

Strojový jazyk, assembler

Petr Jedlička, Cyril Klimeš, Tomáš Pitner

Předmluva

HTML · Slides · PDF | Přehled předmětu

Cíle

- Seznámit se s **instrukční sadou** vybraných procesorů
- Poznat a pochopit principy programování v **jazyku symbolických adres** (assembleru)

🔑 Počítač vykonává instrukce ve svém **strojovém jazyce**. Lidsky čitelný zápis těchto instrukcí se nazývá **jazyk symbolických adres** neboli *assembly language*; **assembler** je program, který tento zápis překládá do strojového kódu.

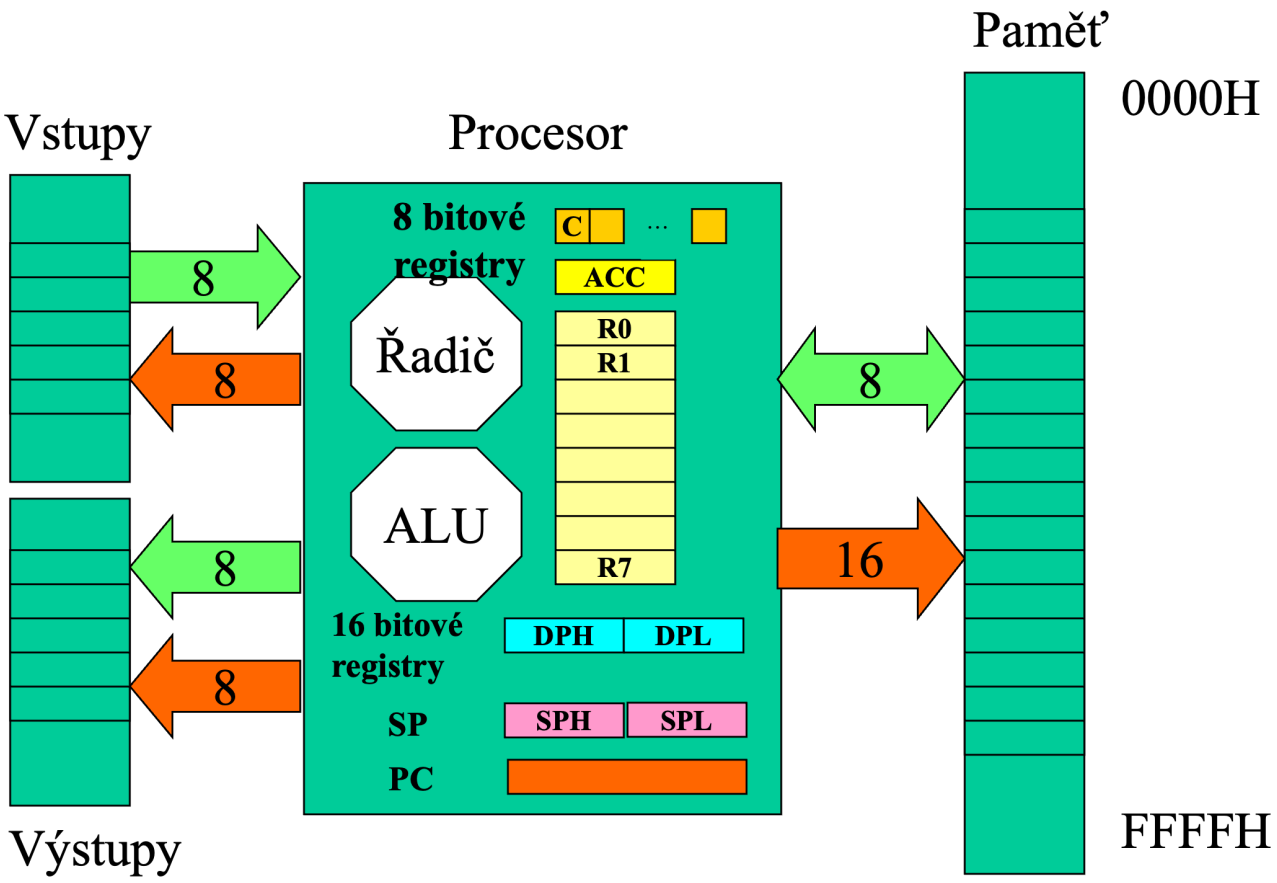
Programování v jazyce symbolických adres

- Znalost tzv. *programátorského modelu* procesoru
- Soupis *instrukcí* procesoru
- Znalost metod určování *operandů*
- Mít k dispozici *assembler*, tedy překladač z jazyka symbolických adres do strojového kódu

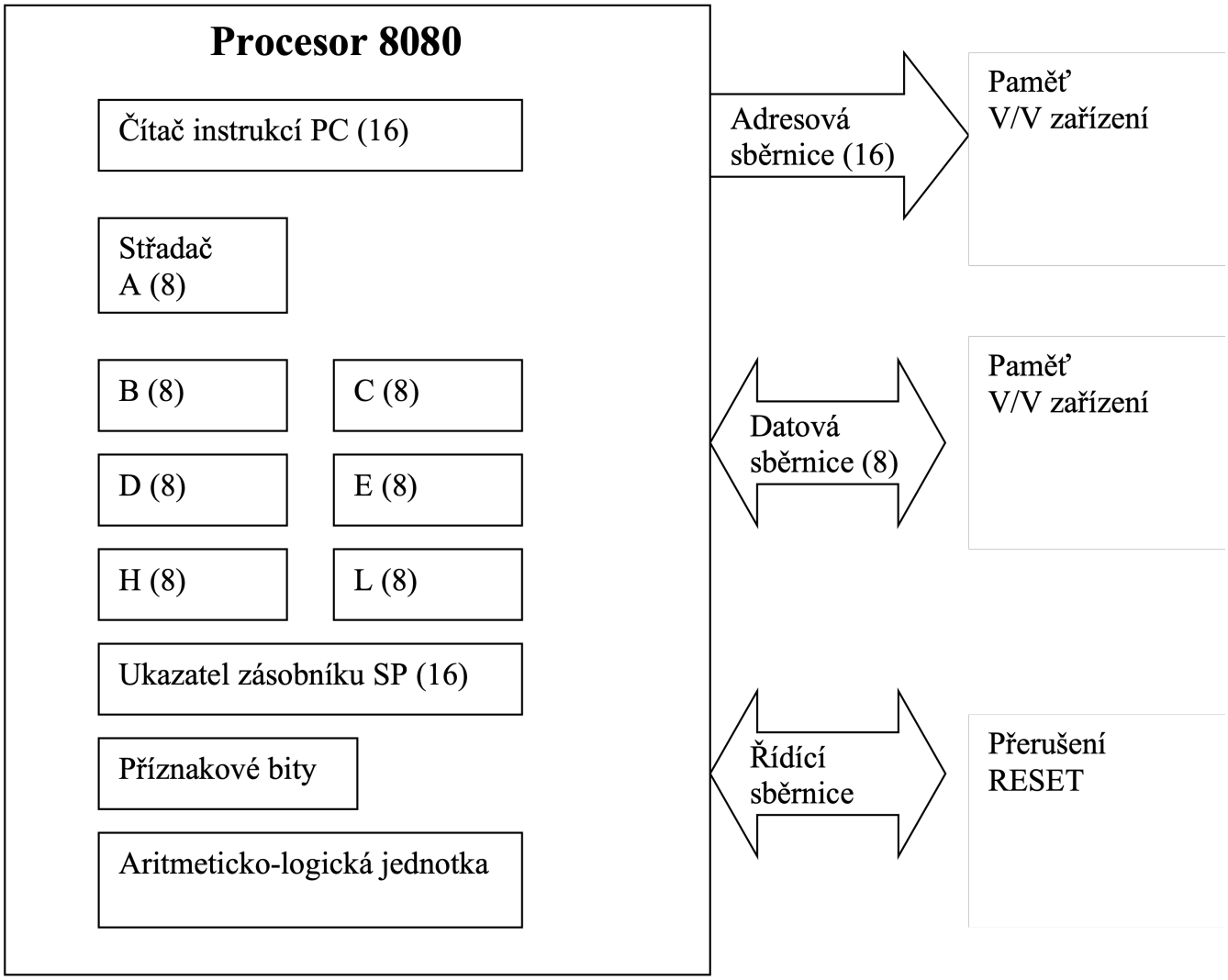
Programátorský model

- Programátorský model je abstrakce popisující, jak software vidí a ovládá procesor: jeho registry, datové typy, instrukce, adresování a stav.
- Jeho znalost je důležitá při vývoji operačních systémů, překladačů, debuggerů a vestavěného softwaru i při analýze výkonu. Běžný aplikační kód však obvykle překládá optimalizující překladač.
- Kvůli jednoduchosti začneme 8bitovým procesorem; základní principy platí také pro 16bitové, 32bitové a 64bitové procesory.

Obecný programátorský model 8-bitového CPU



Procesor Intel 8080



Základní registry procesoru

- **A (Accumulator)** — 8bitový střádač,
- **PC (Program Counter)** — 16bitový čítač instrukcí,
- **SP (Stack Pointer)** — 16bitový ukazatel zásobníku,
- šest 8bitových univerzálních registrů **B**, **C**, **D**, **E**, **H**, **L**,
- registry lze pro 16bitové operace seskupit do párů **BC**, **DE** a **HL**,
- *příznakové bity* (*flags*).

Příznakové bity

Z (Zero)	1 při nulovém výsledku operace, jinak 0
S (Sign)	kopie nejvyššího, tedy znaménkového bitu výsledku
CY (Carry)	přenos z nejvyššího řádu výsledku

AC (Auxiliary Carry)	přenos mezi bity 3 a 4 výsledku
P (Parity)	1 při sudém počtu jedničkových bitů výsledku, jinak 0

Skryté registry

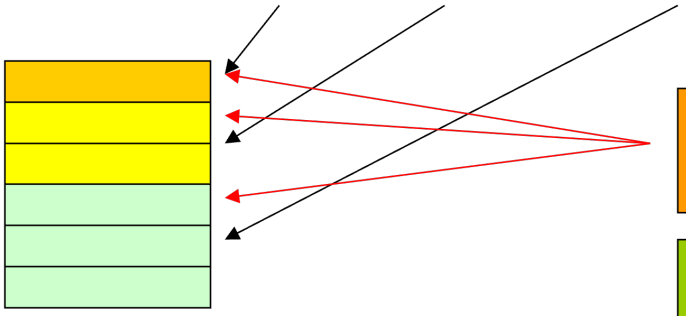
Programátorovi neviditelné:

IR	instrukční registr (8bitový); je napojen na dekodér instrukcí (řadič),
DR	datový registr (8 bitový); registr pro čtení/zápis dat z/do paměti,
AR	adresový registr (16 bitový); adresa pro čtení/zápis z/do paměti,
TA	Temporary Address register (skládá se z TA(H) (TA High - 8 bitů), TA(L) (TA Low - 8 bitů)).

Instrukce procesoru

Skládá se z operačního kódu a operandů/adres

Dělení podle velikosti oblasti paměti, kterou instrukce zabírá:
jednobytové, dvoubajtové, tříbajtové, …



Prvním bytem instrukce je vždy operační kód

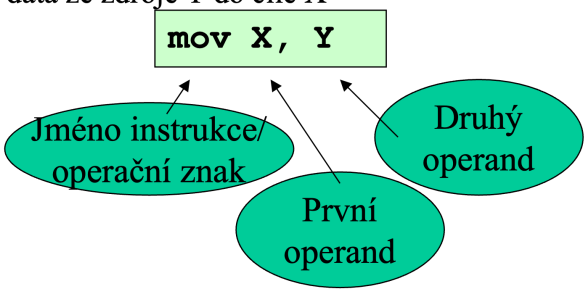
Vícebytové instrukce mají v dalších bytech upřesněny své operandy

Počet bytů pro operandy je vždy dán operačním kódem

Operandy instrukcí mohou být zcela určeny operačním kódem, i jednobytové instrukce tedy mohou mít své operandy.

Zápis instrukce `MOV`

Instrukce kopíruje osmibitová data ze zdroje Y do cíle X



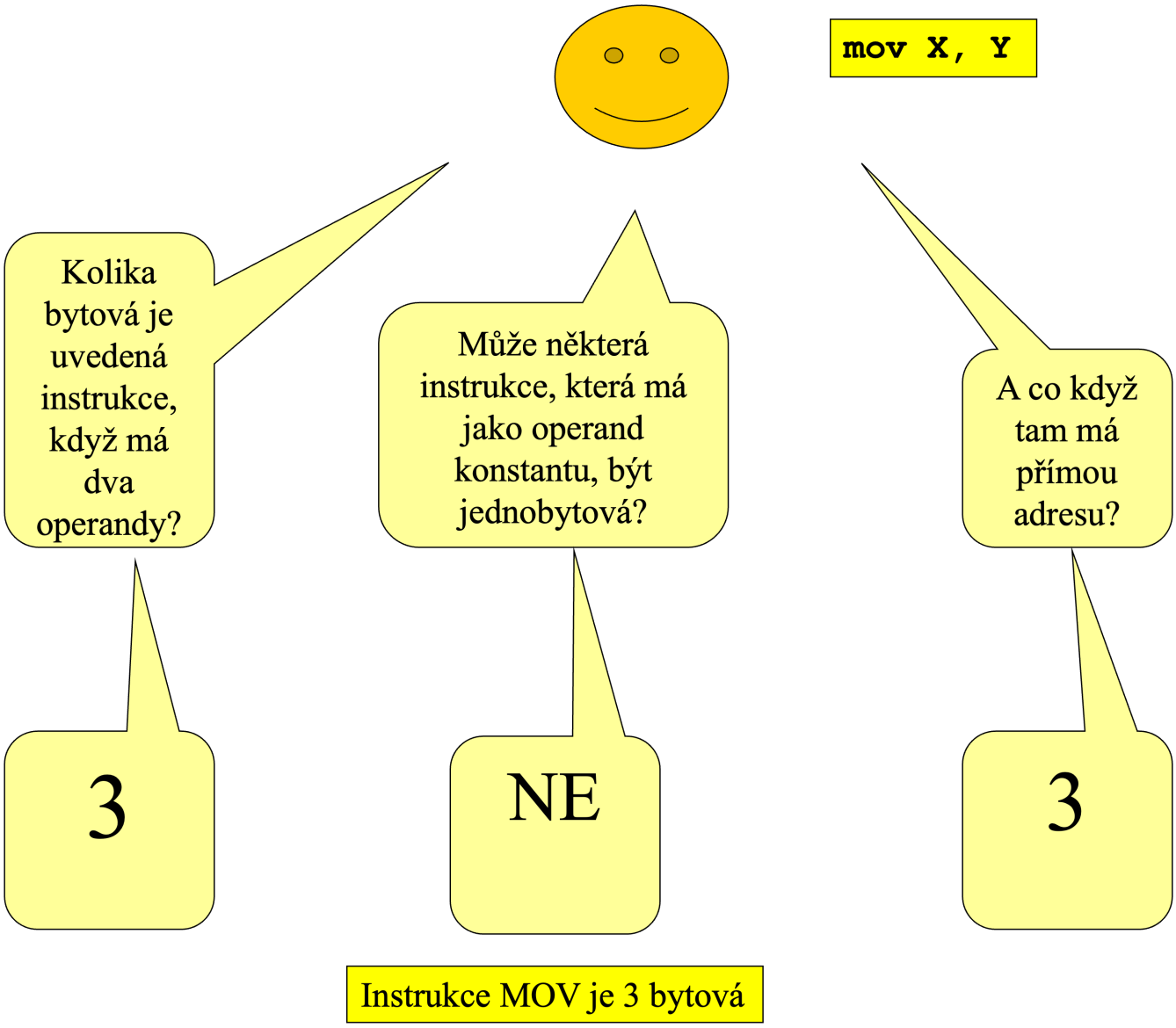
Zápis instrukce se skládá ze jména instrukce/kódu a ze seznamu operandů/adres.

Operandem instrukce může být :

1. vyjmenovaný registr procesoru	mov A, R1
2. konstanta	mov A, #123d
3. místo v paměti určené přímou adresou	mov A, 100b
4. místo v paměti určené nepřímou adresou	mov A, @DPTR

Číselné hodnoty v zápisu instrukcí lze rovnocenně uvádět v dekadické (d) nebo hexadecimální (h) nebo binární (b) soustavě. Zápis hodnoty číselné začíná číslicí 0-9 a může končit určením soustavy. Implicitní je dekadická soustava.

MOV X,Y



Zápis instrukce LDA

- Význam: načtení jednoho bytu z uvedené 16bitové adresy paměti do registru `A`
- Zápis v symbolickém jazyku
  - `LDA adresa`
  - Příklady: `LDA 0010H`, `LDA 1234H`

Příklad programu

- Na adrese X=0000000000010000b a adrese Y=0000000000010001b jsou operandy x = 00000010 a y = 00000011
- Vypočtěte `x := x + y`

```
LDA 0000000000010000b   ;načtení x do registru A
MOV B,A                 ;přesun obsahu A do B
LDA 0000000000010001b   ;načtení y do registru A
```

```
ADD B                   ;sečtení obsahu A a B
STA 0000000000010000b   ;uložení obsahu registru A na adresu X
```

Program v paměti

Adresa d	Adresa h	Adresa b	Obsah b	Význam	Kód h
0	0000	00000000 00000000	00111001	LDA	3A
1	0001	00000000 00000001	00010000	adr x	
2	0002	00000000 00000010	00000000	adr x	
3	0003	00000000 00000011	00100111	MOV B,A	47
4	0004	00000000 00000100	00111001	LDA	3A
5	0005	00000000 00000101	00010001	adr y	
6	0006	00000000 00000110	00000000	adr y	
7	0007	00000000 00000111	01000000	ADD B	80
8	0008	00000000 00001000	00110010	STA	32
9	0009	00000000 00001001	00010000	adr x	
10	000A	00000000 00001010	00000000	adr x	
11	000B	00000000 00001011	00000000		
12	000C	00000000 00001100	00000000		
13	000D	00000000 00001101	00000000		
14	000E	00000000 00001110	00000000		
15	000F	00000000 00001111	00000000		
16	0010	00000000 00010000	00000010	x	
17	0011	00000000 00010001	00000011	y	
18	0012	00000000 00010010	00000000		
19	0013	00000000 00010011	00000000		
20	0014	00000000 00010100	00000000		

Vykonávání a instrukce programu

- Doba provedení instrukce se popisuje počtem taktů; může záviset na instrukci, operandech, přístupu do paměti i stavu mikroarchitektury. U procesoru Intel 8080 se skládá z několika stavů T a strojových cyklů.
- Z pohledu programu se instrukce dokončují tak, aby odpovídaly pořadí určenému čítačem instrukcí, skoky, voláními a přerušeními. Moderní procesor je přitom může interně zpracovávat paralelně nebo mimo pořadí.
- Po resetu začne CPU načítat instrukce z adresy definované danou architekturou. U procesoru Intel 8080 je resetovací adresa `0000H`; jiné architektury ji mohou mít odlišnou. Procesor může být také zastaven instrukcí `HLT`, čekat na událost nebo přejít do úsporného stavu.

Přesun dat (Data Transfer Instructions)

Slouží k přesunu dat mezi registry, pamětí a akumulátorem.

MOV dest, src	přesun dat mezi registry
MVI reg, data	vložení konstanty do registru
LXI rp, data16	načtení 16bitové konstanty do páru registrů
LDA addr / STA addr	načtení/uložení akumulátoru z/do paměti
LHLD addr / SHLD addr	načtení/uložení páru HL z/do paměti
XCHG	výměna HL a DE

Aritmetické instrukce

Provádějí sčítání, odčítání a další operace.

ADD reg / ADI data	sčítání s akumulátorem
SUB reg / SUI data	odčítání od akumulátoru
INR reg / DCR reg	inkrementace/dekrementace
DAD rp	sčítání páru registrů s HL
DAA	úprava akumulátoru pro BCD

Logické instrukce

Provádějí logické operace jako AND, OR, XOR.

ANA reg / ANI data	AND s akumulátorem

ORA reg / ORI data	OR s akumulátorem
XRA reg / XRI data	XOR s akumulátorem
CMP reg / CPI data	porovnání s akumulátorem
RLC, RRC, RAL, RAR	rotace bitů v akumulátoru

Řídicí instrukce

Slouží k řízení toku programu.

JMP addr	skok na adresu
JC, JNC, JZ, JNZ, JP, JM, JPE, JPO	podmíněné skoky
CALL addr / RET	volání a návrat z podprogramu
CC, CNC, CZ, CNZ, CP, CM, CPE, CPO	podmíněná volání podprogramu
RC, RNC, RZ, RNZ, RP, RM, RPE, RPO	podmíněné návraty z podprogramu

Instrukce se zásobníkem

PUSH rp / POP rp	práce se zásobníkem
SPHL	nastavení SP podle HL
XTHL	výměna HL se zásobníkem

Vstupně-výstupní instrukce

IN port	načtení dat z I/O portu do akumulátoru
OUT port	odeslání dat z akumulátoru do I/O portu



- Instrukce `IN` přečte v okamžiku provedení jeden byte ze zvoleného I/O portu do akumulátoru `A`.
- Instrukce `OUT` zapíše jeden byte z akumulátoru `A` do zvoleného I/O portu. Zda zařízení hodnotu uchovává, závisí na jeho řadiči; není to obecná vlastnost všech výstupních portů.

Řídicí instrukce

NOP	žádná operace
HLT	zastavení procesoru
DI / EI	zakázání/povolení přerušení

Příklad výpočtu stejného výrazu na různých architekturách

- Výpočet stejného výrazu s různým počtem explicitních operandů instrukce.
- Více explicitních operandů může snížit počet potřebných instrukcí, ale obvykle zvětšuje jejich kódování. Rychlost závisí také na mikroarchitektuře, přístupech do paměti a práci překladače; z počtu operandů ji nelze určit.

Počet explicitních operandů instrukce

- Nyní opustíme svět procesoru Intel 8080 a porovnáme instrukce podle počtu explicitně zapsaných operandů. Nejde o adresovací režimy a konkrétní ISA nemusí nabízet všechny uvedené varianty.

Tříoperandové	instrukce se třemi explicitními operandy (např. `ADD A,B,C` znamená `A := B + C`)
Dvouoperandové	instrukce se dvěma explicitními operandy (např. `ADD A,B` znamená `A := A + B`)
Jednooperandové	instrukce s jedním explicitním operandem (např. `INC A` znamená `A := A + 1`)
Nulaoperandové	operandy jsou implicitně na zásobníku; například `ADD` odebere dvě horní hodnoty a vrátí jejich součet

Tříoperandová instrukce

- `Y = A * B + (C - D) * E / F`
  1. **Násob**: `A * B → P1`
  2. **Odečti**: `C - D → R`
  3. **Násob**: `R * E → R`
  4. **Děl**: `R / F → R`
  5. **Sečti**: `R + P1 → Y`

Dvouoperandová instrukce

- `Y = A * B + (C - D) * E / F`

1. **Násob A B**: `A * B → R` ; výsledek jde vždy do R
2. **Zapiš do P1**: `R → P1` ; zapisuje se R do P1
3. **Odečti C D**: `C - D → R` ; výsledek jde vždy do R
4. **Násob E**: `R * E → R` ; výsledek jde vždy do R

5. **Děl F**: `R / F → R`
6. **Přičti P1**: `R + P1 → R`
7. **Zapiš do Y**: `R → Y`

Jednooperandová instrukce

- `Y = A * B + (C - D) * E / F`

1. **Načti A**: `A → R`
2. **Násob B**: `R * B → R`
3. **Zapiš do P1**: `R → P1`
4. **Načti C**: `C → R`
5. **Odečti D**: `R - D → R`
6. **Násob E**: `R * E → R`
7. **Děl F**: `R / F → R`
8. **Přičti P1**: `R + P1 → R`
9. **Zapiš do Y**: `R → Y`

Realizace výpočtů v Intel 8080

- sčítání (ADD, ADI) a odčítání (SUB, SUI) je přímo podporováno instrukcemi procesoru
- násobení a dělení je realizováno pomocí podprogramů, protože instrukce procesoru je nepodporují
- dnešní CPU (např. x86-64, Arm/AArch64) obvykle umí násobit i dělit přímo v hardwaru; celočíselné dělení nemusí být podporováno u jednodušších vestavěných jader (např. Cortex-M0), kde se realizuje softwarově

Násobení (8bit × 8bit)

1. Vstupy: činitelé v registrech A a B
2. Inicializuj výsledek na 0.
3. Opakuj B-krát:
4. Přičti A k výsledku.
5. Použij registr jako čítač smyčky.
6. Výsledek je 16bitový, protože součin dvou 8bitových neznaménkových čísel může být až 65025 (`255 x 255`).

Násobení v assembleru Intel 8080

```
    ; Vstup: A = první činitel, B = druhý činitel
```

```
    ; Výstup: DE = 16bitový výsledek
    ; Mění také registry A, C a H
    MOV H, A            ; H = původní první činitel
    MOV C, B            ; C = čítač smyčky
    MVI D, 00H          ; D = vyšší byte výsledku
    MVI E, 00H          ; E = nižší byte výsledku
    MOV A, C
    ORA A               ; nastav Z, pokud je druhý činitel nulový
    JZ MUL_DONE
MUL_LOOP:
    MOV A, E
    ADD H               ; přičti první činitel k nižšímu bytu
    MOV E, A
    JNC NO_CARRY
    INR D               ; přenos do vyššího bytu
NO_CARRY:
    DCR C
    JNZ MUL_LOOP
MUL_DONE:               ; DE obsahuje výsledek
```

Dělení

- Dělení `A / B` vrací podíl v `C` a zbytek v `A`.
- Výsledek je: C = podíl (kolikrát se dělitel `B` vejde do dělence) A = zbytek po dělení

Algoritmus dělení pomocí odčítání

1. Inicializuj podíl na 0.
2. Dokud je dělenec větší nebo roven děliteli:
3. Odečti dělitel od dělence.
4. Inkrementuj podíl.
5. Zbytek je hodnota, která zůstane z dělence.

Dělení dvou 8-bitových čísel v assembleru Intel 8080

- Podprogram vrací podíl v registru `C` a zbytek v registru `A`.
- Vstupní hodnoty jsou dělenec v registru `A` a dělitel v registru `B`.

```
; Dělení dvou 8bitových čísel
; vstup:
;   A = dělenec
;   B = dělitel
; výstup:
;   C = podíl
;   A = zbytek
```

```
;
; předpoklad: B != 0
DIV8:
        MVI C, 00H      ; podíl = 0
LOOP:   CMP B           ; porovnej A s B
        JC  DONE        ; A < B -> konec
        SUB B           ; A = A - B
        INR C           ; podíl++
        JMP LOOP
DONE:                    ; C = podíl, A = zbytek
```

Použití podprogramu

```
        MVI A, 25H      ; dělenec
        MVI B, 06H      ; dělitel
        CALL DIV8       ; výsledek: C = podíl, A = zbytek
                         ; zde C = 06H a A = 01H
```

Moderní architektury: programátorský model x86-64

- Nyní podrobněji k architektuře **x86-64** a.k.a. **Intel x64**, přesněji **AMD64**.
- Základní x86-64 má 16 obecných 64bitových registrů: `RAX`, `RBX`, `RCX`, `RDX`, `RSI`, `RDI`, `RSP`, `RBP` a `R8` až `R15`.
- Rozšíření Intel APX definuje dalších 16 obecných registrů; software je smí použít jen tehdy, když je podporují procesor, operační systém, ABI a nástroje.
- Registry mají pojmenované podčásti, například `RAX`, `EAX`, `AX` a `AL`. Zápis do 32bitové podčásti, například `EAX`, v 64bitovém režimu nuluje horních 32 bitů registru.

Instrukční sada x86-64

- Architekturu původně uvedla společnost AMD jako **AMD64**; Intel používá kompatibilní označení **Intel 64**.
- Zachovává proměnnou délku instrukcí a rozsáhlou zpětnou kompatibilitu x86.
- Volitelná rozšíření, například AVX, AVX2, AVX-512 nebo APX, je nutné před použitím detekovat; nejsou přítomna v každém procesoru.

Paměťový model

- Lineární adresy mají v 64bitovém režimu 64bitový formát, ale musejí být kanonické.
- Čtyřúrovňové stránkování překládá 48 bitů lineární adresy; volitelné pětistupňové stránkování **LA57** překládá 57 bitů.
- Šířka fyzické adresy je implementační vlastnost zjišťovaná instrukcí `CPUID`; architektura ji omezuje nejvýše na 52 bitů.
- V 64bitovém režimu se obvykle používá plochý paměťový model. Báze segmentů `CS`, `DS`, `ES` a `SS` se fakticky neuplatňují, zatímco `FS` a `GS` mohou mít nenulovou bázi.

Režimy běhu

- **Legacy mode** používá tradiční 16bitové nebo 32bitové režimy x86.
- **IA-32e/long mode** zahrnuje dva podrežimy:
  - **compatibility mode** pro vybraný 16bitový nebo 32bitový kód pod 64bitovým operačním systémem,
  - **64-bit mode** pro 64bitový kód a 64bitové obecné registry.

Stav procesoru

- Registr `RFLAGS` obsahuje stavové a řídicí příznaky, například Zero, Carry a Overflow.

- Řídicí registry `CR0`, `CR2`, `CR3`, `CR4` a `CR8` spravují mimo jiné režim procesoru, stránkování a přerušení.
- `EFER` (*Extended Feature Enable Register*) je modelově specifický registr MSR, nikoli člen řady řídicích registrů `CR`.

Výpočty s plovoucí řádovou čárkou

- Historická jednotka x87 používá zásobník registrů `ST(0)` až `ST(7)`.
- V moderním 64bitovém kódu se skalární a vektorové výpočty obvykle provádějí instrukcemi SSE/SSE2 a podle dostupnosti AVX, AVX2 nebo AVX-512.

Obecné registry

Většinu registrů lze použít pro výpočty, data i adresy. Jejich zvláštní role často vyplývá z konkrétní instrukce nebo aplikačního binárního rozhraní (ABI).

`RAX`	tradiční akumulátor; ABI jej obvykle používá také pro návratovou hodnotu
`RBX`	obecný registr s historickou rolí bázového registru
`RCX`	obecný registr a implicitní čítač některých instrukcí
`RDX`	obecný registr používaný implicitně například při násobení, dělení a I/O
`RSI`	obecný registr a implicitní zdrojový index řetězcových instrukcí
`RDI`	obecný registr a implicitní cílový index řetězcových instrukcí
`RSP`	ukazatel zásobníku
`RBP`	obecný registr, který lze podle ABI a nastavení překladače použít jako ukazatel rámce
`R8`–`R15`	další obecné registry základní architektury x86-64



Použitelné podčásti nejsou u všech registrů pojmenovány stejně. Například `RAX` má podčásti `EAX`, `AX`, `AL` a historickou horní osmibitovou část `AH`; registr `R8` má podčásti `R8D`, `R8W` a `R8B`, ale obdobu `AH` nemá.

Ukazatel instrukce a příznaky

`RIP`	*Instruction Pointer*, určuje adresu následující instrukce podle aktuálního toku řízení
`RFLAGS`	registr příznaků
`ZF`	*Zero Flag*, výsledek operace je nula
`CF`	*Carry Flag*, přenos při aritmetice
`SF`	*Sign Flag*, znaménko výsledku

`OF` *Overflow Flag*, přetečení se znaménkem

Vektorové registry

`XMM0`–`XMM15` 128bitové registry základního 64bitového prostředí, používané instrukcemi SSE/SSE2

`YMM0`–`YMM15` 256bitové rozšíření registrů XMM dostupné s AVX

`ZMM0`–`ZMM31` 512bitové registry AVX-512; jejich nižší části se zároveň označují XMM a YMM

- Přesný počet a šířku použitelných vektorových registrů určuje režim procesoru a podporovaná rozšíření ISA.

Řídicí a systémové registry

- Řídicí registry a registry MSR jsou určeny především pro operační systém a privilegovaný software.
- Dostupnost konkrétních registrů a funkcí se zjišťuje pomocí `CPUID`; nelze ji předpokládat pouze podle označení x86-64.

Doporučené zdroje

- Intel 64 and IA-32 Architectures Software Developer’s Manuals (HTML)
- AMD64 Architecture Programmer’s Manual
- NASM — aktuální dokumentace assembleru
- ASM80 — přehled instrukcí Intel 8080
- Český úvod do programování v assembleru

Na další kapitolu: Paměti

HTML · Slides · PDF | Přehled předmětu