# 14. Technologie soudobých procesorů

## *Počítač*

Počítačem je **zařízení schopné přijmout data, aplikovat na ně předepsaný proces a vydat výsledky**. Jinými slovy lze říci, že počítač je stroj pro zpracování dat a získávání informací. Data jsou v počítači zobrazena pomocí fyzikální veličiny (nemusí to být nutně el. napětí). Tato veličina může mít buď spojitý, nebo diskrétní charakter. Dále počítače dělíme na digitální (číslicové), analogové, a hybridní.

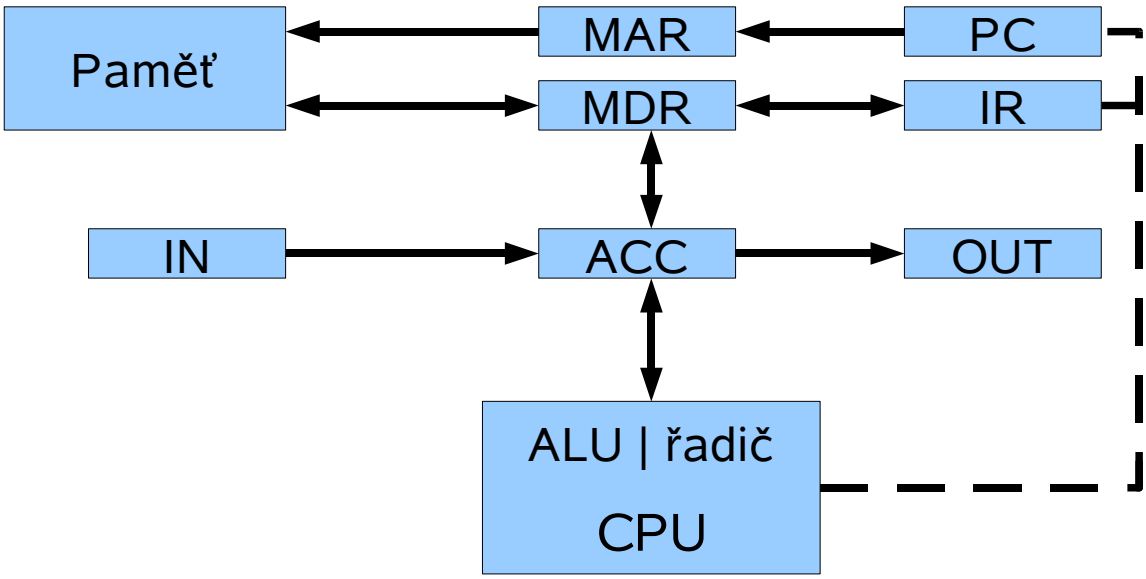
## *von Neumanova koncepce počítače*

základní principy definované von Neumannem:

- 5 základních funkčních bloků – paměť, ALU, zařízení vstupu a výstupu, řídící jednotka (řadič)
- struktura počítače je neměnná, činnost řídí program uložený v paměti. Změny v činnosti stroje jsou dosaženy změnou v programu
- paměť je společná pro data i instrukce
- paměť je rozdělená na stejně velké buňky, které jsou průběžně očíslované, přes číslo buňky (adresu) se dá přečíst nebo změnit obsah buňky
- program je posloupnost instrukcí, které určují elementární změnu stavu stroje.
- pořadí prováděných instrukcí je lineární, výjimku tvoří instrukce skoku
- instrukce, adresy a data jsou kódovány binárně

Von Neumannova koncepce předpokládá, že v jednom okamžiku bude v činnosti maximálně jedna jednotka. Tedy má li procesor N jednotek, každá jednotka N-1 cyklů nic nedělá.

## *Obecný model počítače*




- MAR – Memory adress registr
- PC – Programový čítač

- MDR – Memory data registr
- IR – Instrukční registr
- IN/OUT – Vstup/Výstup
- ACC – Akumulátor

Synchronizace funkčních bloků počítače je prováděna pomocí hodinového impulsu. Hodinový cyklus je doba mezi dvěma čely hodinového impulsu a je to nejmenší rozlišitelné jednotka instrukčního cyklu. Instrukční cyklus je doba potřebná k vykonání jedné instrukce.

## *Harvardská koncepce počítače*

Základním rysem harvardské koncepce je paměť rozdělena na oblast určenou pro ukládán instrukcí a na oblast pro ukládání dat. Výhodou této koncepce je možnost nezávisle číst z paměti data a instrukce zároveň, což počítač podle on Neumanna neumožňuje. Další výhodou rozdělení paměti je možnost použít rozdílnou délku adresové jednotky zvlášť pro data a zvlášť pro intrukce. Dnešní procesory kombinují dohromady obě koncepce. Vnitřně pracuje s pamětí cache rozdělenou pro data a instrukce a vně komunikuje s operační pamětí, společnou pro data a instrukce.

## *Instrukce*

Instrukce je kódovaný příkaz procesoru říkající co se má udělat, s čím se to má udělat, kam se má uložit výsledek a kde je následující instrukce. všechny 4 části instrukce musí být jednoznačně přímo, nebo nepřímo určené. Instrukce má 4 části a 5 logických polí. Operační kód, prvý operand, druhý operand, místo uložení výsledku a adresu následující instrukce.

Provedení instrukce je jediný způsob jak změnit stav stroje. Vykonaná změna stavu se promítá do dvou oblastí – výsledků operace a nastavení příznaků. Příznak charakterizuje ukončenou instrukci a může být použit jako vstupní parametr následující operace.

***instrukční repertoár*** – množina pravidel integrovaná v hardware procesoru, která zcela přesně určuje jednotlivé stavy stroje (jinak také instrukční soubor).

instrukční soubor lze dělit podle různých kritérií:

- instrukce privilegované (nejsou dostupné každému programu)
- instrukce neprivilegované (jsou dostupné každému programu)
- instrukce přenosu dat (paměti, registry, zásobníky)
- diadické instrukce (aritmetické a logické operace)
- monodické instrukce (nulování, inkrement, negace, rotace, posuny)
- instrukce větvení, skoků a cyklů
- instrukce volání podprogramů
- instrukce vstupu, výstupu
- instrukce pro řízení stroje

Většina je implementována jako řada podobných instrukcí nebo jako jedna instrukce, která bude mít různé modality (variace operačního kódu)

# *Instrukční sady*

## RISC

- redukovaný instrukční soubor.
- pouze jednoduché způsoby adresování
- pevný formát instrukce
- instrukce je vykonána v jednom strojovém cyklu
- operace s daty pouze nad registry
- pro styk s paměti se používají výlučně operace LOAD/STORE
- zřetězená realizace instrukcí
- nutnost používat optimalizující kompilátor

RISC procesor vystačí pouze se 30 typickými instrukcemi. RISC počítač musí s tímto souborem pracovat častěji než CISC, ale díky většímu počtu registrů může program proběhnout rychleji, protože většina operací se koná přímo mezi registry a pamětí. CISC CPU má sadu obvykle 16 registrů, zatím co RISC má až 100 volných registrů. Z toho důvodu není centrální jednotka vůbec zatěžována a mimo to je mikrokód zbytečný. Instrukce jsou implementovány hardwarově (nejsou dekódované). Podstatné vlastnosti RISC - architektury umožňující vysokou propustnost dat, mají malý počet jednoduchých instrukcí, instrukce s pevnou délkou a pevným formátem, přímá hardwarová interpretace instrukcí, jednocyklový příkazový režim a díky internímu režimu Pipeling překrývané provádění po sobě následujících instrukcí. Hlavní nárůst rychlosti RISC byl dosažen použitím velkého množství registrů. Tím, že odpadly mikroprogramy, mohla být uvolněna celá oblast na povrchu čipu, která byla tradičně používaná pro mikroprogramy a použitá pro velmi rychlou paměť. U procesorů RISC je jednodušší a rychlejší změna návrhu čipu, než u komplikovaných a komplexních struktur CISC. Architektura RISC má slabinu ve výpočtech v plovoucí desetinné čárce, proto byl zaveden speciální koprocesor pro numerické výpočty.

## CISC

- komplexní instrukční soubor
- snaha o zachování zpětné kompatibility
- rozsáhlý instrukční soubor (přes 200)
- málo registrů
- Řadič mikroprogramový (každá nová instrukce nový mikroprogram, nebo jejich sled)
- Určitá skupina bitů v instrukcích má různý význam. Jedna skupina určuje, co druhá vlastně znamená.
- proměnlivý formát instrukcí

Systémy s rozsáhlým komplexním souborem instrukcí. Pro tyto procesory je typická implementace architektury pomocí mikroprogramování. Výsledkem je velký počet specializovaných typů instrukcí, z časového pohledu mohou trvat až 300 strojových cyklů. Mikroprogramování poskytuje možnost nabízet spektrum strojů se stejnou architekturou, ale přesto rozdílnou hardwarovou realizací. S rostoucím objemem sady instrukcí však bylo pro překladače překládající programy do strojového kódu těžké využít celou škálu speciálních instrukcí. Z toho vyplývá, že komplexní instrukce jsou používány jen zřídka. Při zpracování programu z vyššího programovacího jazyka se velká část času spotřebuje na čtení informace z pracovní paměti a naopak.

## Možnosti navýšení výkonu

- Zvyšování frekvence procesorů – hrubé zvyšování výkonu, nelze zvyšovat výkon donekonečna (určité fyzikální limity), problémy s chlazením (procesor generuje příliš mnoho tepla)
- zvyšování stupně integrace – čím menší dokážeme obvody vyrobit, tím víc funkčních bloků může dát do jednoho čipu.
- proudové zpracování instrukcí – v jednom okamžiku se vykonává více než jedna instrukce, je třeba rozdělit kroky vykonání instrukce na přibližně stejně dlouho trvající kroky, které lze provádět nezávisle. Zkrátí dobu čekání jednotky z N-1 na kratší časový úsek.
- superskalární architektura (paralelní zpracování instrukcí) – Dochází k replikaci některých často využívaných jednotek (ALU, FPU).
- HW podpora funkcí OS – funkce dříve realizované OS jsou integrované v procesoru, čímž dochází k zvýšení výkonu celého stroje

## Překrývání činností

Nová instrukce se začne zpracovávat dříve, než je rozpracovaná instrukce plně dokončena.

příkladem překrývání činností je procesor I8086. Používá dvě jednotky (EU,BIU) EU provádí instrukce uložené ve frontě a BIU mezitím komunikuje s pamětí.

## Proudové zpracování

nejčastěji se proudově provádí:

- aritmeticko-logické operace (proudová sčítačka)
- realizace instrukcí (výběr, dekódování, výpočet adres operandů, provedení, zápis výsledků)
- realizace procesů (prokládaná paměť, sběrnice s rozdělenými transakcemi)

Proudové zpracování předpokládá rozdělení nějaké činnosti na posloupnost kroků. Každý krok je prováděn samostatnými technickými prostředky, které realizují určitou operaci. V jednom okamžiku je prováděno několik činností, každá v jiném stupni rozpracovanosti a každá využívá jiný stupeň řetězu technických prostředků.

předpoklady pro proudovou realizaci nějaké činnosti:

- soustavný přísun dat, nad kterými se provádí příslušná činnost
- činnost musí být rozdělitelná na sekvenci nezávisle proveditelných operací
- trvání každé jednotlivé operace musí být přibližně stejné

proudové zpracování nepřináší jen zvýšení výkonu, ale i vznik problémů. Předpis pořadí zpracování instrukcí není jednosměrný (skoky, smyčky), což vede k datovému a skokovému konfliktu.

- **datový konflikt** - vznikne, když rozpracovaná instrukce nemůže být dokončena, protože

potřebuje pracovat s daty, jejichž modifikací provádí předchozí (taktéž nedokončena) instrukce.

- **skokový konflikt** - vznikne při zjištění, že právě dokončená instrukce mění posloupnost vykonávaných instrukcí a všechny rozpracované instrukce je třeba zrušit (prováděly se zbytečně).

Uvedené problémy je možné řešit softwarovou nebo hardwarovou cestou. Optimalizující kompilátor je jedno z možných (a poměrně nedokonalých) řešení (co třeba s existujícími, už přeloženými programy). Hardwarová řešení předpokládají vyřešení konfliktů za běhu programu tak, aby nedošlo k významnému snížení výkonu stroje. Používají se technologie umožňující předpovídání (predikci) vícenásobného větvení a analýzu datových závislostí, což vede k vykonávání instrukcí mimo pořadí. Stroje s velmi dlouhým instrukčním slovem (VLIW) řeší problém konfliktů kombinací obou přístupů HW + SW.

proudová sčítačka v pevném formátu – řada jednobitových sčítaček s posunem.

proudová realizace sčítání v plovoucí řádové čárce:

- odečtení exponentů
- posun mantisy menšího čísla s menším exponentem doprava o počet bitů, který je roven rozdílu exponentů
- sečtení mantis
- určení počtu nul mezi řádovou čárkou a první platnou číslicí součtu mantis
- posunutí součtu doleva o tolik míst, kolik bylo nalezeno 0 za řádovou čárkou
- zmenšení původního exponentu o počet nalezených 0

poslední dva kroky se provádějí paralelně.

## *Replikace jednotek*

30-40% jednotek pracuje s celými čísly, proto je výhodné použít více jednotek pro práci pevnými čísly. Nejčastěji se používají dvě jednotky – jedna plnohodnotná a druhá s redukovanými funkcemi. Sekvenční přísun dat není schopen zásobit více jednotek, proto dochází k replikaci vydávací jednotky.

Superskalární procesor – na jeden takt více než jedna instrukce

Problémy vznikající při paralelním zpracování jsou podobné jako při proudovém zpracování. Novým problémem se stává potřeba aby všechny funkční jednotky byly vytížené. Řešení je několik:

- kompilátor – v době kompilace se snaží umístit instrukce tak, aby byly všechny jednotky vytíženy
- dynamický HW způsob – jednotka před výběru instrukce, případně provádění instrukcí mimo pořadí
- architektura VLIW (Very Long Instruction Word) – architektura s velmi dlouhým instrukčním slovem. V jednom instrukčním slovu je činnost pro všechny jednotky.

## *Pentium 4*

charakteristika:

- 7. generace procesorů firmy Intel
- architektura NetBurst

- vylepšení pro práci s čísly s plovoucí řádovou čárkou (instrukční sada SSE-2; 128b aritmetika
- tloušťka čáry – 180nm
- 42 000 000 tranzistorů
- frekvence 1,5 GHz (spotřeba 55W)
- systémová sběrnice 64b, 100MHz, 3.2 GB/s, 4 přenosy na jeden takt
- různé frekvence funkčních jednotek

Funkční části:

vstupní část

- zpracování instrukcí v předepsaném stavu
- přísun instrukcí z L2 cache (úprava aby byly proveditelné v EU)
- cache L1
- jednotka predikce skoku
- dekódovací jednotky – převádí instrukce z IA-32 na mikrooperace
- ROM mikrokódu

zpracování instrukcí mimo pořadí

- jednotka alokace vyrovnávací paměti, přejmenování registrů a provádění mikrooperací mimo pořadí

výkonné jednotky

- 2x ALU (dvojnásobná rychlost čipu)
- 2x FPU
- jednotky generace adres operandů
- datová cache L1

paměťový subsystém

- cache L2 – 256KB(Data+Instrukce)
    - 8cestná asociativní paměť (řádek 128B)
    - strategie zpožděného zápisu
    - výkon 1,5GHz – 48GB/s
    - spolupracuje s jednotkou předvýběru – monitoruje přistup, usiluje o doplnění (optimální náskok 256B)
    - snaha detekovat nezávislé programové celky (aby v cache bylo co bude potřeba a nebylo co potřeba nebude)