Adresový prostor 8086

programový čítač (registr IP) ... adr. násl. instr.:

16 b ... $2^{16} = 64 \times 1\,024 = 64\,\mathrm{K} = 65\,536$

max. kapacita HP (pro 8086): ? perspektivní ?

$1\,\mathrm{M} = 2^{20} = 1\,048\,576$ ... 20 b

Řešení? *segmentace*:

IP neobsahuje „absolutní adresu“, ale tzv. její *posunutí* [offset]

vůči 16násobku obsahu segm. reg. CS, tzn.

posunutí = adresa − 16 × ⟨segm. reg.⟩

tedy:

adresa = posunutí + 16 × ⟨segm. reg.⟩

Př.: $IP = 1234_{16}$ $CS = 9ABC_{16}$

1234	0001 0010 0011 0100	pos.
9ABC0	1001 1010 1011 1100 0000	segm.
9BDF4	1001 1011 1101 1111 0100	adr.

Př.: Je-li CS = 9ABC, pak:

minim. adr.: 9ABC0 = 9ABC0 + 0

maxim. adr.: AABBF = 9ABC0 + FFFF

tzv. *segment*

??? adresy 9ABBF nebo AABC0 apod. ???

nutno změnit obsah segmentového registru

Segmentace

hlavní paměť: rozdělena na *paragrafy* po 16 B

1 MB = 64 K paragrafů

segment. registr obsahuje číslo prvního paragrafu

segment: začíná na začátku některého paragrafu

64 KB = 4 K paragrafů

program: segm. reg. CS ⇒ programový segment

(někdy: kódový segment)

data: segm. reg. DS ⇒ datový segment

analogicky jako programový segment:

V instrukci není uvedena adresa dat,

ale její posunutí vůči 16násobku DS

další segmentové registry:

SS — tzv. zásobník (všimneme si později)

ES — data ve speciálních případech

implicitní segm. reg.: CS — program

DS — data (obvykle)

ES — data (někdy)

SS — zásobník

počáteční nastavení registrů:

techn. prostředky (HW) — CS, IP, DS, SS, ES, FLAGS

progr. prostředí (SW) — CS, IP, SS, SP (zprav.)

ostatní — nutno zajistit v programu !!!

Jednoduchý program

(1 datový a 1 programový segment)

```
    .MODEL SMALL      ; 1 dat. a 1 progr. segm.
    .STACK 100H       ; zásobník
    .DATA             ; datový segment
A   DW   12 DUP (?)
B   DW   1
C   DW   ?
      :
    .CODE             ; programový segment
      :
S:  MOV  AX, @DATA    ; start
    MOV  DS, AX       ; nastavení DS
      :
K:  MOV  AX,4C00H     ; stop
    INT  21H
      :
    END  S            ; S = startovaci adresa
```

.MODEL, .STACK, .DATA, .CODE, END } tzv. direktivy (příkazy) asembleru

@DATA = první paragraf dat. segmentu (konstanta)

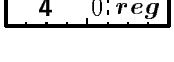
Strojový kód 8086

(příklady)

① CLI ... IF := 0

② $reg \in \{AX, \ldots DI\}$ — 2 slabiky (2 B)

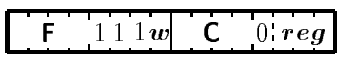
INC reg



př.: INC SI ... 46

③ reg ... dat. reg. nebo ukazatel — 1 nebo 2 B

INC reg

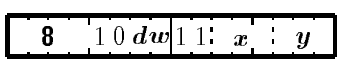


[word]

w	
0	1 B
1	2 B

př.: INC DH ... FE C6 INC SI ... FF C6

④ MOV $r1, r2$



[direction]

d	
0	$x \rightarrow y$
1	$x \leftarrow y$

př.: MOV CH,AL ... 88 C5 nebo 8A E8

Strojový kód 8086 (příklady) 2

⑤ MOV *reg, konst*

B	*w*	*reg*	*konst*

př.: MOV CH,41H ... B5 41
MOV CX,123H ... B9 23 01

⑥ MOV *reg, pam* a MOV *pam, reg*

8	1 0 *dw*	0 0	*reg*	1 1 0	*posunutí*

d	
0	*reg* → *pam*
1	*reg* ← *pam*

př. předp.: posunutí(Beta) = $12_{10} = C_{16}$
MOV CX,Beta ... 8B 0E 0C 00
MOV Beta,CX ... 89 0E 0C 00

⑦ JNZ *nv*

7	5	*vzd*

př.: 0004F: 40 CYKL: INC AX
00050: 75 FD JNZ CYKL
00052: ?? ???
4F − 52 = −3 ∼ FD (doplňkový kód)

Nepodmíněné skoky (JMP)

- tzv. **krátké** — JMP SHORT *náv*
  vzd = „vzdálenost“: $-128 \le vzd \le 127$

E B	*vzd*

(2 B)

IP := IP + *vzd*

srov.: **podmíněné skoky** (všechny jsou „krátké“)

- tzv. **blízké** — JMP NEAR PTR *náv*
  uvnitř segmentu (programového)

E 9	*vzd*

(3 B)

IP := IP + *vzd*

- tzv. **daleké** — JMP FAR PTR *náv*
  mezi segmenty (programovými)

E A	*posunutí*	*segment*

(5 B)

IP := *posunutí* [offset]
CS := *segment*

Pozn.: existují ještě jiné nepodmíněné skoky
(„cílová“ adresa je v registru nebo v proměnné)

Pozn.: Asembler zprav nevyžaduje, aby se uvádělo „NEAR PTR“

Způsoby adresace

operandy/výsledek: 1. registry
2. místa v hlavní paměti

ad 2:

- *přímé adresy*: dosud uvažované adresy, např.:
  ADD AX,xyz nebo ADD AX,xyz+3
- *indexované adresy* — reg. SI a DI, např.:

```
Arr     DB    8,3,5,4
        :
        MOV   SI,0          ; SI = 0
        MOV   AL,0
        MOV   CX,4
cyklus: ADD   AL, Arr[SI]   ; AL = 8, 11, 16, 20
        INC   SI            ; SI = 1, 2, 3, 4
        LOOP  cyklus
```

lze však také použít např. 7[DI] nebo [SI] ≡ 0[SI]

- *bázované adresy* — reg. BX a BP
  analogicky — např. [BX], 3[BX] anebo Arr[BP]
- *kombinace* — např. [SI][BX], 5[BP][DI]

Pozn.: Jsou možné i jiné zápisy, např.
5[BP+DI] ≡ 5[BP][DI]

datové segmenty

implicitní segmentový registr — DS
výjimky: adresy bázované registrem BP — SS
některé operady operací s řetězci — ES

Př.: SI = 1, BP = 2, DS = 3, SS = 4
8[SI] ∼ 00039 8[BP] ∼ 0004A

explicitní určení sementového registru:
prefixové instrukce (krátce — prefixy):

ES:	SEGES	(26)
CS:	SEGCS	(2E)
SS:	SEGSS	(36)
DS:	SEGDS	(3E)

Př.: SI = 1, BP = 2, DS = 3, SS = 4
SS:8[SI] ∼ 00049 DS:8[BP] ∼ 0003A

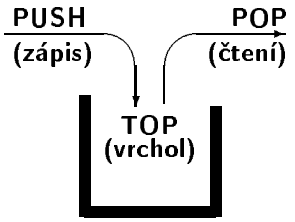
Př.: mov ax,ES:qwert ≡ SEGES mov ax,qwert

Př.: JSI	stroj. kód
mov ax,1[si]	8B 44 01
mov ax,ES:1[si]	26 8B 44 01

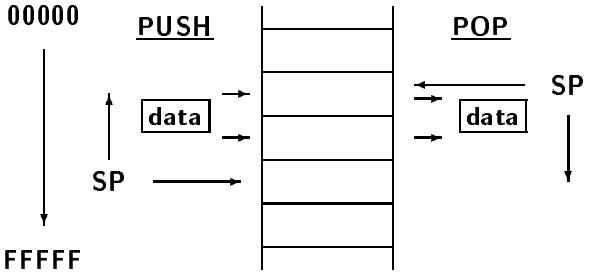
Zásobník

(zásobníková paměť, sklípková paměť, paměť LIFO)

[stack, push-down, ..., Last-In-First-Out]



zásobník simulovaný v hlavní paměti



adresa v hlavní paměti = SP + 16 × SS
SP ... posunutí [offset]

Instrukce PUSH a POP

PUSH γ ... $\gamma \rightarrow$ *zásobník*; **SP := SP − 2**

γ: **registr — 2 slabiky — kromě IP a FLAGS**
paměť — 2 slabiky

predekrementace SP — nejprve snížit, pak zapsat

Př.: **SS = 8A74 SP = 20**

PUSH AX

1. **SP := 1F**
2. **AH → 8A75F**
3. **SP := 1E**
4. **AL → 8A75E**

POP δ ... $\delta \leftarrow zsobnk$; **SP := SP + 2**

δ: **registr — 2 slabiky — kromě CS, IP a FLAGS**
paměť — 2 slabiky

postinkrementace SP — nejprve číst, pak zvýšit

Př.: **SS = 8A74 SP = 1E**

POP AX

1. **⟨8A75E⟩ → AL**
2. **SP := 1F**
3. **⟨8A75F⟩ → AH**
4. **SP := 20**