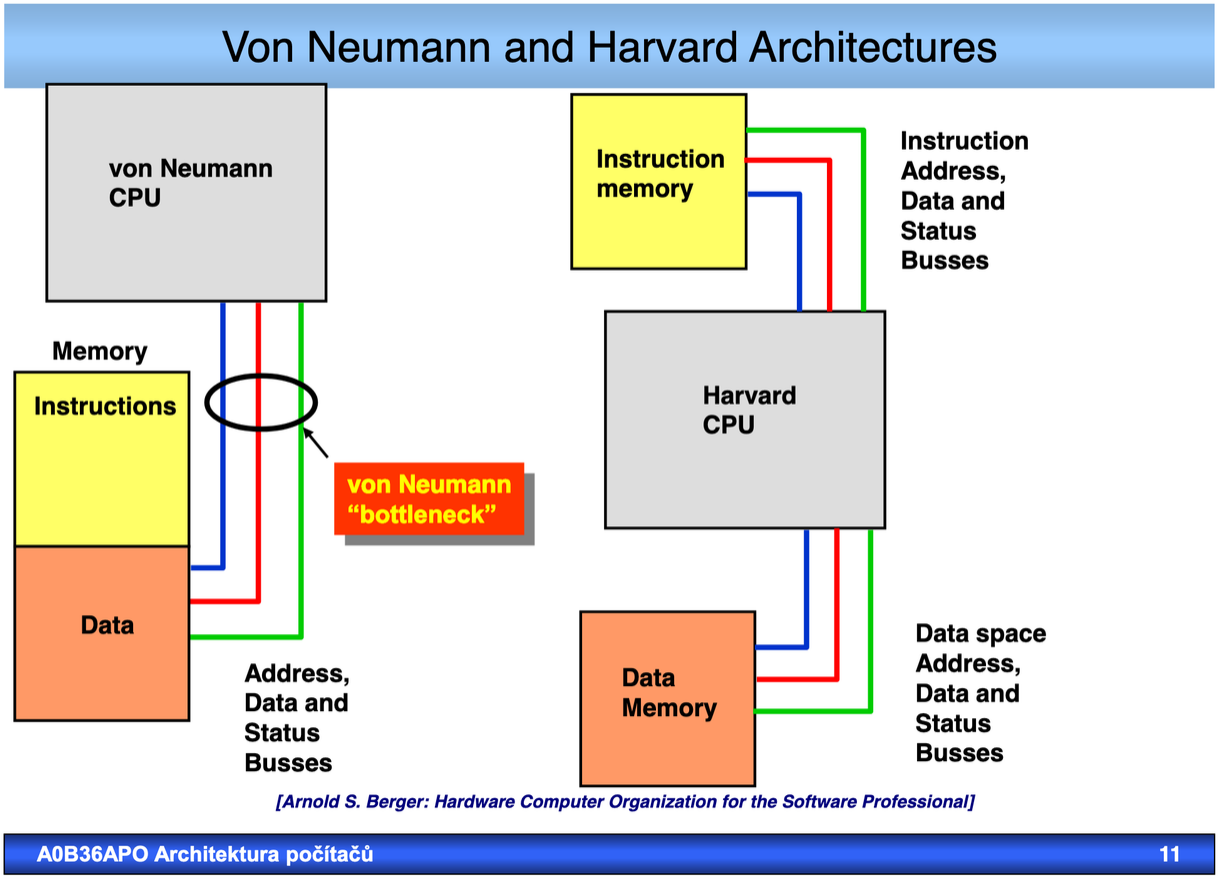
7. Architektura počítače; CPU, paměti, subsystémy.

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

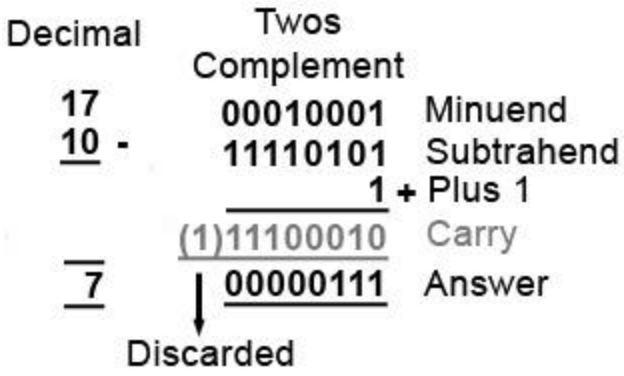
Architektura počítače

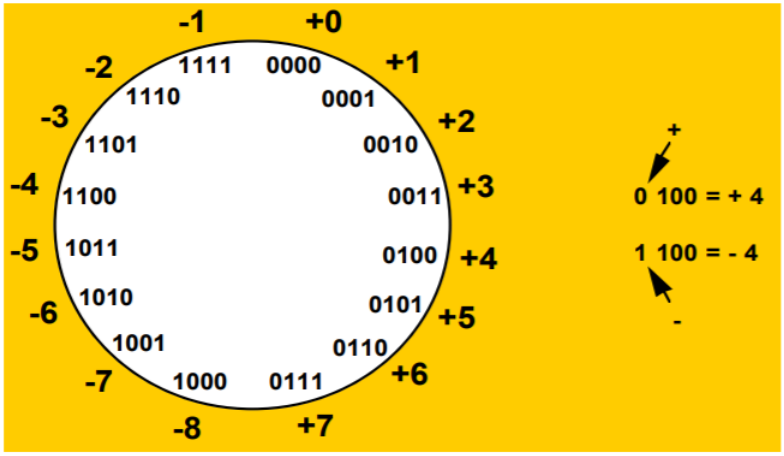
Von Neumanova - jednodušší, pomalejší, možnost rozdělit paměť data/instrukce dle potřeby **Harvardská** - paměť pro instrukce a data fyzicky oddělena Sběrnice propojující paměti se skládá ze tří sběrnic: adresní, datové, řídící



Reprezentace čísla v počítači

Two's complement





Endianita

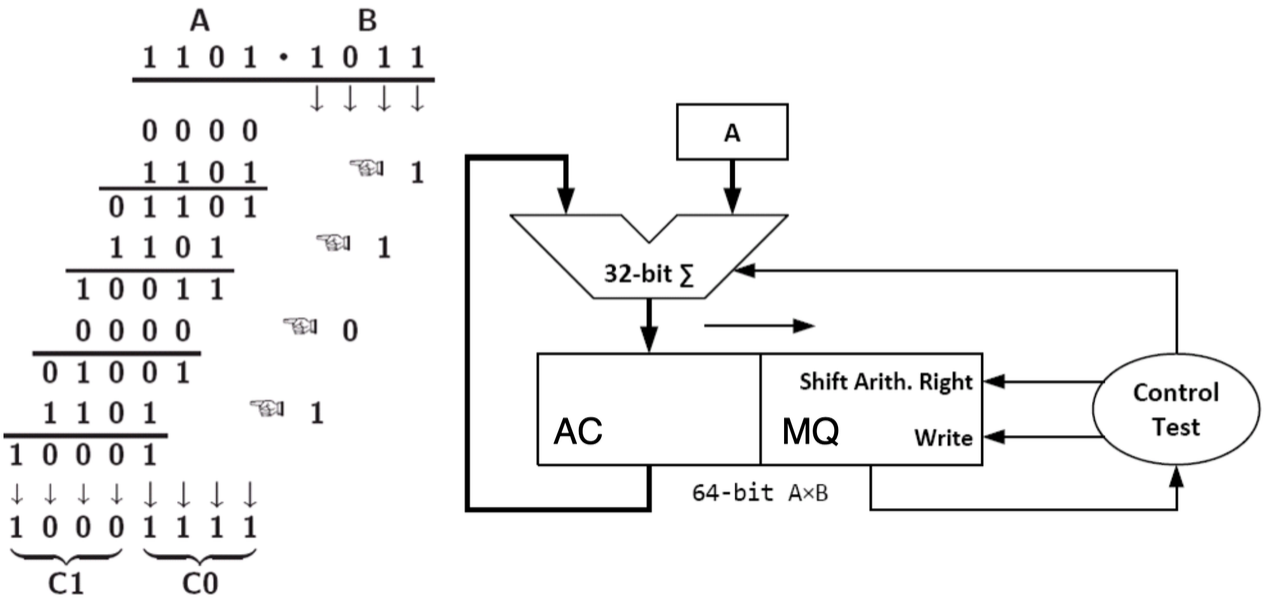
- Little endian: V tomto případě se na paměťové místo s nejnižší adresou uloží nejméně významný byte
  32bitové číslo `0x4A3B2C1D` se na adresu `100` uloží takto:

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

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

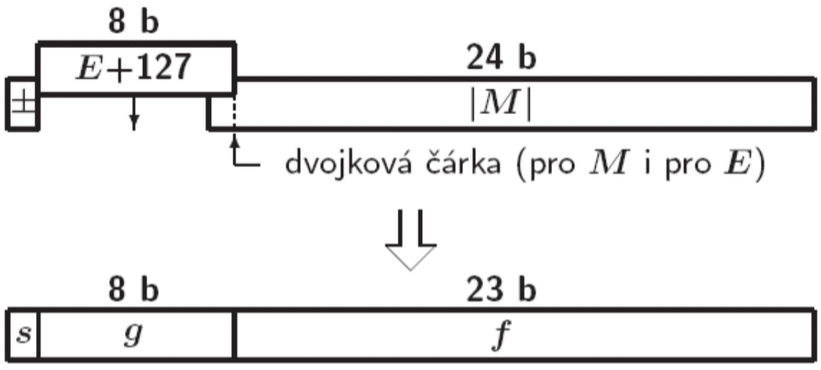
	100	101	102	103	
...	4A	3B	2C	1D	...

Sekvenční HW násobička (varianta 32b)



M - mantisa (zlomková část), v přímém kódu
E - exponent, v aditivním kódu

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



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

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

Prevod cisla na binarni 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:

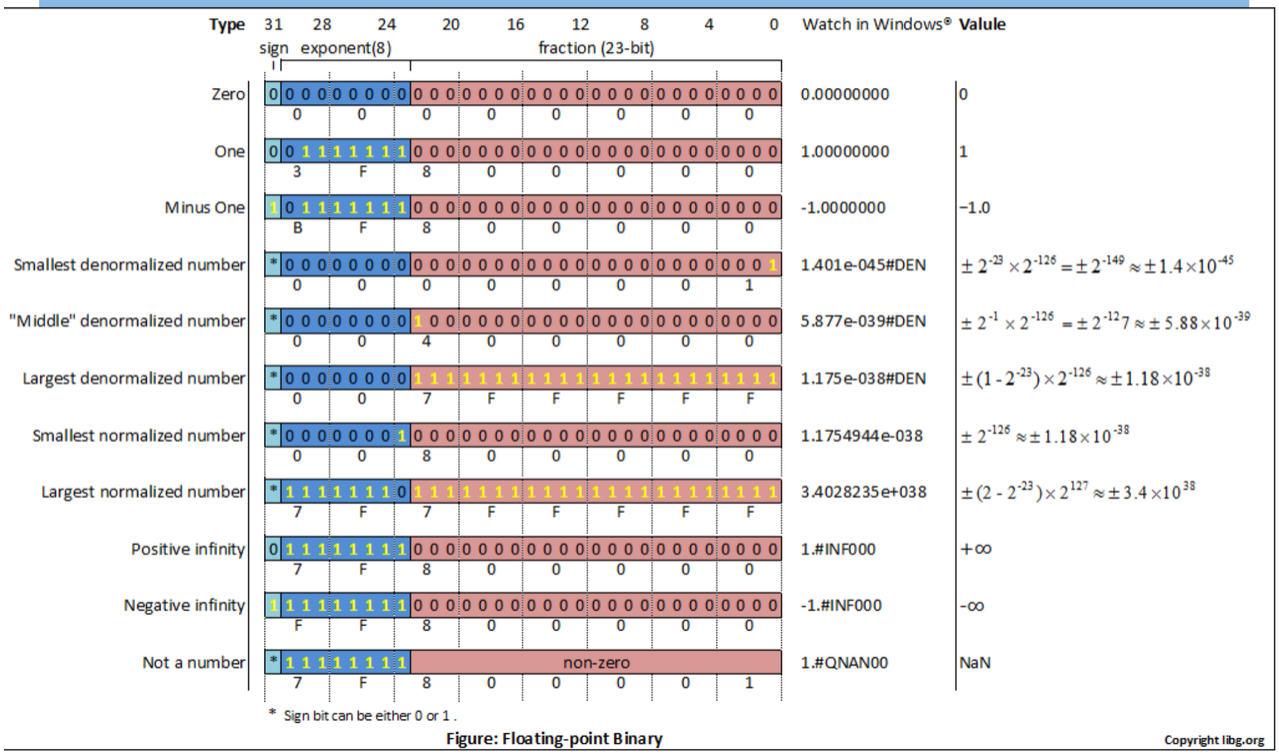
Vyplňte pole, znaménko je záporné -> S=1.

Exponent + 127 -> 130 -> 1000 0010 .

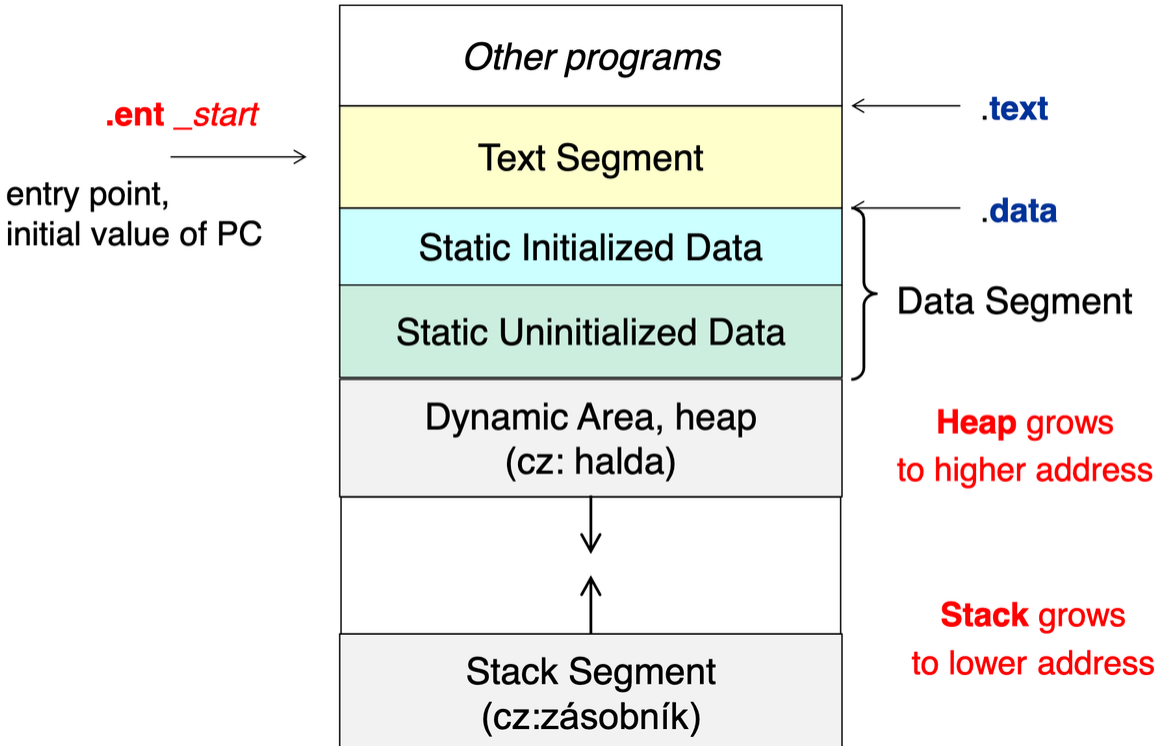
Úvodní bit 1 mantisy je skrytý ->

1 1000 0010 . 1001 0100 0000 0000 0000 000

Širší přehled



Layout of program in memory

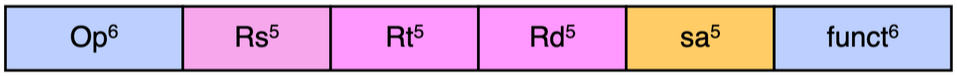


Instruction format

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 .

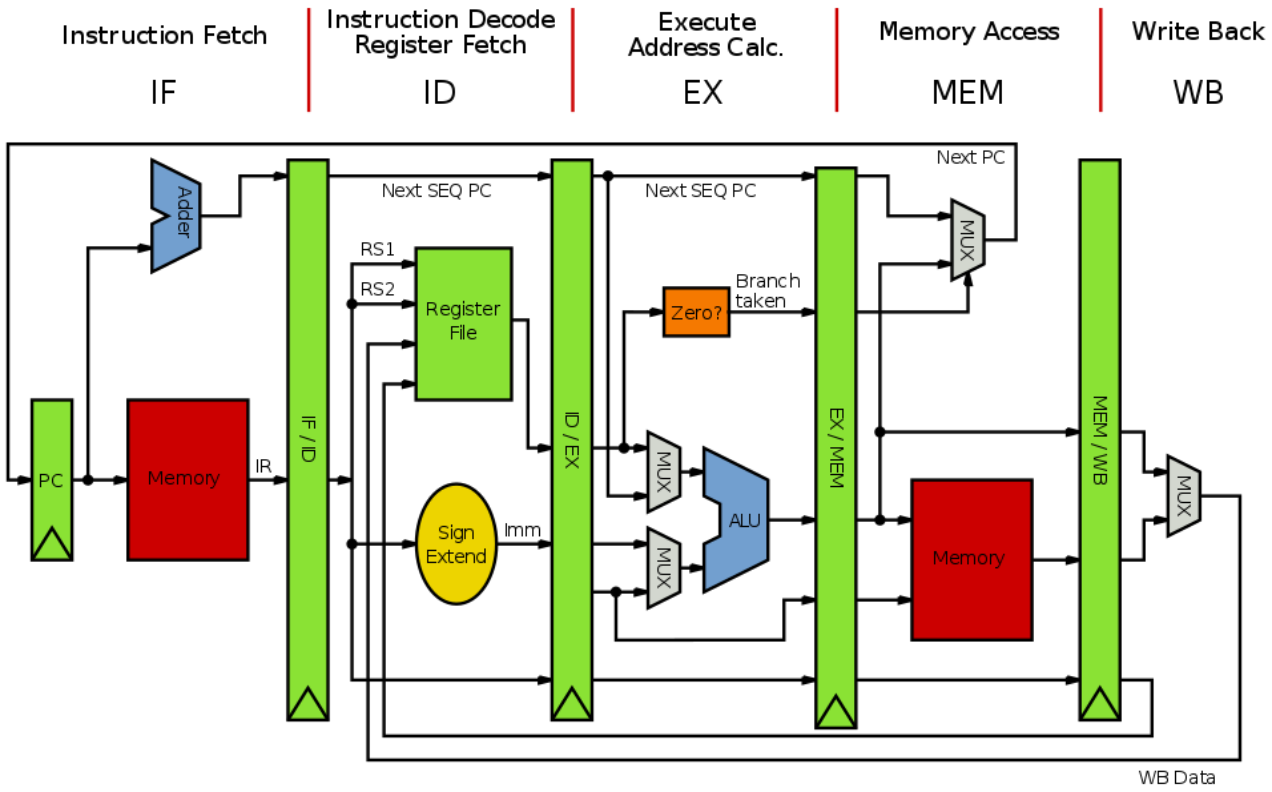
Vice o instrukcich na:
https://cw.fel.cvut.cz/wiki/_media/courses/b35apo/lectures/03-n-aritmetika3-procesor.pdf

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
ALU (aritmeticko-logická jednotka) (může jich být i více) zajišťuje všechny aritmetické a logické výpočty
PC (program counter) - čítač instrukcí - uchovává stav paměti, procesor má vždy po restartu nastavenou určitou hodnotu registry

- *datové* - ukládání hodnot
- *adresní* - uchovávají adresy odkud mají být data načítána / kam ukládána
- procesor může vykonávat aritmetické/logické operace pouze nad daty v registrech

Ukazatel zásobníku - stack pointer - SP

- uchovává adresu posledního záznamu uloženého na zásobníku
- obvykle "růst dolů": při push se *SP* zvětší, při pop se *SP* snižuje
- 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

T**ypy instrukcí**
CISC complex instruction set computer

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

RISC reduced instruction set computing

- snížení počtu instrukcí, snaha dosáhnout 1 takt=1 instrukce

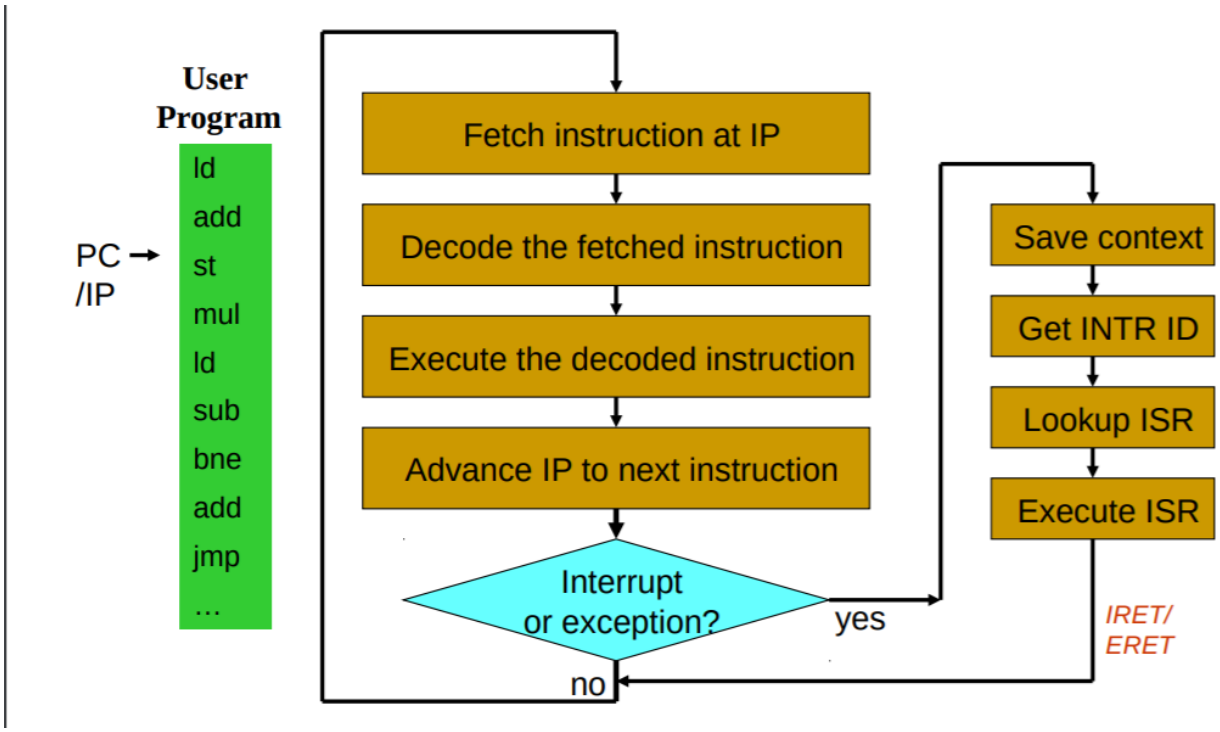
Které z následujících tvrzení je pravdivé?

- a) Pro architekturu CISC je typické větší množství identických univrezálních 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)
```

0.4.3 jeden cyklus cpu - asi není nezbytné
Nevíme, co znamená HP, asi Heap Pointer.
Tak jako tak, prvním krokem je načtení obsahu paměti na adrese PC, asi tím mysleli adresu v hlavní paměti... Tohle je detail.

- 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, 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 je rozdělena minimálně na 2 úseky:

1. Načtení a dekódování instrukce
2. Provedení instrukce a případné uložení výsledku

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 vícejádrových 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
```

Architektura paměti

Struktury a hierarchie pamětí

- Rychlé a malé paměti jsou umístěny blízko CPU.
- Větší a pomalejší paměti jsou dále od CPU.
- Princip cachování nejčastěji používaných dat v rychlých pamětech - zrychlení přístupu k datům.
- od nejrychlejší a nejmenší po největší a nejpomalejší:
  - on-chip L1 caches
  - off-chip L2 caches (SRAM)
  - L3 společná pro více vláken (logických CPU)
  - hlavní paměť (DRAM)
  - 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.

Způsoby adresace

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

Stránkování

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

Struktura logické adresy

Logická adresa se skládá ze dvou částí. První část je index v tabulce stránek - díky němu se v tabulce stránek vyhledá fyzická adresa stránky. Druhá část je offset - posunutí ve stránce.

Převod logické adresy na fyzickou

- 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.

Různá šíře logických a fyzických adres

CPU může mít menší datovou sběrnici, než je šíře fyzické adresy. Např. fyzická adresa má 20 bitů, ale datová sběrnice CPU je jen 16 bitů. Zbývající 4 bity jsou offset segmentu a jsou uloženy ve 4 segmentových registrech CPU. FA = (segment << 4)+offset.

Opačný případ - CPU generuje více logických adres (např. 64 bitový procesor generuje 2 64 adres) než je fyzických adres (LA je větší než FA). Tento problém se řeší virtualizací paměti. Virtualizace spočívá v tom, že FAP se rozšiřuje o úseky na vnější paměti (např. na pevném disku). LAP lze také využít k přístupu k periferiím. V předmětu APO jsme takto například přistupovali k LED na té vývojové desce.

0.4 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

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)

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

0.6 DMA přenosy

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

Architektura subsystém

Řadiče

- řadič anglicky control unit
- jasně odlišitelná část systému, která řídí nějaký úkon
  - např.: řadiče displaye, jednotka řídící teplotu vody , atd..
- v CPU se stará o řízení toku dat a o řízení práce všech jednotek, zejména ALU, a to v závislosti na právě vykonávané instrukci

Programovatelný řadič

- varianta sekvenčního obvodu realizovaná přes paměť
- flexibilní

Pevný řadič = Řadič klasický, též obvodově realizovaný, tedy tzv. obvodový

- rychlejší
- automat realizovaný přes sekvenční obvody
- ve velmi jednoduchých případech levnější

Mikroprogramovatelný automat

- řadič, který nefunguje s fixní konfigurací, ale používá tzv. mikrokód
- použití v CPU -> instrukce z instrukční sady se překládá na sekvenci mikroinstrukcí, nahrazuje rozsáhlou logiku pevného řadiče

- mikroinstrukce definují, které hardwarové části je potřeba propojit, aby byla vykonána samotná instrukce
- oproti fixní konfiguraci má výhodu v možnosti opravy chyb procesoru pomocí aktualizace tabulky překladu instrukce -> mikroprogram
- mikroinstrukce se provádějí velmi rychle a lze je paralelizovat
- příklad mikroprogramu jedné instrukce přičítání např.:
  - přiveď registr AX k ALU jako první operand
  - přiveď dočasný registr k ALU jako druhý operand
  - nastav ALU do režimu sčítání
  - nastav carry na 0
  - ulož výsledek do regstru AX
  - nastav příznaky
- Tato sekvence je částí vykonávání jedné instrukce. V kontrastu, pevný řadič by operandy ALU dekódoval a logickým součinem povoloval / zakazoval přímo z operačního znaku instrukce. Stejně tak by podle operačního kódu řídil režim činnosti ALU a umístění výsledku. Pro větší počet instrukcí narůstá velikost potřebné logiky.