Generický procesor

a)účel, charakteristika, struktura a princip činnosti
b)formát instrukce, instrukční soubor, adresové módy
c)fáze vykonání instrukce, provedení instrukcí, zápis v registrové notaci

Architektura PC – obecně

Je globální pohled na všechny podstatné vlastnosti PC. Je to souhrnný přehled množiny registrů, paměti, instrukčního souboru, datových formátů a adresových módů.

Pohled z hlediska programátora strojového kódu

Poprvé tento popis použil, Gene Ambdahi (hlavní architekt OS 360). Pohled spočívá v rozdělení stroje do 4 základních rovin:

Struktura

propojení jednotlivých funkčních bloků

(šířka sběrnice, co je k čemu připojeno, např. sdílení dat s periferními zařízeními pomocí společného adresového prostoru, nebo jen speciální instrukce)

Organizace

dynamické implementace jednotlivých bloků.

(jak jednotlivé části fungují, např. jak pracuje matematický koprocesor s daty)

Implementace

návrh a obvodová realizace

(nevím, domyslet…)

Funkce

popis stroje funkčního celku

(zajímá nás pouze výsledek stroje nikoli, jak s daty pracuje, např. soudobé procesory x86 vydávají stejné výsledky, ačkoli jsou třeba různé značky.)

Obecný model PC

Paměť

Pasivní zařízení sloužící k uložení dat

Procesor

Jest aktivní prvek, který je schopen interpretovat program a v jako jediné části stroje je zde vznik nové informace

(když sečtu dvě čísla instrukcí ADD, vznikne součet, což je nová informace)

Propojení

Přenáší data, mění umístění dat, nikoli propojení¨

(tedy jinak, paket si počítač klidně pošle, ale kabel si do jiného počítače sám nepřipojí)

Transduktor

Mění reprezentace dat, jiný způsob kódování

(neboli převodník dat z počítače do lidské formy, např. monitor, tiskárna)

Dále je nutné definovat instrukční soubor.

Uvedené skutečnosti definují skalární počítač: stroj, který postupně (sekvenčně) zpracovává instrukce

Základní funkční bloky PC

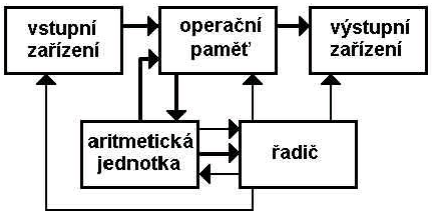
1. Paměť
2. ALU
3. Zařízení vstupu a výstupu

(někdy členěno na dvě části)

4. Řídící jednotka používá dvoustavovou logikou (binární), reprezentovanou elektrickým napětím

(neboli řadič)

(Vychází z von Neumannova schématu)



Několik základních definic generického (obecného) procesoru

Základní fakta funkčních bloků

1)

Adresa

Je označení místa uložení instrukce, nebo dat v paměti stroje. Je to pořadové číslo, udávající pozici adresované jednotky od lineárního bloku paměti.

Instrukce

Je kódovaný příkaz, který způsobí, že stroj udělá určitou činnost.

Instrukční cyklus

Je vykonání jedné instrukce, které je složeno z určitých elementárních úkonů.

Hodinový cyklu

Je časový interval mezi dvěma následujícími čely hodinového impulzu – je to nejmenší rozlišitelná časová jednotka instrukčního souboru

Strojový jazyk

Je úplný soubor strojových instrukcí (instrukční sada, nebo instrukční soubor)

Počítač

Je stavový stroj, který pracuje synchronně, činnost je řízena hodinovým signálem

Literál

Je datová položka, která nemá svou vlastní identifikaci. Je to lexikální hodnota, která přímo reprezentuje hodnotu

Adresace

Určuje, jak bude k datům, nebo instrukcím přistupováno

Efektivní adresa

Adresa spočítaná v době vykonání instrukce

Registr

Místo pro dočasné uložení dat

K vykonání instrukce jsou potřeba minimálně 3 kroky:

1) Vytažení
2) Dekódování
3) Provedení

A jsou-li data v paměti, nebo má-li, být výsledek v paměti přibudou další 2 kroky

4) Vytáhnout data
5) Uložit výsledná data

Pro adresaci instrukcí se používá relativní adresace.

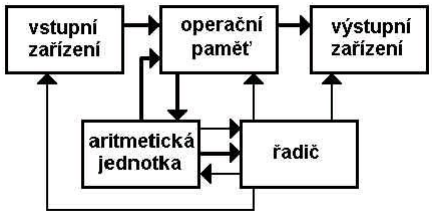
U procesoru 8086 máme segmentový registr a registr IP, který po spočtení ukazuje na instrukci, stačí aby procesor (v případě 8086) registr IP a ukazuje na další instrukci

Instrukce dělíme podle:

1. Dostupnosti
    1. Instrukce privilegované – nejsou dostupné každému programu

    2. Instrukce neprivilegované – jsou dostupné každému programu
2. Počtu operandů
    1. Monadické instrukce – např. nulování, inkrementace, dekrementace, negace, rotace, posuny
    2. Dyadické instrukce – aritmetické a logické operace
3. Účelu
    1. Větvení, skoky a cykly
    2. Řízení chodu programů
    3. Vstupy a výstupy
    4. Řízení stroje
    5. Přesun dat (50% veškerých operací)
    6. Aritmeticko-logické instrukce

Von Neumannova koncepce



Předpokládá, že počítač je sestaven z následujících komponent:

- Vstupní zařízení
- Výstupní zařízení
- Operační paměť - pro instrukce i data společná !
- Aritmeticko-logická jednotka
- Řadič

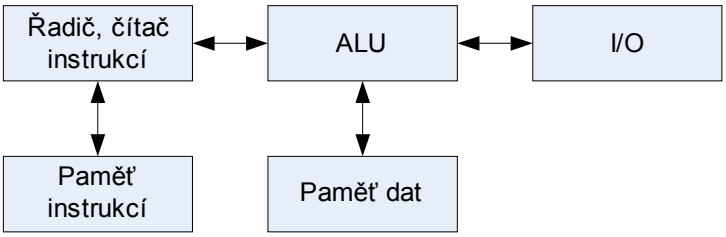
Von Neumann stanovil základní kritéria pro Von Neumannův počítač:

1. Počítač se skládá z vstupních/výstupních zařízení, operační paměti, ALU a řadiče
2. Struktura počítače je nezávislá na typu řešené úlohy, počítač se programuje obsahem paměti.
3. Následující krok počítače je závislý na kroku předchozím
4. Instrukce a data (operandy) jsou ve stejné paměti
5. Paměť je rozdělena do buněk stejné velikosti, jejichž pořadová čísla se využívají jako adresy.
6. Program je tvořen posloupností instrukcí, ty se vykonávají jednotlivě v pořadí, v jakém jsou zapsány do paměti.
7. Změna pořadí provádění instrukcí se provede instrukcí podmíněného nebo nepodmíněného skoku.
8. Pro reprezentaci instrukcí, čísel, adres a znaků, se používá dvojková číselná soustava.

Von Neumannova koncepce sebou přináší:

- Výhody
    - Rozdělení paměti pro kód a data určuje programátor
    - Řídící jednotka přistupuje k datům i instrukcím jednotným způsobem
    - Jedna sběrnice-jednodušší výroba
- Nevýhody
    - Společné uložení dat a kódu může vést při chybě k přepsání vlastního kódu programu
    - Jediná sběrnice tvoří až příliš moc úzké místo

Harvardská koncepce počítače:



Předpokládá, že počítač je složen z následujících bloků

4. Řadič – čítač instrukcí
5. ALU – aritmeticko-logická jednotka
6. I/O – vstup/výstup
7. Paměť pro instrukce
8. Paměť pro data

Harvardská koncepce je odvozená od von Neumannovy koncepce, s tím, že má oddělené instrukce (programový kód) od dat.

Instrukce – počty operandů

Instrukce musí vědět:

1. Co se má udělat (definuje instrukce)
2. S čím se to má dělat (operandy)
3. Kam se má uložit výsledek (také operandy)
4. Kde je následující instrukce

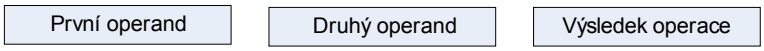
Každá instrukce má operační kód, který specifikuje co daná instrukce provádí.

4 - adresové instrukce

- Obsahuje všechny čtyři části
- Používali se u strojů s magnetickým bubnem
- Dnes občas interně užívány v některých strukturách procesorů

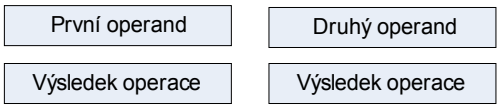
3 – adresové instrukce

- Programový kód je nejkratší, ale vykonání instrukce je náročnější (dlouhá instrukce, protože obsahuje 3 operandy/adresy operandů, opakované čtení z paměti kvůli více operandům)
- Neobsahují pole adresy následující instrukce, to je nepřímo určeno pomocí registru programového čítače (v 8086 Instruction Pointer), který je modifikován prováděnou instrukcí (délkou nebo adresou – skoková/neskoková), tak aby ukazoval na následující instrukce



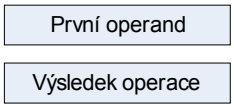
2 – adresové instrukce

- Nepřímo vyjádřená adresa výsledku
- Předpokládá, že výsledek bude uložen na místo jednoho z operandů (závisí na typu instrukce, tzn., může to být první operand, stejně tak i druhý operand)
- Dnes nejpoužívanější typ instrukce
- Např. ADD op1, op2; DIV op1,op2 …



1 – adresové instrukce

- Předpokládá použití ACC (=akumulátor), což je speciální registr, kde je jeden z operandů do kterého bude uložen výsledek prováděná operace.
- Nebo hodnota uložená v ACC je adresou na operandu
- Výhodou strojů s ACC a zásobníkově orientovaných je, že výsledek zůstává v procesoru a může být opakovaně použit pro další operaci, bez nutnosti čtení, či zápisu do operační paměti.



Bezadresové instrukce

- Zásobníkově orientované stroje, které nepotřebují adresy
- Výsledky i operandy jsou uloženy v zásobníku
- Instrukce naplnění zásobníku (push) a vytažení obsahu zásobníku však mají adresu
- Tyto typy strojů jsou vhodné jen pro určité typy výpočtů
- Operační kód je nejdelší, ale vykonání instrukce je nerychlejší (není třeba vypočítat adresy operandů)

Instrukce – adresace operandů

Bezprostřední adresace - instrukce pro práci s literálem

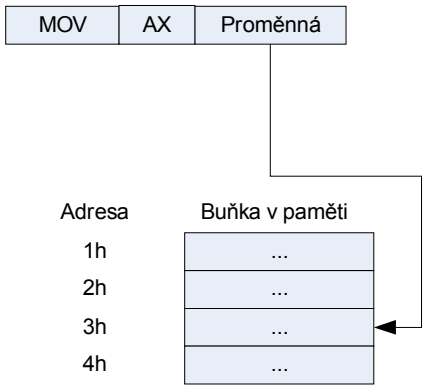
- Instrukce obsahuje operand
- Data jsou přímo součástí instrukce
- Uložené hodnoty jsou neměnné, proto se nazývají instrukce pro práci s literálem

```
mov ax,10
add ax,10
```

Přímá adresace

- Instrukce obsahuje adresu operandu – tato adresa je efektivní adresa

```
mov ax, proměnná
add ax, proměnná
```

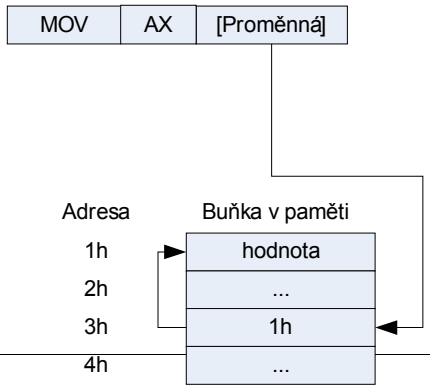


Předpokládáme-li, že existuje Proměnná, tak právě její název nám označuje místo, kde se nachází operand

Nepřímá adresace

- Instrukce obsahuje adresu, na které je adresa efektivní

```
mov ax,[proměnná]
add ax,[proměnná]
```

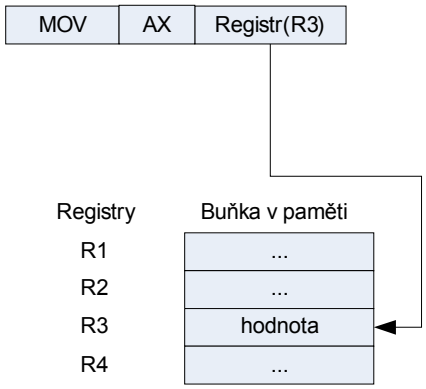


Opět, předpokládáme-li, že existuje proměnná, tak hodnota (která se nachází na efektivní adrese) je na adrese která je uložená na adrese proměnné

Registrová přímá adresace

- Instrukce obsahuje číslo (malou adresu) registru, který obsahuje operand
- Velmi efektivní instrukce – doba vykonání je zvláště krátká

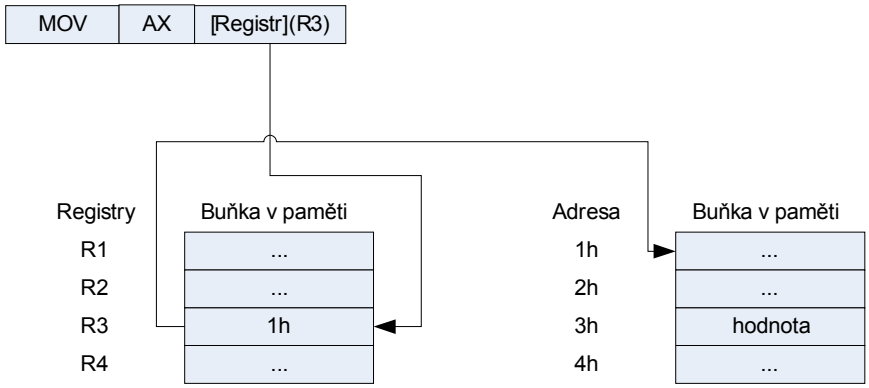
```
mov ax,bx;
add ax,bx;
```



Registrová nepřímá adresace

- Instrukce obsahuje číslo registru (malou adresu) ve kterém je uložena adresa operandu
- Tento adresový mód je zvláště efektivní, pokud chceme adresu operandu plynule měnit, čehož lze dosáhnout plynulým přičítáním/odečítáním k registru, ve kterém je uložena adresa

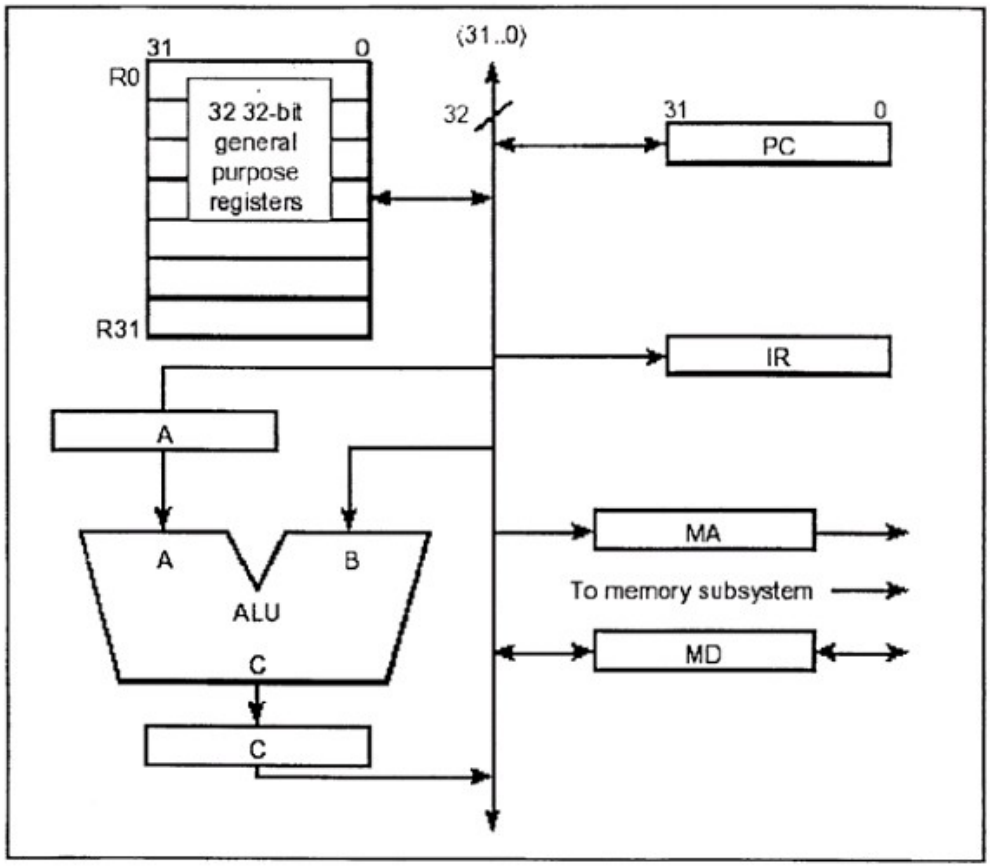
```
mov ax, [bx];
add ax, [bx];
```



Registrová nepřímá adresace s posuvem

- Instrukce obsahuje číslo registru, ve kterém je uložena adresa a posunu
- Efektivní adresa operandu se získá sečtením těchto dvou hodnot
- Posun může být konstanta, nebo číslo registru (proměnná hodnota uložená v registru)

PROCESORY.



1. Soubor 32 třiadvaceti bitových registrů
2. ALU = Aritmeticko-logická jednotka
3. Jednotka realizace podmíněných skoků
4. Dekódování instrukce a extrakce konstant - IR
    a. Probíhá dekódování instrukce
    b. Zde se uchovává právě prováděná instrukce
    c. Extrahují se konstanty
5. Rozhraní pro styk s operační pamětí
    a. MA = Memory Adress – zajišťuje přístup k operační paměti – podá operační paměti požadavek na adresu a MD data přijme
    b. MD = Memory Data – přijímá data z operační paměti
6. Čítač posuvů

Vykonání instrukce neproběhne v jednom kroku, ale jako soubor elementárních úkonů. Základem je cyklus vytáhni-vykonej (fetch-execute)

1. vytáhni instrukci z paměti
2. zjisti, co se má vykonat
3. vykonej, co se žádá
4. přejdi zpět ke kroku 1

Aritmetické a logické instrukce

- Pomocí této skupiny instrukcí vzniká v procesoru nová informace
- Aritmetické operace se provádí s operandy numerického charakteru
- Logické operace se provádí s operandy nenumerického charakteru (práce se symboly, zpracovávání grafiky)
- V této skupině jsou monadické i dyadické instrukce

```
ADD ra, rb, rc
;sečti čísla v registrech rb a rc a nahraj výsledek do registru ra
```

Krok	Registrová notace	Popis
T_0	MA ← PC; PC+4	Nahraj hodnotu programového čítače (PC) do MA; Sečti PC a 4 a nahraj do dočasného čítače
T_1	MD ← M[MA]; PC ← C	Nahraj instrukci z paměti z adresy, která byla v MA do jednotky pro styk s prostředím MD(=memory data)
T_2	IR ← MD	Nahraj právě prováděnou instrukci do IR z MD
T_3	A = R[rb]	Do akumulátoru A nahraj hodnou v registru rb
T_4	C = A+R[rc]	Sečti hodnotu v akumulátoru A a registru rc
T_5	R[ra] ←C	Nahraj výsledek z registru ALU do registru ra

```
ADDI rA, rB, K
; sečti čísla rb a K (konstanta) a nahraj výsledek do rA
```

Krok	Registrová notace	Popis
T_0	MA ← PC; PC+4	Nahraj hodnotu programového čítače (PC) do MA; Sečti PC a 4 a nahraj do dočasného čítače
T_1	MD ← M[MA]; PC ← C	Nahraj instrukci z paměti z adresy, která byla v MA do jednotky pro styk s prostředím MD(=memory data)
T_2	IR ← MD	Nahraj právě prováděnou instrukci do IR z MD
T_3	A = R[rb]	Do akumulátoru A nahraj hodnotu v registru rb
T_4	C = A+K	Sečti hodnotu v akumulátoru A a konstanty K
T_5	R[ra] ←C	Nahraj výsledek z registru ALU do registru ra

```
NEG rA, rC
; provede dvojkový doplněk na čísle rC a výsledek uloží do
; registru rA
```

Krok	Registrová notace	Popis
T_0	MA ← PC; PC+4	Nahraj hodnotu programového čítače (PC) do MA; Sečti PC a 4 a nahraj do dočasného čítače
T_1	MD ← M[MA]; PC ← C	Nahraj instrukci z paměti z adresy, která byla v MA do jednotky pro styk s prostředím MD(=memory data)
T_2	IR ← MD	Nahraj právě prováděnou instrukci do IR z MD
T_3	A = R[rc]	Nahraj hodnotu v registru rc do akumulátoru
T_4	C = ~A	Proveď operaci dvojkového doplňku a ulož výsledek do registru v ALU C
T_4	R[ra] ← C	Přesuň obsah registru v ALU C do registru ra

```
SHR rA, rB, P
; provede posun do leva (shift left)na čísle rB o P bitů a
; výsledek uloží do registru rA
```

Krok	Registrová notace	Popis
T_0	MA ← PC; PC+4	Nahraj hodnotu programového čítače (PC) do MA; Sečti PC a 4 a nahraj do dočasného čítače
T_1	MD ← M[MA]; PC ← C	Nahraj instrukci z paměti z adresy, která byla v MA do jednotky pro styk s prostředím MD(=memory data)
T_2	IR ← MD	Nahraj právě prováděnou instrukci do IR z MD
T_3	n←P	Do jednotky pro posuvy nahraj hodnotu operandu P (kolikrát se má posuv opakovat)
T_4	n=0→ (n←P)	Pokud n obsahuje hodnotu 0, tak nahraj do n hodnotu P
T_5	C←R[rB]	Provede se jeden posun
T_6	Opakuj krok T_5 n-krát	
T_7	R[ra] ← C	Přesuň obsah registru v ALU C do registru ra

Instrukce řízení chodu programů

- Slouží k měnění posloupnosti vykonávaných příkazů
- Žádná nová informace nevzniká
- Podle provedení je dělíme na instrukce
  - Bez návratu (podmíněné a nepodmíněné skoky)
  - S návratem (volání podprogramů) – předpokladem je zapamatování návratové adresy (adresy instrukce za skokem) a nějaký mechanismus (instrukce návratu), který obnoví původní stav
- Instrukce nepodmíněného skoku předá řízení jiné části programového kódu (je to podmíněný skok, jehož podmínka je splněna vždy)
- Skupina instrukcí podmíněných potřebuje k rozhodnutí, zda se skok provede splnění určité podmínky

```
BR rB, rC, P
; provede posun do leva (shift left)na čísle rB o P bitů a
; výsledek uloží do registru rA
```

Krok	Registrová notace	Popis
T_0	MA ← PC; PC+4	Nahraj hodnotu programového čítače (PC) do MA; Sečti PC a 4 a nahraj do dočasného čítače
T_1	MD ← M[MA]; PC ← C	Nahraj instrukci z paměti z adresy, která byla v MA do jednotky pro styk s prostředím MD(=memory data)
T_2	IR ← MD	Nahraj právě prováděnou instrukci do IR z MD
T_3	CON ← cond(R[rc])	
T_4	CON→ (PC←R[rb])	

```
BRL rA, rB, rC, P
; provede posun do leva (shift left)na čísle rB o P bitů a
; výsledek uloží do registru rA
```

Krok	Registrová notace	Popis
T_0	MA ← PC; PC+4	Nahraj hodnotu programového čítače (PC) do MA; Sečti PC a 4 a nahraj do dočasného čítače
T_1	MD ← M[MA]; PC ← C	Nahraj instrukci z paměti z adresy, která byla v MA do jednotky pro styk s prostředím MD(=memory data)
T_2	IR ← MD	Nahraj právě prováděnou instrukci do IR z MD
T_3	R[ra] ←PC	

T_4	CON ← cond(R[rc])	
T_5	CON→ (PC←R[rb])	

P={0..5}	Popis podmínky	Mnemonika	
000	Neskončí nikdy	brnv	brlnv
001	Skončí vždy	br	brl
010	Skončí, když R[rc]=0	brzr	brlzr
011	Skončí, když R[rc]≠0	Brnz	brlnz
100	Skončí, když R[rC]≥0	brpl	brlpl
101	Skončí, když R[rC]≤0	brmi	brlmi

Instrukce přesunu dat

- Zajišťují přesun operandů (data, nebo adresy)
- Žádná nová informace nevzniká
- Základem je instrukce LOAD – přemístění dat do procesoru a STORE – uložení dat z procesoru do paměti.

```
LOAD rA, rB, P;
```

Krok	Registrová notace	Popis
T_0	MA ← PC; PC+4	Nahraj hodnotu programového čítače (PC) do MA; Sečti PC a 4 a nahraj do dočasného čítače
T_1	MD ← M[MA]; PC ← C	Nahraj instrukci z paměti z adresy, která byla v MA do jednotky pro styk s prostředím MD(=memory data)
T_2	IR ← MD	Nahraj právě prováděnou instrukci do IR z MD
T_3	A← ((R[rb]=0)→0 : ((R[rb]≠0)→R[rb])	Pokud je rb 0 nahraj do A hodnotu 0, pro nenulové rb nahraj hodnotu rb
T_4	C←A+K[IR]	Pokud n obsahuje hodnotu 0, tak nahraj do n hodnotu P
T_5	MA←C	Provede se jeden posun
T_6	MD←R[ra]	
T_7	M[MA]←MD	Přesuň obsah registru v ALU C do registru ra

```
STORE rA, rB, P
;
```

Krok	Registrová notace	Popis
T_0	MA ← PC; PC+4	Nahraj hodnotu programového čítače (PC) do MA; Sečti PC a 4 a nahraj do dočasného čítače
T_1	MD ← M[MA]; PC ← C	Nahraj instrukci z paměti z adresy, která byla v MA do jednotky pro styk s prostředím MD(=memory data)
T_2	IR ← MD	Nahraj právě prováděnou instrukci do IR z MD
T_3	A← ((R[rb]=0)→0 : ((R[rb]≠0)→R[rb])	Pokud je rb 0 nahraj do A hodnotu 0, pro nenulové rb nahraj hodnotu rb
T_4	C←A+K[IR]	Pokud n obsahuje hodnotu 0, tak nahraj do n hodnotu P
T_5	MA←C	Provede se jeden posun
T_6	MD←M[MA]	
T_7	R[ra] ← MD	Přesuň obsah registru v ALU C do registru ra

Assemblery a zavaděče

Strojový kód

- Programovací jazyk nejnižší úrovně, jehož instrukce je procesor schopen bezprostředně vykonávat
- Představuje binární zápis, v němž se instrukce ukládají do paměti pro přímé použití procesorem počítače
- Pro člověka prakticky nesrozumitelné

Assembler

- Je sestavovací program (překladač) jazyka symbolických instrukcí do strojového kódu, nebo do přemístitelného kódu

Přemístitelného ve smyslu relokace adres, které dosáhnu tak, že všechny instrukce budou adresovat své operandy relativně vůči nějaké hodnotě v registru, změnou hodnoty v registru můžu adresovat na úplně jiných adresách – relocable code

- Jedna instrukce v Assembleru představuje jednu instrukci ve strojovém kódu

Jazyk symbolických instrukcí

- Strojově orientovaný jazyk, jehož instrukce, nebo její části lze zapisovat symbolicky
- Představuje notaci pro pohodlnější reprezentaci programů ve strojovém kódu v symbolice, která je člověku lépe čitelná
- Je-li programových sekcí více, nebo jsou-li užity moduly z knihovny, následuje spojení do jednoho souboru, což provede sestavovací program
- Sestavování může být
    - Absolutní
    - Relativní – výsledkem je kód s relativními adresami vzhledem k počátku programové sekce (viz poznámka u Assembleru)

Sestavovací program

- Program sloužící k vytvoření spustitelného (=zaveditelného, proveditelného) kódu
- Řeší tři základní úlohy
    1. Spojení jednotlivých programovacích sekcí
    2. Vytvoření vnitřních odkazů mezi nimi
    3. Konečné nastavení adres