Strojový kód

instrukce = příkaz (zakódovaný jako „číslo“)

Obsahuje (popř. může obsahovat) tyto informace:

① co se má provést (jaká operace)

② s čím se to má provést (operandy) a kam se má uložit výsledek

③ kde se má pokračovat (adr. násl. instr.)

Tyto informace mohou být obsaženy:

- explicitně *v instrukci*

  Př.: počítač SAPO — 5 adresové instrukce:
  - ① ... tzv. operační znak
  - ② ... 2+1 adresa
  - ③ ... 2 adresy — následující instrukce při záporném a nezáporném výsledku
- zčásti explicitně *v instrukci*, zčásti určeny *implicitně* architekturou procesoru, např.:
  - ① ... operační znak — OZ
  - ② ... adresová část instrukce

  } instrukce
  - ③ ... von Neumannova koncepce ⇒ další instrukce na následující adrese ← architektura

operační kód = *soubor instrukcí*: OZ ∼ operace

dále předpokl.: von Neumannova koncepce ⇒

⇒ *programový čítač* PC [Program Counter] obsahuje adresu následující instrukce

⇒ skoky (nepodmíněné a podmíněné) „neproduktivní“ operace

⇒ kratší instrukce

3 adresová instrukce:

OZ	a_1	a_2	a_3

- „nejpřirozenější“: 2 operandy + 1 výsledek

  např.: $\langle a_1\rangle - \langle a_2\rangle \to a_3$
- poměrně dlouhá

2 adresová instrukce:

OZ	a_1	a_2

- výsledek se ukládá na místo prvního operandu

  např.: $\langle a_1\rangle - \langle a_2\rangle \to a_1$
- je třeba zavést „neproduktivní“ operaci přesun: $\langle a_2\rangle \to a_1$

Př.: $\langle x\rangle - \langle y\rangle \to z \quad \equiv \quad \begin{cases} \langle x\rangle \to z \\ \langle z\rangle - \langle y\rangle \to z \end{cases}$

1 adresová instrukce:

OZ	a

- zvl. registr — *střadač* S [Accumulator]

  1. operand a výsledek např.: $\langle S\rangle - \langle a\rangle \to S$
- operace přesunu: $\langle a\rangle \to S$ a $\langle S\rangle \to a$

Př.: $\langle x\rangle - \langle y\rangle \to z \quad \equiv \quad \begin{cases} \langle x\rangle \to S \\ \langle S\rangle - \langle y\rangle \to S \\ \langle S\rangle \to z \end{cases}$

více střadačů ⇒ číslo střadače ∈ instrukce

⇒ lze zavést operace mezi střadači

Př.: Motorola — řada 68000

datové registry D_0, D_1, ..., D_7 à 32 b

odčítání 32 b: $D_n - \text{paměť} \to D_n$

OZ:

9	n	0	B	9

instrukce:

OZ	adresa

$\langle D_3\rangle - \langle$18 FF20÷18 FF23$\rangle \to D_3$:

96B9 0018 FF20

Zápis programu

strojový kód: strojové instrukce — „čísla“

Př.: Intel 8086 (zjednodušeno)

AX — střadač 16 b

A10401 $\langle 104\div105\rangle \to$ AX

2B060601 $\langle$AX$\rangle - \langle 106\div107\rangle \to$ AX

A30201 $\langle$AX$\rangle \to 102\div103$

- pracné programování i změny
- nepřehledný zápis

vyšší programovací jazyk (VPJ)

(např. Pascal)

Př.: a := b − c

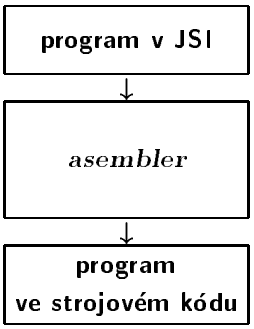
- omezení (zejm. pro „speciální“ účely)
- méně efektivní (?)

jazyk symbolických instrukcí (JSI):

operační znak i adresy zapsány symbolicky

Př.:
```
MOV AX,b
SUB AX,c
MOV a,AX
```

Vytváření programů ve strojovém kódu



vstupní data

program

výstupní data

VPJ: analogicky
místo asembleru jiný program
— *překladač (kompilátor)*

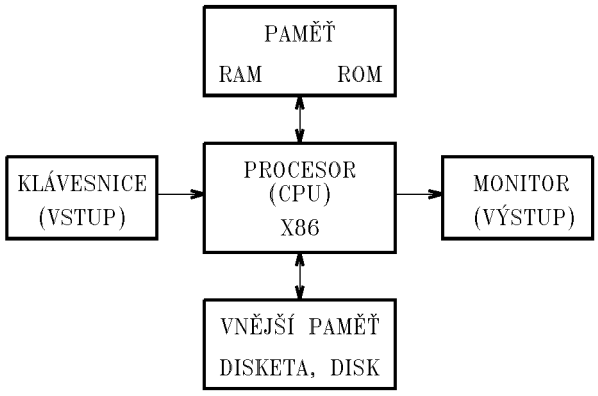
někdy:

program v JSI
program ve VPJ
} → překladač → modul

moduly → spojovací program [linker] → stroj. kód

Ideové schéma technické platformy

(IBM PC–kompatibilní osobní počítač)



Řada procesorů Intel 80x86

rok		tranz. [tis.]	MIPS	při MHz	dat.	adr.
1971	*4004	2,3	0.06	0,108	4	10
1978	8086	29	0.33	5	16	20
1979	8088	29	0.33	5	8	20
1982	80286	134	1.2	8	16	24
1985	80386DX	275	6	16	32	32
1988	80386SX	275	2.5	16	16	24
1989	i486DX	1200	20	25	32	32
1993	Pentium	3100	112	66	64	32
1995	Pentium Pro	5500	300	133	64	36
1997	Pentium II	7500		300	64	36
1999	Pentium III	28000		733	64	36

1981 IBM PC
1984 IBM PC-AT

Typy instrukcí X86 (=INSTRUKČNÍ SOUBOR)

- přesun dat
  (MOV,...,IN,OUT,...PUSH,POP,CLI,..)
- aritmetické i. (ADD,...,INC,..)
  logické i. (AND,...,XOR,...)
  posuvy (SHL,...,RCR,...)
- skoky (JMP,...,JC,JNC,....)
  cykly (LOOP,...,LOOPZ,...)
  podprogramy (CALL,RET,...)
  přerušení (INT,IRET,....)

- instrukce pro práci s řetězci
- pseudoinstrukce (např. deklarace proměnných DB,DW,..)
- makroinstrukce

Registry procesorů řady 80x86

16 bitů = 2 × 8 bitů

datové registry

0	AX	4	AH	AL	0 [Accumulator]
1	CX	5	CH	CL	1 [Counter]
2	DX	6	DH	DL	2 [Data]
3	BX	7	BH	BL	3 [Base]

ukazatelé

4	SP	Ukazatel zásobníku [Stack Pointer]
5	BP	Bázový registr [Base Pointer]
6	SI	Indexregistry [Source Index]
7	DI	Indexregistry [Destination Index]

segmentové registry

0	ES	[Extra Segment]
1	CS	program [Code Segment]
2	SS	zásobník [Stack Segment]
3	DS	data [Data Segment]

IP	Programový čítač [Instruction Pointer]
FLAGS	Registr příznaků [Status Flags]

registr příznaků — FLAGS

				O	D	*I*	T	S	Z		*A*		P		C
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

0	C ≡ CF	přenos	[Carry]
2	P ≡ PF	parita	[Parity]
4	***A*** ≡ ***AF***	*pomocný přenos*	[Auxiliary carry]
6	Z ≡ ZF	nula	[Zero]
7	S ≡ SF	znaménko	[Sign]
8	T ≡ TF	krokování	[Trap]
9	***I*** ≡ ***IF***	*přerušení*	[Interrupt]
10	D ≡ DF	směr	[Direction]
11	O ≡ OF	přeplnění	[Overflow]

IF, TF a DF: **ovlivňují činnost procesoru**

ostatní: **nastavovány některými instrukcemi informace o výsledku operace**

vybrané instrukce 8086

(v JSI)

přesuny dat

MOV *kam,co* ... *kam* := *co*

kam: datový reg., ukazatel, segment. reg., paměť

co: datový reg., ukazatel, segment. reg., paměť, konstanta

nelze: segment. reg. := konstanta
paměť := paměť

Př.:
```
MOV AL,DH        MOV DS,BX
MOV SI,65        MOV prom,AX
```

konstanta — tzv. *přímý operand* [immediate]

různé způsoby zápisu hodnot:

65535 = 0FFFFH = 1111111111111111B
65 = 41H = 01000001B = 1000001B = 'A'
3142H = '1B'

zápis čísla musí začínat desítkovou číslicí:
FFFFH — špatně 0FFFFH — dobře

nutný souhlas délek — platí i pro další instrukce !!!
Př.: MOV AX,BL — špatně (16 b versus 8 b)

deklarace proměnných:

— vyhrazení místa v paměti (pro data)
— přidělení symbolických jmen
— příp. jejich inicializace (počáteční hodnoty)

tzv. pseudoinstrukce:

- **DB** [Define Bytes] — slabiky (1 B = 8 b)
- **DW** [Define Words] — slova (2 B = 16 b)
- **DD** [Define Dwords] — dvojitá slova (4 B = 32 b)

⋮

Příklad:
```
bflm  DB   ?    ; poč. hodn. není definována
psvz  DB   5    ; poč. hodn. = 5
tab   DB   0FH, 5, '='   ; nahrazuje 3 deklarace
text  DB   'UPS'   ; místo 'U', 'P', 'S'
pole  DB   3 DUP (8, ?)   ; místo 8, ?, 8, ?, 8, ?
prom  DW   ?
      DB   1011B
adr   DD   bflm, prg   ; „adresy“
prg:  ⋮
      MOV AL,psvz   ; bflm = ?
      MOV bflm,AL   ; bflm = 5
```

aritmetické operace

Operandy a výsledek lze interpretovat jako:

- **čísla v doplňkovém kódu, např. (pro 1 B = 8 b):**
  $FF_{16} \sim$ (-1) $\in \langle -128_{10}; 127_{10} \rangle$
- **nezáporná čísla, např. (pro 1 B = 8 b):**
  $FF_{16} \sim FF_{16} = 255_{10} \in \langle 0; 255_{10} \rangle$

ADD α,β ... $\alpha := \alpha + \beta$

α:	β:
datový reg.	datový reg.
ukazatel	ukazatel
paměť	paměť
	konstanta

nelze: paměť := paměť + paměť

příznaky **CF, OF, SF a ZF:**

Př.: ADD AL,BL

nezáporná č.			doplňkový kód			příznaky			
AL	BL	AL+BL	AL	BL	AL+BL	CF	OF	SF	ZF
7A	78	0 F2	∼7A	∼78	∼(-E)	0	1	1	0
7A	FF	179	∼7A	∼(-1)	∼79	1	0	0	0

pozn.: dále se nastavují příznaky PF a AF

ADC α,β ... $\alpha := \alpha + \beta + CF$

SUB α,β ... $\alpha := \alpha - \beta$

analogické ADD, až na ***příznak*** **CF:**

$\alpha < \beta \iff CF := 1$ (α, β — nezáporná čísla)

zde: **CF ... tzv. výpůjčka** [borrow]

SBB α,β ... $\alpha := \alpha - \beta - CF$

CMP α,β ... $\alpha - \beta$

stejné jako SUB, ale neukládá se výsledek — nastaví se však ***příznaky !***

NEG α ... $\alpha := 0 - \alpha \quad = -\alpha$

(uvažuje se doplňkový kód)

Př.: MOV AL,1 ... 0000 0001
NEG AL ... 1111 1111 = 0FFH
CF=1, ZF=0, SF=1, OF=0

INC α ... $\alpha := \alpha + 1$ **nemění se příznak CF**

DEC α ... $\alpha := \alpha - 1$ **nemění se příznak CF**

logické operace

AND α,β ... $\alpha := \alpha \wedge \beta$ **logický součin**

OR α,β ... $\alpha := \alpha \vee \beta$ **logický součet**

XOR α,β ... $\alpha := \alpha \oplus \beta$ **součet modulo 2**

příznaky: CF := 0
SF ... výsl. < 0
ZF ... výsl. = 0
OF := 0

	0	1	0	1
	0	0	1	1
AND	0	0	0	1
OR	0	1	1	1
XOR	0	1	1	0

TEST α,β ... $\alpha \wedge \beta$ **stejné jako AND, ale neukládá se výsledek — nastaví se však** ***příznaky !***

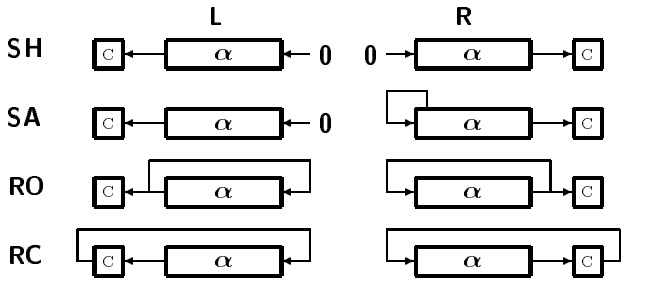
NOT α ... $\alpha := \overline{\alpha}$ **negace** ($\overline{0} = 1; \overline{1} = 0$)

Př.: MOV DH,13H ... 0001 0011
XOR DH,86H ... 1000 0110
1001 0101 = 95H
ZF=0, SF=1
NOT DH ... 0110 1010 = 6AH
ZF=0, SF=0

posuvy

posuv
- **logický** **SH** [SHift]
- **aritmetický** **SA** [Shift Arithmetic]
- **cyklický** **RO** [ROtate]
- **cyklický přes C** **RC** [Rotate through C]

posuv
- **vlevo** **L** [Left]
- **vpravo** **R** [Right]



SHL α, *o_kolik* $\quad o\_kolik = \begin{cases} 1 \\ CL \end{cases}$

analogicky: SHR, SAL, ...RCL

kromě příznaku C ≡ CF se mění některé další příznaky

Př.: MOV AX,1234H ... AX = 1234H
MOV CL,4
ROL AX,CL ... AX = 2341H CF=1

skoky

nepodmíněný skok (nejjednodušší varianta):

JMP *náv* — *náv* ... návěští instrukce

```
Př.:            MOV  AX,1      ①
                JMP  NAVESTI   ②
        COSI:   ADD  BX,CX
        NAVESTI: RCR AX,1      ③
```

podmíněné skoky:

JC *náv* ... je-li C=1, skok na *náv* (jinak nic)

JNC *náv* ... je-li C=0, skok na *náv* (jinak nic)

Př.: AX := max (BX,CX) — nezáporná čísla:

```
        MOV  AX,BX
        CMP  CX,BX
        JC   OK
        MOV  AX,CX
OK:
```

analogicky: JZ, JNZ, JS, JNS, JO, JNO, JP, JNP

podmíněné skoky orientované na srovnání:

relace	doplňkový kód	nezáporná čísla
$<$	JL	JB
$\leq$	JLE	JBE
$=$	JE	JE
$\neq$	JNE	JNE
$\geq$	JGE	JAE
$>$	JG	JA

E ... Equal
L ... Less
G ... Greater
B ... Bellow
A ... Above

podmínka skoku ⇐ příslušné příznaky, např.:

JB ... CF = 1 $\Longrightarrow$ JB ≡ JC
JBE ... CF∨ZF = 1
JL ... SF⊕OF = 1
JGE ... SF⊕OF = 0 atd.

pozn.: v předch. příkl. lze JC nahradit JB

Př.: AX := max (BX,CX) — doplňkový kód :
řešení: v předch. příkl. JC nahradit JL

rozsah podm. skoků: −128 ÷ 127 vůči IP

Př.:
```
        JC   OK
        MOV  AX,CX   ←IP   ↑128↑   ↓127↓
```

„delší“ skoky: opačný podm. skok + nepodm. skok

cykly

```
Př.:            MOV  CX,300
        CYKL:   ⋮
                DEC  CX      ; mění příznaky
                JNZ  CYKL
```

LOOP *náv* CX := CX − 1;
je-li CX ≠ 0, skok na *náv*

nemění příznaky

```
Př.:            MOV  CX,300
        CYKL:   ⋮
                LOOP CYKL    ; nemění příznaky
```

LOOPE *náv* CX := CX − 1;
je-li CX ≠ 0 a ZF = 1, skok na *náv*

LOOPNE *náv* CX := CX − 1;
je-li CX ≠ 0 a ZF = 0, skok na *náv*

LOOPZ *náv* jako LOOPE *náv*

LOOPNZ *náv* jako LOOPNE *náv*

rozsah skoku stejný jako u podmíněných skoků