Generický procesor

Otázky

Architektura PC - obecně

- Účel generického procesoru:
  - Zjednodušení pohledu na procesor (pro výuku). Obsahuje základní části procesoru, které má každý procesor
- Charakteristika procesoru:
  - 32 bitový
  - RISC
- Účel sběrnice:
  - Přenos dat mezi jednotkami sběrnice
- Účel registrů:
  - Uchovávání
    - Výsledky Aritmeticko-Logických instrukcí
    - Adresy dat (adresy instrukcí uchovává programový čítač)
- ALU:
  - K jednotce patří čítač posuvů (opakované volání ALU)
  - K jednotce patří 2 záchytné registry
- Účel ALU
  - Provádění Aritmeticko-Logických operací
- Účel Programového čítače (PC)
  - Jednotka starající se o správné provádění posloupnosti instrukcí
- Účel Instrukčního registru (IR)
  - Stará se o posloupnost úkonů vytáhni a vykonej
- Instrukční registr - IR
  - Samotný instrukční registr (uchovává právě prováděnou instrukci)
  - Dekódovací jednotka
  - Jednotka extrakce konstant
  - Řadič
- Účel řadiče
  - Stará se o generování hodinových impulzů (protože počítač je stavový stroj)
- Účel jednotky pro styk s prostředím (MA a MD)
  - Přesun dat mezi pamětí a procesorem
- Jednotka MA a MD - Popis:
  - MA (Memory adress) – jednocestná jednotka (může pouze odesílat) pro informování operační paměti o adrese instrukce/operandu
  - MD (Memory data) – dvoucestná jednotka (může odesílat a přijímat) pro příjem/odesílání dat z/do operační paměti
- Registrová notace
  - Je notace, která popisuje jednotlivé elementární operace při provádění instrukcí
- Účel registrové notace
  - Registrová notace poskytuje základní vysvětlení jednotlivých dějů instrukcí v procesoru (jelikož i generický procesor je pro vysvětlení)
- Formát instrukce generického procesoru - obecně.
  - Formát instrukce je rozložení a význam jednotlivých operandů a operačního kódu instrukce
- Formát Aritmeticko-Logické instrukce
  - (viz formát instrukce)
- Formát instrukcí pro Řízení chodu programů
  - (viz formát instrukce)
- Formát instrukcí pro přesun dat

- (viz formát instrukce)

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

(koukáme se na stroj jako na fyzický celek, takže u procesoru nás zajímají pouze vstupy/výstupy a jak jsou např. tranzistory uspořádány uvnitř)

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

Základní rysy RISC

1. Minimální stručný repertoár

Snaha o co nejmenší počet instrukcí

2. Jednoduché způsoby adresování
3. Pevný formát instrukce

CISC instrukce mají variabilní délku instrukce, RISC naopak pevnou

4. Vykonání jedné instrukce v jednom strojovém cyklu

Tzv. proudové zpracování instrukcí

5. Datové operace pouze nad registry

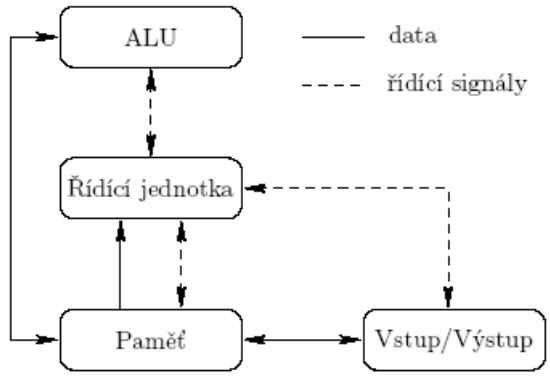
Nelze jako operand uvést adresu v paměti, ale operand musí být pro zpracování instrukce v registrech

6. Styk s pamětí výlučně pomocí instrukcí LOAD / SAVE
7. Zřetězená realizace (provádění) instrukcí

Viz bod 4.

   a. Optimální kompilátor nutností

(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

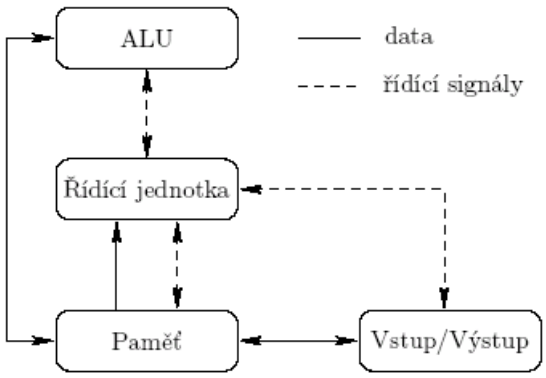
Pro adresaci instrukcí se používá relativní adresace.

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:

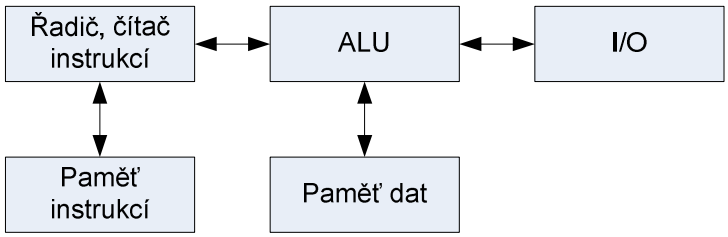
1. Počítač se skládá z vstupních/výstupních zařízení, operační paměti, ALU a řadiče
2. Struktura počítače je nezávislá na typu řešené úlohy, počítač se programuje obsahem paměti.
3. Následující krok počítače je závislý na kroku předchozím

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.

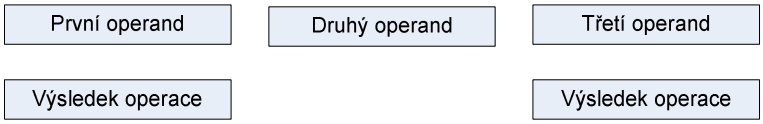
4 - adresové instrukce

- Obsahuje první operand, druhý operand, adresu kam se má uložit výsledek operace a adresu další instrukce
- Používali se u strojů s magnetickým bubnem
- Dnes občas interně užívány v některých strukturách procesorů

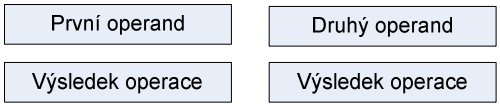
3 – adresové instrukce

- 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, v generickém procesoru PC – programový čítač), který je

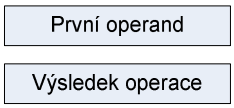
modifikován prováděnou instrukcí (délkou nebo adresou – skoková/neskoková), tak aby ukazoval na následující instrukce
- První operand, nebo poslední operand může být adresa výsledku operace
- Výsledek operace může mít i jinou adresu
- Např. ADD op1,op2, výsledek – ADD je instrukce sčítání a op1 je první operand, op2 druhý operand a výsledek je adresa, nebo místo v paměti, kam se má uložit výsledek



2 – adresové instrukce
- Nepřímo vyjádřená adresa výsledku (operand může být adresa, odkud se má operand vytáhnout a zároveň adresa)
- 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 …, kde oba dva operandy figurují jako operace a jeden z operandů může být adresa, kam se má uložit výsledek



1 – adresové instrukce
- Předpokládá použití ACC (=akumulátor), což je speciální registr, kde je jeden z operandů do kterého bude uložen výsledek prováděná operace. Tzn. první operand, může být místo v paměti a druhý operand už je uložen v procesoru v nějaké jednotce (např. záchytné registry ALU)
  - 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)

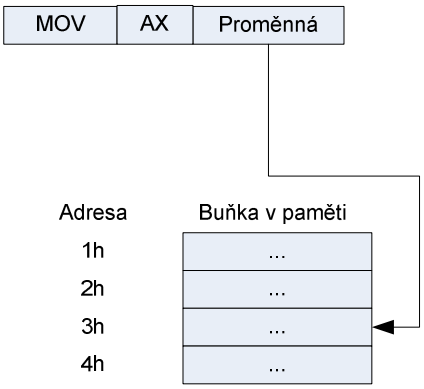
Instrukce – adresace operandů

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
```

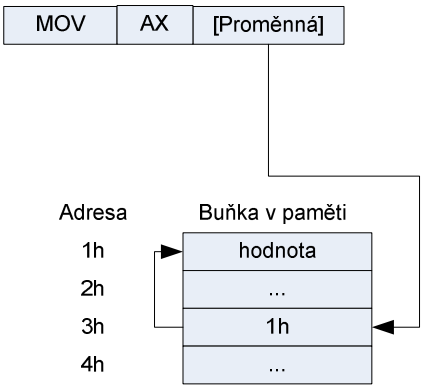


Předpokládáme-li, že existuje Proměnná, tak právě její název nám označuje místo, kde se nachází operand

Nepřímá adresace

- Instrukce obsahuje adresu, na které je adresa efektivní

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

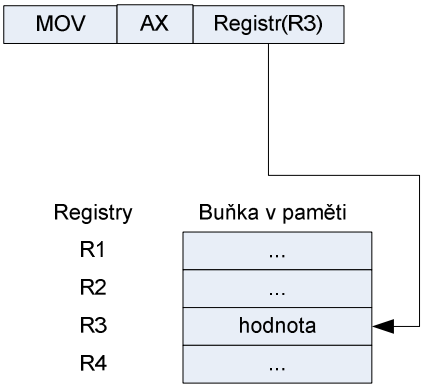


Opět, předpokládáme-li, že existuje proměnná, tak hodnota (která se nachází na efektivní adrese) je na adrese která je uložená na adrese proměnné

Registrová přímá adresace

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

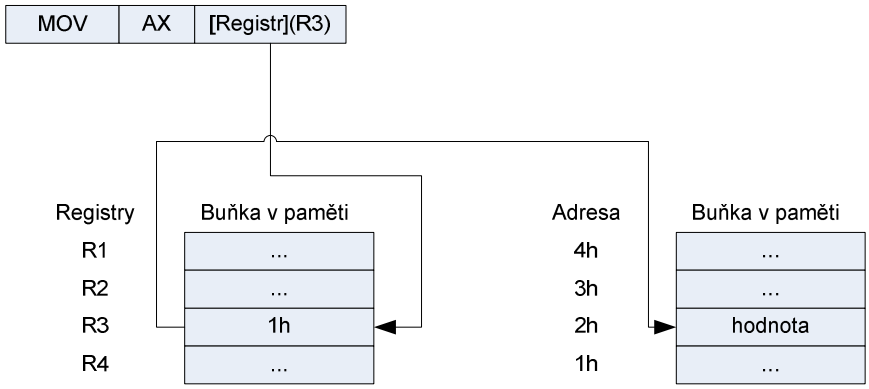
```
mov ax,bx;
add ax,bx;
```



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)

Generický procesor

Účel

Zjednodušení pohledu na procesor (pro výuku procesorů a jejich architektury), jelikož procesor obsahuje základní funkční části procesoru

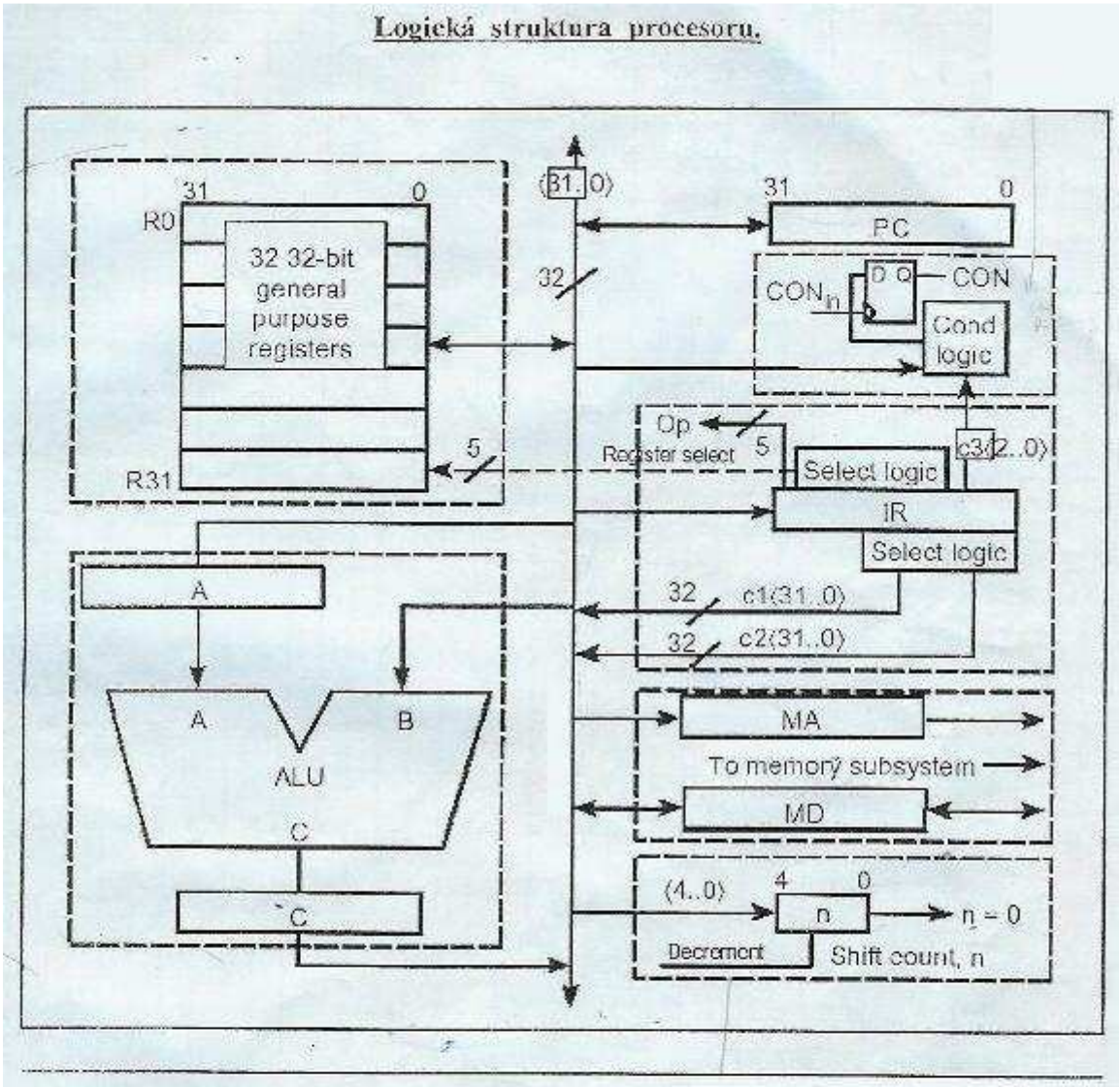
Charakteristika

- 32 bitový procesor
- 32 registrů
- RISC procesor (=redukovaná instrukční sada)
- Instrukce má konstantní délku 4 bajty

Jednotlivé funkční bloky

- **Registr R0…R31** - slouží k ukládání operandů aritmeticko-logických instrukcí, výsledky operací a adresy dat. Soubor 32 třiadvaceti bitových registrů
- **Sběrnice** – slouží k přenosu dat mezi jednotkami procesoru, její šířka je 32 bitů (stejně jako u registrů) a je společná pro adresy i pro operandy
- **ALU** – slouží k provádění aritmeticko-logických operací. K ALU patří čítač posuvů a dva záchytné registry (A – akumulátor, C – dočasné ukládání výsledků, dokud se nepřesunou na místo určení)
- **Jednotka pro řízení chodu programů –** provádí instrukce řízení chodu programů (instrukce podmíněného a nepodmíněného skoku)
- **Programový čítač** – PC – slouží k provádění posloupností instrukcí. Uchovává adresu místa v paměti, kde se nachází následující instrukce
- **Jednotka pro styk s pamětí** – slouží k přenosu dat mezi pamětí a procesorem a je složena z dvou jednotek
    - MA – Memory Adress je jednotka, která předává adresu operandu v paměti, kam se mají operandy uložit/odkud se mají vzít. Jednotka MA je jednosměrná, tzn. je schopna odeslat adresu, nikoli přijmout
    - MD – Memory Data je jednotka, která předává data operační paměti / přijímá data od operační paměti. MD je obousměrná jednotka.
- **Instrukční registr** – IR
    - stará se o posloupnost
        - Vytáhni
        - Vykonej
    - Součástí jednotky je:
        - Registr IR(=Instruction Register)
        - Dekódovací jednotka
        - Extrakce konstant
        - Řadič – stará se o generování hodinových impulzů, protože počítač je stavový stroj

Logická struktura procesoru.



Logická struktura procesoru je složena z bloků :

1. Soubor 32 dvaatřicetibitových registrů
2. Aritmeticko logická jednotka
3. Realizace podmíněných skoku
4. Dekódování instrukce a extrakce konstant
5. Rozhraní pro styk s operační pamětí
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

Formát instrukce

- Formát instrukce je rozložení a význam jednotlivých operandů a operačního kódu instrukce
- Maximální délka instrukce je 4 bajty
- Každá buňka je dlouhá 5 bitů, vyjma konstanty, která je dorovnána od posledního registrového operandu, do 4 bajtů
- Z operačního kódu (OP) si IR zjistí o jaký typ instrukce se jedná a podle toho vyhodnotí, jaké bity obsahují operandy
- Buňky A, B a C obsahují číslo registru jejich délka je 5 bitů (máme 32 registrů, 2^5=32 resp. 0…31)

OP	A	B	C	K

Příklady rozkladu instrukcí

2C 41 00 00

2				C				4				1				0				0				0				0			
0	0	1	0	1	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LA				R17				65536																							

29 DE FF FF

2				9				D				E				F				F				F				F			
0	0	1	0	1	0	0	1	1	1	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
LA				R7				R15				65535																			

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
- Formát instrukce

OP	A	B	C	K

V prvních 5 bitech je operační kód, který definuje instrukci i kolik bude mít operandů. Aritmeticko-logické instrukce mohou pracovat s jedním až třemi operandy, operandy s kterými pracujeme jsou používány z leva a registr nevíce na vpravo v instrukci označuje registr, kam se má uložit výsledek.

```
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 prava (shift right)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
- Formát instrukce

OP	A	B	C	P

Opět v závislosti na operačním kódu:

- V případě, že se jedná o instrukci s návratem má operandy A,B a P, kde A je návratová adresa, B adresa, kam se má skočit (resp.kam se má nastavit prog.čítač) a P podmínku
- V případě, že se jedno o instrukci bez návratu má operandy B a P, specifikuje B adresu kam se má skočit a P podmínku
- Operand C specifikuje podmínku, pro kterou musí platit podmínka P v posledním operandu

```
BR rB, rC, P
; adresa skoku s návratem chybí operand A, takže se jedná o
; instrukci bez návratu (toto zjistí IR). rB je adresa, která se
; nahraje do prog.čítače, tzn. kam se má skočit. rC je podmínka
; která je vyhodnocena v jednotce ; podmíněného skoku a musí
; platit pro podmínku P, jinak je skok neproveden a instrukce
; zahozena
```

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 se podmíněný skok na adresu rB s návratem na adresu rA.
; Za podmínky kterou určije P prováděné na operandu rC
```

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.
- Formát instrukce

```
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