Generácie počítačov:

0. generácia (1936 – 1946)
- **elektromechanické počítače, základná súčiastka elektromagnetické relé**
- za prvý zostrojený programovateľný počítač sa považuje počítač, ktorý zostrojil Nemec Konrad Zuse v roku 1936; bol nespoľahlivý a nevhodný pre praktické použitie
- v roku 1943 Howard Aiken a Grace Hopper dokončili počítač **MARK 1** (USA)

1. generácia (1946 – 1956)
- **elektrónkové počítače, základná súčiastka vákuová elektrónka**
- prvý univerzálny plne elektronický počítač s názvom **ENIAC** (Electronic Numerical Integrator and Computer alebo Calculator) bol daný do prevádzky 16. februára 1946 v USA
- COLOSSUS (1943-1944, VB) bol prvým využívaným plne elektronickým počítačom používaným na prelomenie šifier nemeckého šifrovacieho stroja ENIGMA
- assembler – jazyk symbolických inštrukcií

2. generácia (1956 - 1964)
- **základná súčiastka polovodičový tranzistor**
- prvým počítačom obsahujúcim tranzistory (ešte však kryštálové patentované v roku 1947) bol počítač EDVAC
- skonštruovanie magnetického disku
- vyššie programovacie jazyky (FORTRAN, ALGOL, COBOL)
- prvé operačné systémy

3. generácia (1964 - 1971)
- **použitie integrovaných obvodov**, ktoré na svojich čipoch integrujú veľké množstvo tranzistorov
- sériová výroba počítačov IBM System-360
- jazyka BASIC, Pascal

4. generácia (1971 – súčasnosť)
- **miniaturizácia súčiastok až po mikroprocesor** (procesor integrovaný na jedinom polovodičovom čipe; procesor – súčiastka schopná pracovať ako subsystém určitého systému, napr. CPU, grafický, zvukový, matematický koprocesor)
- rozšírenie operačných systémov a aplikácií
- osobný počítač IBM PC (1981)
- mikropočítače

5. generácia
- výskum v oblasti umelej inteligencie, neurónových sietí, kvantových a molekulových počítačov

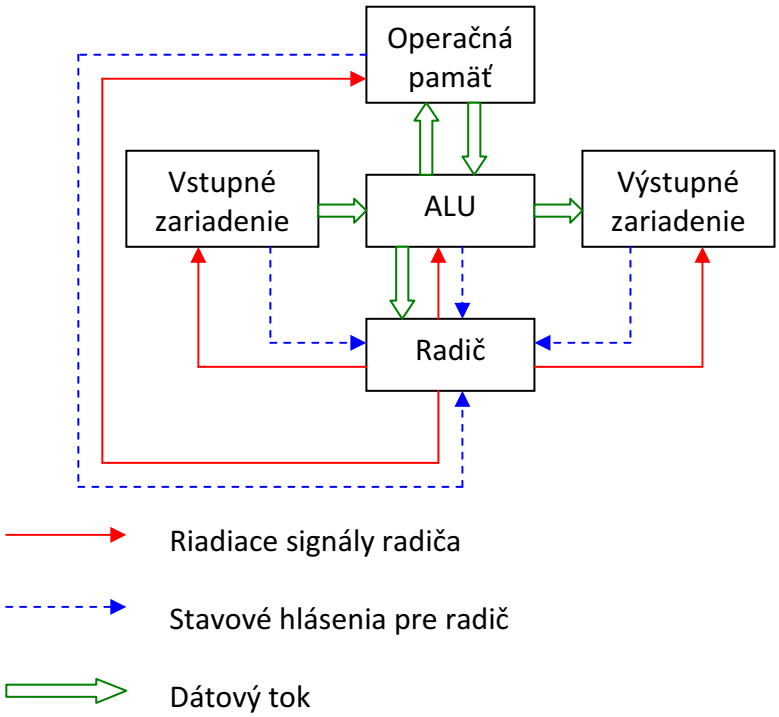
Počítačová architektúra popisuje spôsob, akým treba jednotlivé súčiastky a komponenty prepojiť tak, aby počítač spoľahlivo a rýchlo pracoval.
Súčasné počítače najčastejšie vychádzajú z **von Neumannovej architektúry** s vlastnosťami:
- počítač sa skladá z procesora, pamäte a vstupno-výstupných zariadení
- program je uložený v pamäti počítača
- procesor vykonáva inštrukcie programu postupne
- údaje sa spracúvajú v dvojkovej sústave.

Poznámka:
von Neumannova architektúra je označovaná aj ako princetonská architektúra, inou je harvardská architektúra (fyzicky oddeľuje pamäť programu a dát a ich spojovacie obvody; názov pochádza od počítača Harvard Mark I, ktorý bol postavený na tejto architektúre).

Princíp činnosti počítača podľa von Neumannovej schémy:
1. Do operačnej pamäte sa pomocou vstupného zariadenia cez ALU (aritmeticko-logickú jednotku) umiestní program, ktorý bude vykonávať výpočet.
2. Rovnakým spôsobom sa do operačnej pamäte umiestnia dáta, ktoré bude program spracovávať.
3. Vykoná sa vlastný výpočet, ktorého jednotlivé kroky vykoná ALU. Táto jednotka je v priebehu výpočtu, spolu s ostatnými modulmi, riadená radičom - riadiaca jednotka počítača, ktorá riadi jeho celú činnosť; riadenie sa uskutočňuje pomocou riadiacich signálov, ktoré predáva každému zariadeniu; reakciou na riadiace signály sú

stavové hlásenia radiča, ktoré sú mu posielané na spracovanie a následné rozhodnutie nad ďalším krokom. Medzivýsledky výpočtu sú ukladané do operačnej pamäte alebo registrov procesora.

4. Po skončení výpočtu sú výsledky poslané cez ALU na výstupné zariadenie.



ALU spolu s radičom tvoria procesor. Súčasťou procesora je ešte sada registrov, ktoré uchovávajú operandy a medzivýsledky a cache pamäte.
Ak sa chce zdôrazniť, že všetky údaje aj program sú uložené v operačnej pamäti, v schéme vyššie sa dátový tok smeruje do operačnej pamäte a nie ALU, cez ktorú len prechádza, analogicky pre výstupné zariadenie. Funkcia DMA (Direct Memory Access) umožňuje priamy prístup do pamäte bez prechodu dát cez procesor.

Nasledujúcu kapitolu z práce Evy Hanulovej (GJH Bratislava) by bola veľká škoda neuviesť.

Skôr ako začneme programovať

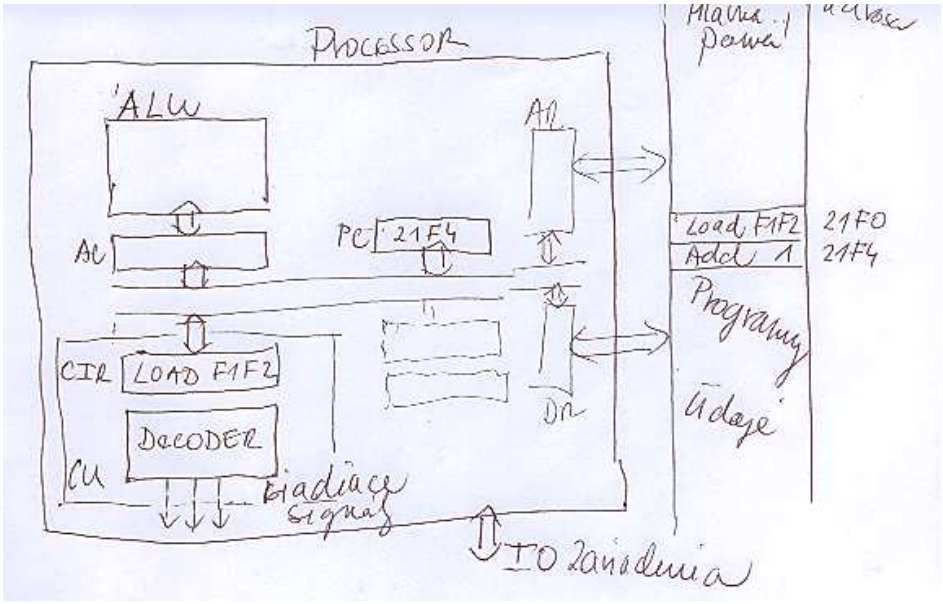
Počítač rozumie programu napísanému v **strojovom kóde**. Je to jazyk závislý na platforme počítača (Mac, IBM PC, …). Pod platformou rozumieme organizáciu hardvéru a komunikácie v počítači.

Počítač je rýchly stroj vykonávajúci stále tú istú monotónnu činnosť: vykonáva príkazy programu v strojovom kóde, hovoríme, že je riadený programom, ktorý práve vykonáva. Celú jeho „inteligenciu“ majú na svedomí programátori, ktorí vložili inteligentné správanie do svojich programov, či už programov operačného systému, prehliadačov alebo aplikácii.

Základnou časťou počítača je procesor - CPU (Central Processing Unit). Základné časti CPU tvoria riadiaca jednotka (Conrol Unit -CU) a aritmeticko-logická jednotka (Arithmetic Logic Unit – ALU). CU riadi činnosť počítača. V ALU sa vykonávajú aritmetické a logické operácie v dvojkovej sústave. V CPU sa nachádzajú okrem riadiacej jednotky a aritmeticko-logickej jednotky aj špeciálne rýchle malé pamäte – registre, ktoré slúžia na zapamätanie medzivýsledkov a iných dôležitých hodnôt v čase spracovania programu. Niektoré majú špeciálnu funkciu, ako napr.

- register PC (Program Counter), v ktorom sa pamätá adresa inštrukcie, ktorá sa bude vykonávať ako nasledujúca,
- register CIR (Current Instruction Register), do ktorého sa uloží kód inštrukcie načítanej z hlavnej pamäte, ktorá sa má práve vykonať. Riadiaca jednotka rozpozná kód inštrukcie a na základe toho zariadi pomocou riadiacich signálov vykonanie postupnosti krokov realizujúcich inštrukciu

Jednoduchá schéma počítača

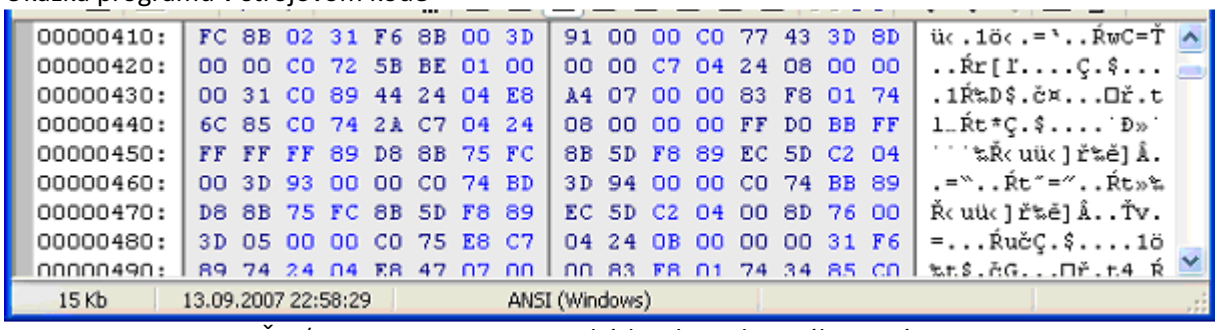


- registre na uchovávanie adries špeciálnych častí pamäte,
- registre na uchovávanie medzivýsledkov a
- registre na zaznamenanie stavu systému.

Príklad inštrukcií strojového kód

- presuň hodnoty z danej adresy v hlavnej pamäti do zadaného registra v CPU,
- zapíš hodnotu z registra do hlavnej pamäte na zadanú adresu,
- pripočítaj k registru hodnotu zo zadaného miesta v pamäti,
- presuň hodnotu z jedného registra do druhého,
- ak hodnota v registri je rovná nule, pokračuj v spracovaní programu inštrukciou na zadanej adrese v pamäti,
- načítaj hodnotu zo vstupného portu do registra v CPU a podobne.

Ukážka programu v strojovom kóde



Časť programu v strojovom kóde v hexadecimálnom zápise

Celá práca počítača sa dá popísať ako **neustále opakovania** inštrukčného cyklu počítača

1. načítaj do CIR inštrukciu, ktorej adresa je v PC (načítaj do CPU inštrukciu, ktorá sa má vykonať)
2. zvýš hodnotu v PC (aby tam bola adresa nasledujúcej inštrukcie)
3. dekóduj inštrukciu v CIR (deje sa v dekóderi, ktorý je súčasťou riadiacej jednotky)
4. vykonaj inštrukciu (riadiaca jednotka vysiela postupnosť riadiacich signálov ostatným častiam CPU, čím zabezpečí vykonania inštrukcie)

Na začiatku počítačovej éry programátori písali programy v strojovom kóde. Potom niekoho múdreho napadlo, že programátor sa nemusí trápiť so strojovým kódom a vymyslel a napísal program, ktorý program v inom programovacom jazyku dokázal preložiť do strojového kódu. Takémuto programu sa hovorí prekladač (translator). Najprv to bol taký menší pokrok, lebo prvý jazyk, ktorý bol poľudštením strojového jazyka bol jazyk symbolických adries (assembly language). V počiatkoch sa líšil od strojového kódu len tým, že sa v ňom dali používať mnemonické názvy príkazov a buniek pamäte, v ktorých boli zapamätané hodnoty dôležité pre spracovanie

problému. Programátori už viac nemuseli počítaču zadávať program v dvojkovej sústave. Program pripomínal v jazyku symbolických adries bol pre školených ľudí čitateľný. Pred spustením museli text programu zadať kompilátoru, ten ho preložil do strojového kódu.

Ukážka programu v assembleri (za bodkočiarkami sú poznámky)

```
MOV R7, #99             ;bottle count kept in R7
MOV R12, R14            ;store caller return address

.beginverse             ;(_prints verses then returns to caller_)
BL  bottlesofbeer
ADR R0, onthewall
SWI "OS_Write0"         ;prints string at address in R0
BL  bottlesofbeer
SWI "OS_NewLine"
ADR R0, take
SWI "OS_Write0"
SUBS R7,R7,#1
BLNE bottlesofbeer      ;beer left
BLEQ nobeer             ;no beer left
ADR R0, onthewall
SWI "OS_Write0"
SWI "OS_NewLine"
SWI "OS_NewLine"
BNE beginverse          ;go again if there's beer left
BL buymorebeer          ;print last verse
MOV PC, R12             ;exit to caller
```

zdroj: http://99-bottles-of-beer.net/language-assembler-(arm)-799.html

Človek si rád uľahčí svoju prácu a tak nasledovala postupnosť vyšších programovacích jazykov. Ich príkazy sú podobnejšie prirodzenej reči, preto sa v nich ľahšie formulujú riešenia problémov.

Preklad programu

Bežne sa používajú 2 prístupy pri preklade programu z vyššieho programovacieho jazyka do nižšieho (napr. strojového kódu). Tieto spôsoby si priblížime príkladom.

1. spôsob pripomína preklad knihy z angličtiny do slovenčiny.
Prekladateľ sa dohodne s autorom, že preloží knihu z angličtiny do slovenčiny. Zoberie si výtlačok knihy v angličtine, preloží text do slovenčiny, preložený text odovzdá vydavateľstvu, vydavateľstvo knihu vydá, my si ju kúpime v kníhkupectve a prečítame v slovenčine. Ak rozumieme po slovensky, tak na pochopenie knihy nepotrebujeme anglický originál ani žiadnu inú pomôcku, často si ani neprečítame meno prekladateľa.

Spustíme program prekladač z vyššieho programovacieho jazyka, napr. Pascalu do strojového kódu, podhodíme mu ako vstup súbor s našim programom vo vyššom programovacom jazyku (zdrojový súbor), prekladač spracuje náš zdrojový súbor a vyprodukuje ekvivalentný program v strojovom kóde (cieľový súbor) zvyčajne v tvare súboru s príponou exe, ktorý potom môžeme veľakrát spustiť nezávisle na našom zdrojovom programe a prekladači. Takýto druh prekladača sa nazýva **kompilátor**.

2. spôsob pripomína simultánny preklad z angličtiny do slovenčiny.
Na medzinárodnej konferencii pán Black prednáša svoj príspevok v angličtine. Väčšina účastníkov nerozumie anglickému príspevku. Zavolali na pomoc tlmočníka. Pán Black povie vetu po anglicky a počká kým ju tlmočník povie po slovensky. Takto to ide až do konca prednášky. Tlmočník musí byť počas celej prednášky prítomný na konferencii. Po skončení príspevku pána Blacka si nebudú môcť účastníci prečítať príspevok po slovensky, lebo tlmočník im poskytol len slovný preklad, interpretoval príspevok pána Blacka postupne. Účastníci mali možnosť prednáške porozumieť.

Spustíme program prekladač a povieme mu, ktorý program vo vyššom programovacom jazyku má dať postupne príkaz za príkazom vykonať. Prekladač preloží do strojového kódu prvý príkaz, nechá ho počítač vykonať, potom preloží ďalší príkaz, zase ho nechá vykonať a takto to ide príkaz po príkaze až do konca programu. Počítač program pomocou prekladača vykonal, ale preklad sa neuchoval. Ak budeme chcieť, aby sa program znovu vykonal, musíme znovu pustiť prekladač a povedať mu, aký program má vykonať. Takémuto prekladaču sa hovorí **interpreter**.

Preklad a výpočet programu v Jave

Programy písané v Jave sa najprv prekladajú do takzvaného bajtkódu (bytecode) a potom je bajtkód interpretovaný pomocou programu, ktorému sa hovorí Java Virtual Machine alebo JVM. Pre všetky platformy je bajtkód rovnaký, JVM je naopak závislá na platforme (platformy sa líšia operačným systémom alebo strojovým kódom). To znamená, že programátor môže vyvíjať program na počítači s jednou platformou a spúšťať na počítači s inou platformou. Zdrojový text programu v jave má príponu Jáva (napr. mojprvy.java). Preložený bude mať príponu class (mojprvy.class). Ako písať, prekladať a spúšťať programy sa dozvieme o pár riadkov ďalej.

Preklad ani interpretácia sa nemusia podariť. Preklad sa nepodarí, ak prekladač nepozná preklad niektorých slov alebo jazykovú konštrukciu. Hovoríme tomu gramatické pravidlá alebo **syntax jazyka**. So syntaxou ste sa stretli na základnej škole, keď ste celé hodiny slovenčiny trávili nad vetnými skladmi. Príklady syntakticky nesprávnych slovenských viet:

Moja otec rád korčuľuje. Netlač mi kareláby do vlahy!

Javovské programy možno spúšťať ako akékoľvek iné aplikácie ale, a to prinieslo Jave jej široké použitie, aj v podobe apletov z web stránok. V apletoch sa z bezpečnostných dôvodov nemôžu používať všetky prostriedky Javy, aby sa chránil počítač, na ktorom sa aplet spúšťa. Spustiť aplet cez web stránku znamená, že prehliadač web stránok zavedie aplet zo zadaného serveru do pamäte nášho počítača a tam sa interpretuje pomocou JVM. Samozrejme, že aj počas interpretácie programu sa môže vyskytnúť chyba - **chyba počas behu programu** (run-time error). Od dobrého programátora sa očakáva, že situácie, ktoré by mohli viesť k chybám počas výpočtu v svojom kóde ošetrí.

(koniec prevzatého učiva)

Systémová jednotka

obsahuje **základnú dosku** (matičnú, mainboard, motherboard), ktorej úlohou je prepojiť pomocou zbernice jednotlivé časti počítača do fungujúceho celku a poskytnúť im elektrické napájanie. Väčšinu funkcií matičnej dosky zabezpečuje integrovaný obvod (zvyčajne dvojica) nazvaný čipová sada (čipset). V súvislosti s PC založenými na systémoch triedy Intel Pentium čipset často označuje dva hlavné čipy matičnej dosky: northbridge (severný most, systémový radič) a southbridge (južný most, vstupno-výstupný radič). Pomenovanie vzniklo z obvyklého umiestnenia týchto čipov na matičnej doske: northbridge sprostredkuje rozhranie medzi procesorom a operačnou pamäťou, a je umiestnený bližšie k procesoru (ktorý je obvykle umiestnený na hornej časti (na „severe“) matičnej dosky, ak je montovaná vertikálne, z dôvodu umiestnenia rozširujúcich konektorov („slotov“) v štandardnej skrinke typu minitower); southbridge obsahuje rozhranie k rozširujúcim konektorom a iným pomalším perifériám a obvykle je umiestnený na spodku dosky (na „juhu“). Severný most zabezpečuje „rýchlu“ komunikáciu medzi CPU, pamäťou RAM, AGP portom alebo PCI Express zbernicou a tiež zaisťuje spojenie s južným mostom. Niektoré severné mosty obsahujú integrované grafické karty. Južný most má v typickom počítači na starosti obsluhu pomalších zariadení (klávesnice, myši, diskov, audio, USB, PCI,...). **Zbernica** je sústava vodičov, fyzicky zabezpečujúca prepojenie jednotlivých častí počítača. Podľa určenia prenášaných údajov môže byť adresná, dátová a riadiaca. Je charakterizovaná napríklad rýchlosťou v MHz alebo šírkou zbernice, t.j. počtom paralelných vodičov a tomu zodpovedajúcemu počtu prenášaných bitov v danom okamihu (8, 16, 32, 64-bitová). Zbernica v northbridge sa nazýva aj systémová (system bus), prepája predovšetkým CPU a RAM, ale aj severný a južný most; zbernica v časti southbridge sa nazýva aj I/O bus.

Procesor je umiestený v pätici (socket) na základnej doske. Vykonáva inštrukcie programu uloženého v operačnej pamäti a riadi činnosť ostatných častí počítača. Skladá sa najmä z obrovského (milióny) počtu tranzistorov, v súčasnosti ide vývoj v procesoroch smerom k integrácii viacerých jadier (viacero procesorov) do jediného čipu.

Procesor charakterizuje najmä:

- taktovacia frekvencia
  udáva, koľkokrát je procesor schopný zmeniť stav z 0 na 1 alebo opačne za sekundu (v súčasnosti v GHz)
- frekvencia zbernice QPI
  (QuickPath Interconnect) udáva, akou frekvenciou je taktovaná zbernica medzi procesorom a severným mostom, donedávna zbernica FSB (Front Side Bus)
- šírka operanda
  je to počet bitov, ktoré je schopný procesor spracovať v jednom kroku (32, 64 bitov)
- cache procesora
  veľkosť a počet rýchlych vyrovnávacích pamätí slúžiacich na vyrovnanie rýchlosti medzi pomalou hlavnou pamäťou a rýchlym procesorom
- počet jadier.

Program pre procesor musí byť zapísaný v strojovom kóde (postupnosť čísel), príkazy ktorého nazývame strojové inštrukcie. Každý procesor má vlastnú inštrukčnú sadu (strojový kód). V niektorých strojových jazykoch má každá inštrukcia rovnaký počet bitov, zatiaľ čo v iných typoch sa môže počet bitov inštrukciu od inštrukcie líšiť. Niektoré inštrukcie vyžadujú operandy, ktoré upresňujú, ako sa inštrukcia má vykonať, prípadne s akými dátami má pracovať. Program uložený v pamäti počítača je sekvenciou inštrukcií strojového jazyka. Procesor jednotlivé inštrukcie sekvenčne načíta, dekóduje a vykoná.

Nasledujúce programy robia to isté (Informatika pre stredné školy, str.81):

- program v Pascale:

```
var A: array [1..10] of integer;
    i: integer;
begin
        for i:= 1 to 10 do A[i]:= i * 5;
end;
```

- program v jazyku Asembler:

```
asm
        lea eax, A
        mov ecx, 10
        mov edx, 5
@hop:
        mov [eax], edx
        add eax, TYPE Integer
        add edx, 5
        loop @hop
end;
```

- program v strojovom kóde (každému príkazu v Asembleri zodpovedá jedna strojová inštrukcia, ktorá môže zabrať jeden alebo aj niekoľko bajtov):

```
8d 05 20 08 44 00 b9 0a
00 00 00 ba 05 00 00 00
89 10 83 c0 04 83 c2 05 e2 f6
```

Inštrukčné sady procesorov

Vývoj architektúry súčasných univerzálnych procesorov sa rozvíja v dvoch základných smeroch, ktoré vyplývajú z definície inštrukčného súboru, ktorý vie procesor spracovať.

Architektúra **CISC** (Complex instruction set computer, počítač s rozsiahlou inštrukčnou sadou alebo počítač s úplnou inštrukčnou sadou) - vývoj klasických von neumannovských architektúr procesorov bol smerovaný do architektúr orientovaných na vyššie programovacie jazyky. Cieľom bol stav, kedy jazyk vysokej úrovne bude „priamo strojovým jazykom“ daného procesora a jeho inštrukčný súbor bude realizovať zložité a komplexné funkcie operačného systému. To viedlo k tvorbe stále zložitejších a väčších (komplexnejších) inštrukčných súborov s cieľom pokryť a podporovať čo najširšie spektrum aplikácií. Počet inštrukcií neustále stúpal a boli stále zložitejšie. Vzrastali problémy jednak s realizáciou inštrukcií na kremíkovom čipe a jednak s prekladom programu z vyššej úrovne do inštrukčného súboru. Architektúra CISC svojho času eliminovala aj nedostatky technických prostriedkov, ako napr. drahé a pomalé pamäte a zbernice s nízkou prenosovou rýchlosťou dát a inštrukcií, ktoré boli postupne prekonané. CISC procesory na konci 70. rokov minulého storočia boli už také zložité, že sa objavili prvé pokusy o celkové zjednodušenie štruktúry procesorov.

Architektúra **RISC** (Reduced instruction set computer, počítač s obmedzenou sadou inštrukcií) - smer, vedúci naopak k redukcia inštrukčného súboru procesora a k aplikácii jednoduchých inštrukcií. Vychádza z predpokladu, že na vykonanie cca 80% operácií programu postačuje cca 20% inštrukcií. Procesor má zabudované len základné mikroinštrukcie, ktoré sú jednoduchšie a ľahšie, preto rýchlejšie vykonateľné (bez mikrokódov, ktoré sa používa pri procesoroch CISC).

Jednoduché inštrukcie procesora RISC vyžadujú stále rovnakú postupnosť operácií:

1. zavedenie inštrukcie
2. dekódovanie

3. realizácia inštrukcie
4. zavedenie operandov z registrov
5. uloženie výsledkov do registra

Inštrukcie, ktoré nie sú v základnom inštrukčnom súbore sa jednoducho naprogramujú. Výhoda procesorov RISC, sú rýchlejšie ako CISC. RISC procesor predpokladá rýchlejšie zbernice, pamäte, cache atď. Operačné systémy pracujúce pod procesormi RISC sú väčšinou založené na filozofii operačného systému UNIX. Od firmy Intel sú to procesory Pentium a všetky vyššie rady. Pričom procesory Pentium využívajú architektúru RISC len vo vnútornom spracovávaní inštrukcií ale navonok sa javia ako procesory CISC. Je to z toho dôvodu, aby sa zachovala kompatibilita z predchádzajúcimi typmi procesorov firmy Intel, ktoré boli navrhované v architektúre CISC.

Architektúra procesora

Pod architektúrou procesora rozumieme základné usporiadanie jednotlivých funkčných častí procesora z hľadiska ich vzájomného vzťahu a úloh, ktoré plnia. Medzi význačné rysy architektúry súčasných procesorov patrí:

- prúdové spracovanie inštrukcií
- dynamické spracovanie inštrukcií
- cache pamäť

Prúdové spracovanie inštrukcií

Cieľom prúdového spracovania inštrukcií (pipelining) je urýchliť činnosť procesora tým, že sa inštrukcie vykonávajú paralelne. Vychádzame z toho, že spracovanie inštrukcie možno rozložiť na (spravidla) päť jednoduchších úkonov, ktoré na seba nadväzujú:

- prenos inštrukcie z pamäti do procesora (Instruction Fetch - IF)
- dekódovanie (Instruction Decode - ID) - inštrukcia sa konvertuje do jednoduchých povelov (mikrooperácií), ktoré výkonné jednotky procesora dokážu vykonať
- vykonanie (Execution - EX) - aj výpočet adresy
- výber operandu z pamäti (Data Access - DA)
- zápis výsledku do pamäti (Write Back - WB)

Každý úkon sa vykonáva v samostatnom funkčnom bloku, prípadne sa rozloží na ešte jednoduchšie operácie a vykonáva sa v niekoľkých funkčných blokoch. Funkčný blok (functional unit) je skupina logických obvodov, ktoré vykonávajú spoločnú prácu (napr. aritmeticko-logická jednotka). Pri klasickom spracovaní údajov by sa ďalšia inštrukcia začala spracovávať až po ukončení spracovania predchádzajúcej inštrukcie. Pri takomto spôsobe práce by funkčné bloky väčšinu času zaháľali a čakali na príchod ďalších údajov. Prúdové spracovanie údajov prebieha tak, že funkčné bloky sú zoradené logicky za sebou a tvoria kanál (pipe). Akonáhle spracovanie inštrukcie postúpi z prvého stupňa (funkčného bloku) do druhého, môže prísť ďalšia inštrukcia a vstúpiť do prvej fázy svojho spracovania (Tab. 1).

Tab.1	Hodinový takt									
	1	2	3	4	5	6	7	8	9	10
1. inštrukcia	prenos	dekód.	vykon.	operan.	zápis					
2. inštrukcia		prenos	dekód.	vykon.	operan.	zápis				
3. inštrukcia			prenos	dekód.	vykon.	operan.	zápis			
4. inštrukcia				prenos	dekód.	vykon.	operan.	zápis		
5. inštrukcia					prenos	dekód.	vykon.	operan.	zápis	

funkčné bloky

Tým sa dosiahne, že, až na niekoľko výnimiek, sa v každom hodinovom cykle ukončí jedna inštrukcia.
V niekoľkých prípadoch sa môže narušiť plynulosť inštrukčného toku. Ide najmä o tieto situácie (označujú sa pojmom *pipeline hazards*):

- **dátová závislosť** - vyskytuje sa, keď nasledujúca inštrukcia musí čakať na výsledok predchádzajúcej inštrukcie.
- **konflikty riadenia** - vyskytujú sa pri inštrukciách vetvenia, kedy sa môže prerušiť sekvenčné vykonávanie inštrukcií a preniesť riadenie na vzdialené miesto v programe. Vetviaca inštrukcia obsahuje nejakú logickú podmienku, ktorá sa vyhodnotí a vykoná sa skok na návestie, ak je podmienka splnená. Do pipelinu však

prichádzajú inštrukcie ešte predtým, ako sa podmienka vyhodnotí. Ak sa v okamihu vyhodnotenia podmienky zistí, že pipeline obsahuje inštrukcie z tej vetvy programu, ktorá sa nebude vykonávať, je potrebné pipeline vyprázdniť a načítať inštrukcie zo správnej vetvy.

- **štrukturálne konflikty** - vyskytujú sa v prípade, že viac inštrukcií súčasne požaduje rovnaký prostriedok procesora. Najčastejšie dochádza k sporom o zbernicu, či už internú alebo externú. V požiadavke o prostriedok dostáva prednosť tá inštrukcia, ktorá je viac rozpracovaná (je dlhšie v pipeline), zatiaľ čo druhá inštrukcia (a všetky, ktoré za ňou nasledujú v pipeline) musí čakať, kým sa prostriedok uvoľní.

Dátové a štrukturálne konflikty sa riešia zdržaním pipelinu (zaradením prázdneho taktu). Súvislý tok inštrukcií sa tým naruší a vznikajú v ňom bubliny, v ktorých pracuje len jeden funkčný blok, ostatné stoja.

Dynamické spracovanie inštrukcií

Dynamické spracovanie inštrukcií je súbor metód, ktorých cieľom je zvýšiť efektívnosť práce procesora tým, že sa predpovedá a mení poradie vykonávania inštrukcií. Zameriame sa na predpovedanie skokov.

Predpovedanie skokov zohráva svoju úlohu v prípade nesekvenčného vykonávania programu v dôsledku príkazu skoku. Pretože príkaz skoku prenáša riadenie do iného miesta programu, môže sa stať, že príslušné inštrukcie nie sú v okamihu potreby preložené a pripravené na vykonanie. Úlohou predpovedania skokov je na základe správania sa programu v minulosti predvídať, na akej inštrukcii bude vykonávanie programu pokračovať, aby sa ďalšie inštrukcie mohli vybrať z pamäti a dekódovať skôr, ako sa vykoná inštrukcia skoku. Táto snaha je opodstatnená, lebo skoky predstavujú 15 až 25 % všetkých vykonávaných inštrukcií.

Zdržaniu v práci procesora sa dá ľahšie zabrániť pri nepodmienených skokoch (príkaz goto). Nepodmienený skok nezávisí od výsledku predchádzajúcej operácie, preto vzápätí po jeho dekódovaní procesor vie, na ktorej inštrukcii bude vykonávanie programu pokračovať a výberová jednotka môže hneď priniesť inštrukciu z adresy, ktorá bola operandom inštrukcie nepodmieneného skoku.

Pri podmienených skokoch (if, case, for, while, repeat) je situácia zložitejšia, pretože až v okamihu vyhodnotenia podmienky skoku procesor vie, na ktorej inštrukcii má pokračovať. Môže však predvídať, ako sa podmienka skoku vyhodnotí. Vychádza sa pritom z predpokladu, že obidve vetvy podmieneného skoku nemajú rovnakú pravdepodobnosť vykonania. Ak sa podmienený skok opakuje, je viac pravdepodobné, že sa jeho podmienka vyhodnotí rovnako ako v predchádzajúcom prípade. Táto metóda predpovedania skokov sa nazýva dynamická predikcia. Predikčná logika si v tabuľke histórie skokov (Branch History Table - BHT) pamätá niekoľko desiatok až stoviek posledných skokových inštrukcií, presnejšie vývoj, čiže udalosti, ktoré pri nich nastali (či sa skok vykonal, alebo nie). Veľkosť BHT v Pentiu 4 je 4096 položiek, čo pokrýva väčšinu skokov v programe. Pri takejto veľkosti tabuľky je presnosť predikcie 82 až 99 %.

Vykonávanie inštrukcií mimo poradia je možné vďaka tzv. **superskalárnej** architektúre. Superskalárna architektúra znamená, že procesor má niekoľko výkonných jednotiek, ktoré môžu pracovať súčasne. Napríklad to môžu byť aritmeticko-logické jednotky (ALU) pracujúce na dvojnásobnej frekvencii; aritmetická jednotka pre celočíselné operácie; jednotka pre spracovanie čísiel v pohyblivej rádovej čiarke, t.j. reálnych čísel.

Cache pamäť

Cache pamäť je rýchla pamäť, ktorá slúži ako vyrovnávacia pamäť medzi rýchlym procesorom a pomalou hlavnou pamäťou. Jej efektívnosť je, okrem rýchlosti prístupu, daná malým rozsahom a odlišnou správou uložených údajov. Pre programátora je táto pamäť neprístupná. O jej obsahu rozhoduje hardware. Stratégia presúvania údajov medzi hlavnou pamäťou a cache je založená na štatisticky overených vlastnostiach tzv. časovej a miestnej lokality.

Časová lokalita sa zakladá na pravdepodobnosti, že adresa, ktorá bola práve teraz požadovaná, bude čoskoro požadovaná znova. Preto sa každý údaj, ktorý sa číta z hlavnej pamäti, automaticky ukladá do cache.

Miestna lokalita znamená, že program v krátkom časovom úseku pristupuje k susedným pamäťovým miestam. Preto sa pri čítaní z hlavnej pamäti presúva do cache blok obsahujúci niekoľko susedných údajov.

Z technologických a konštrukčných dôvodov sa rozlišuje cache viacerých úrovní: 1. úrovne (L1), cache 2. úrovne (L2),...

Zdroj: Wikipédia, internet, Kalaš a kol.: Informatika pre stredné školy (učebnica)