Strojový kód

- **Strojový kód** (*Machine code*) je program vyjádřený v počítači jako posloupnost instrukcí procesoru (posloupnost bajtů, resp. bitů).
  - Z hlediska uživatele je strojový kód nesrozumitelný, z hlediska systému je přímo proveditelný

- **Strojový kód procesoru je tedy množina všech strojových instrukcí, které je procesor schopen vykonávat.**
  - Kódování závisí na typu procesoru

- **Cvičný strojový kód:**

LNA x	x → S	**ADD x**	<S> + <x> → S
LNA x [IND]	x + <IND> → S	**SUB x**	<S> - <x> → S
LDA x	<x> → S	**MLP x**	<S> . <x> → S
LDA x [IND]	<x + <IND>> → S	**AND x**	<S> AND <x> → S
LDI x	<<x>> → S	**OR x**	<S> OR <x> → S
LDI x [IND]	<<x + <IND>> → S	**XOR x**	<S> XOR <x> → S
STA x	<S> → x	**MOD x, y**	<x> MOD <y> → S
STA x [IND]	<S> → x + <IND>	**DIV x, y**	<x> DIV <y> → S
STI	<S> → IND	**SHR R, n**	posun <R> o n bitů vpravo
JMP x	x → IP	**SHL R, n**	posun <R> o n bitů vlevo
JZ x	je-li <S> = 0, pak x → IP	**RSHR R, n**	rotace <R> o n bitů vpravo
JP x	je-li <S> > 0, pak x → IP	**RSHL R, n**	rotace <R> o n bitů vlevo
JN x	je-li <S> < 0, pak x → IP		
LOOP x	<IND> - 1 → IND, je-li <IND> > 0, pak x → IP		
HLT	zastavení procesoru		
JSR x	<SP> - 1 → SP, <IP> → <SP>, x → IP	**PUSH x**	<SP> - 1 → SP, <x> → <SP>
RET	<<SP>> → IP, <SP> + 1 → SP	**POP x**	<<SP>> → x, <SP> + 1 → SP

Instrukce počítače

- **Formát strojové instrukce**
  - **operační kód** – určuje, jaká instrukce se bude vykonávat
  - **adresa vstupního operandu** – identifikuje operand(y), se kterým bude instrukce pracovat
  - **adresa výstupního operandu** – stanoví adresu(y), na kterou bude po provedení instrukce uložen výsledek
  - **adresa následně vykonané instrukce** – sdělí procesoru, jakou instrukci má vykonávat jako následující
    - pokud není uvedena, procesor začne vykonávat bezprostředně následující instrukci

- U různých typů počítačů je **různý instrukční kód** a instrukce mohou mít **různý počet operandů** (obvykle 0,1,2,3).

Činnost procesoru při provádění programu

- Veškeré **programy jsou převedeny do strojového kódu** (sekvence strojových instrukcí)

- Po spuštění programu procesor postupně zpracovává instrukce
  - adresa zpracovávané instrukce je v **čítači instrukcí** (IP)

- Každá **instrukce je zpracována ve dvou fázích**
  - **výběrová fáze**
    - řadič přenese obsah čítače instrukcí (IP) na adresní sběrnici a vydá příkaz ke čtení z hlavní paměti (HP)
    - paměť přenese obsah adresované buňky na datovou sběrnici
    - řadič uloží instrukci do registru instrukcí (RI)
    - obsah čítače instrukcí se zvýší o 1
  - **prováděcí fáze**
    - instrukce v RI je řadičem dekódována
    - je-li nastaven příznak modifikace indexregistrem (IND), je k operandu přičten obsah indexregistru
    - řadič naadresuje HP a přečte obsah vstupních operandů
    - v součinnosti s ALU je instrukce provedena a výsledek uložen do střadače
    - řadič zajistí přenos výsledku ze střadače na adresu výstupního operandu

Vstupní a výstupní operandy

- Vstupní a výstupní **operand může být umístěn** v
  - **hlavní paměti** – jednoznačně určen adresou
  - **procesoru** – procesor má vlastní registry (paměťová místa), do kterých lze ukládat

- **Způsob adresace = způsob, jakým bude operand instrukce na základně obsahu procesorem identifikován**

- **Dále použité symboly:**
  - **X** operand
  - **S** střadač
  - **<…>** obsah operandu, adresy, …

Způsoby adresace

- **bezprostřední** **X → S**
  - *hodnota operandu přímo do S*
  - operand je procesoru k dispozici okamžitě po načtení instrukce
  - není třeba žádného dalšího čtení z paměti nebo registru

  - **Příklad:**
    - **X = 100**
    - do **S** se uloží hodnota **100**

- **přímá** <X> → S
  - *obsah adresy X se uloží do S*
  - hodnota v poli operandu je interpretována jako adresa hlavní paměti, na které je operand uložen

  - **Příklad:**
    - **X = 100**
    - do **S** se uloží **obsah adresy 100**, tj. **102**

adresa	obsah
100	102
101	40
102	101

- **nepřímá** <<X>> → S
  - *obsah obsahu adresy X se uloží do S*
  - hodnota v poli operandu je interpretována jako adresa hlavní paměti, na které je teprve uložena adresa operandu

  - **Příklad:**
    - **X = 100**
    - do **S** se uloží **obsah obsahu adresy 100**, tj. **101**

- **s posuvem** <X + <R>> → S
  - *k obsahu R se přičte X, obsah výsledné adresy se uloží do S*
  - Adresaci s posuvem lze použít několika různými způsoby:
    - **relativní adresace**
      – R = čítač instrukcí, X = posuv ±
    - **bázová (segmentová) adresace**
      – R = základní adresa v HP (báze, segment), X = kladný posuv
    - **indexová adresace**
      – X = určitá adresa v HP, R = kladný posuv (IND)

  - **Příklad:**
    - **X = 100, IND = 1**
    - do **S** se uloží **obsah adresy 101**, tj. **40**

- Pomocí **indexové** a **přímé** adresace lze vytvořit **nepřímou** adresaci
  - **<<X>> → S ≡ <X> → S ; S → IND ; <0> IND → S**

Typy strojových instrukcí

- **instrukce pro přenos dat** — **LNA, LDA, LDI, STA, STI**
  - přesouvají data mezi paměťovými místy, která se nacházejí v HP, registrech nebo zásobníku.
- **aritmetické instrukce** — **ADD, SUB, MLP, MOD, DIV**
  - instrukce pro základní aritmetické operace s čísly v pevné a pohyblivé řádové čárce (sčítání, odčítání, násobení a dělení)
- **logické instrukce** — **AND, OR, XOR**
  - logické instrukce realizují booleovské funkce na stejnolehlých bitech operandů.
- **instrukce posuvu a rotace** — **SHR, SHL, RSHR, RSHL**
  - posouvají obsah paměťového místa o v instrukce stanovený počet vlevo nebo vpravo
- **instrukce (ne)podmíněného skoku** — **JMP, JZ, JP, JZ**
  - instrukce obsahuje na místě operandu adresu, na kterou je proveden skok při splnění určité podmínky
- **instrukce pro podporu cyklů** — **LOOP**
  - instrukce, které sníží nebo zvýší obsah indexregistru a provedou skok na adresu uvedenou v poli operandu, pokud obsah indexregistr není nulový.
- **instrukce pro podporu podprogramů** — **JSR, RET**
  - důležité instrukce, které dovolují změnit právě vykonávanou sekvenci instrukcí za sekvenci jinou.
- **instrukce operace se zásobníkem** — **PUSH, POP**
  - instrukce pro čtení a ukládání dat na zásobník (speciální datová struktura, ze které jsou naposledy uložená data vybírána jako první – strategie LIFO, tj. *Last In First Out*)
- **instrukce pro I/O operace**
  - slouží pro komunikaci s přídavnými zařízeními.
- **instrukce obsluhy přerušení** — **INT, RETI**
  - slouží pro generování vnitřního přerušení a návrat z obslužné subrutiny přerušení
- **systémové instrukce** — **HLT**
  - speciální instrukce používané operačním systémem

- Některé instrukce lze provádět jen v systémovém stavu → **privilegované instrukce**
  - smí je užít **jen operační systém**, ne uživatelský program
  - pokus o provedení v uživatelském stavu vyvolá **přerušení**

Logické instrukce

- Logické instrukce realizují booleovské operace na stejnolehlých bitech obou operandů.

- Strojové instrukce obvykle realizují booleovské operace **AND** (logický součin), **OR** (logický součet), **EQ** (ekvivalence, stejnost), **XOR** (vylučující nebo, nestejnost) a **NON** (negace, opak, doplněk) .

P	Q	NON P $\neg P$	P AND Q $P \wedge Q$ $(\cdot)$	P OR Q $P \vee Q$ $(+)$	P XOR Q $P \oplus Q$	P EQ Q $P \Leftrightarrow Q$
0	**0**	**1**	**0**	**0**	**0**	**1**
0	**1**	**1**	**0**	**1**	**1**	**0**
1	**0**	**0**	**0**	**1**	**1**	**0**
1	**1**	**0**	**1**	**1**	**0**	**1**

- **Obecná pravidla bitových operací:**

x AND $\mathbf{0} \rightarrow \mathbf{0}$ x OR $\mathbf{0} \rightarrow x$ x XOR $\mathbf{0} \rightarrow x$

x AND $\mathbf{1} \rightarrow x$ x OR $\mathbf{1} \rightarrow \mathbf{1}$ x XOR $\mathbf{1} \rightarrow \neg x$

x AND $x \rightarrow x$ x OR $x \rightarrow x$ x XOR $x \rightarrow \mathbf{0}$

- **Příklad:**

  - **X = $(10101010)_2$**   **M = $(00001111)_2$ - „maska“**
    - **Maska** znamená nastavení hodnoty pomocného registru (**M**) nebo paměťového místa tak, aby po provedení vybrané logické instrukce s registrem **X** a maskou **M** došlo k požadované změně registru **X** podle obecných pravidel bitových operací.

  - **X AND M** = **$(00001010)_2$**
    - První čtyři bity vynulovány, ostatní zůstanou zachovány.

  - **X OR M** = **$(10101111)_2$**
    - První čtyři bity zůstanou zachovány a ostatní nastaveny na jedničku.

  - **X XOR M** = **$(10100101)_2$**
    - První čtyři bity zůstanou zachovány a ostatní budou znegovány (nastaveny na opačnou hodnotu).

  - **X XOR X** = **$(00000000)_2$**
    - Použití operace XOR na tentýž registr (vždy na bity se stejnou hodnotou) způsobí vynulování celého registru.
    - Vynulování registru pomocí instrukce XOR potřebuje ke svému provedení méně strojových cyklů než přiřazení hodnoty 0 do registru.

Přerušení

- **Přerušení** je signál, který způsobí změnu stavu systému, tj. přechod z uživatelského do systémového stavu (módu jádra).
  - Vznikne **kdykoliv nastane nějaká událost**.

- Přerušení se uplatňují podle **priorit** = úroveň „důležitosti“ přerušení.
  - V případě, že nastane více událostí najednou, je nejprve obslouženo přerušení s nejvyšší prioritou
  - Vykonávání obsluhy přerušení je možné přerušit pouze událostí s vyšší prioritou, než je priorita právě obsluhovaného přerušení

- **Událost** (*event*) = cokoliv, co má vliv na řízení práce výpočetního systému.

- **Přerušení** (události) lze **rozdělit** na:

  - **Vnitřní** (program, HW) – událost vznikající uvnitř procesoru - *trap*
  - **Vnější** (operátor, prostředí) – vzniká vně procesoru - *interupt*

  - **Očekávané** (mají nastat), např. zařízení “splnilo úkol”
  - **Neočekávané** (poruchy)

  - **Synchronní** (vyvolá instrukce)
  - **Asynchronní** (nezávisí na instrukci)
    - Při asynchronním přerušení se vždy prováděná instrukce nejprve dokončí celá.

  - **Maskovatelné** – lze mu zabránit nastavením k tomu určeného registru procesoru, tzv. **masku přerušení**
  - **Nemaskovatelné** – události, které nelze maskovat. Obvykle se jedná o chybnou činnost počítače

- **Obsluha přerušení** - počítač zahájí práce na programu operačního systému
  - zjistí příčinu přerušení
  - uloží stavové slovo současného procesu a obsah čítače instrukcí na zásobník do hlavní paměti
  - na základě tzv. **vektoru přerušení** najde v **tabulce vektorů přerušení** odpovídající subrutinu
    - při vnitřním přerušení součást instrukce INT
    - při vnějším přerušení vloží vektor přerušení na datovou sběrnici zařízení, které žádá o přerušení
  - provede příslušnou subrutinu ošetření přerušení
  - obnoví obsahy pracovních registrů – instrukce RETI
    - obnova stavového slova procesoru
    - obnova čítače instrukcí
  - rozhodne zda opětovně spustí přerušený proces nebo zda zařadí jiný

- K ovládání přerušovacího systému je ve strojovém kódu k dispozici **instrukce**
  - **INT x** – generování vnitřního přerušení, **x** je interpretován jako vektor přerušení
  - **RETI** – ukončení vykonávání subrutiny ošetření přerušení