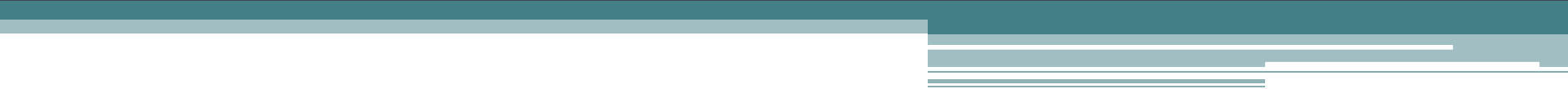
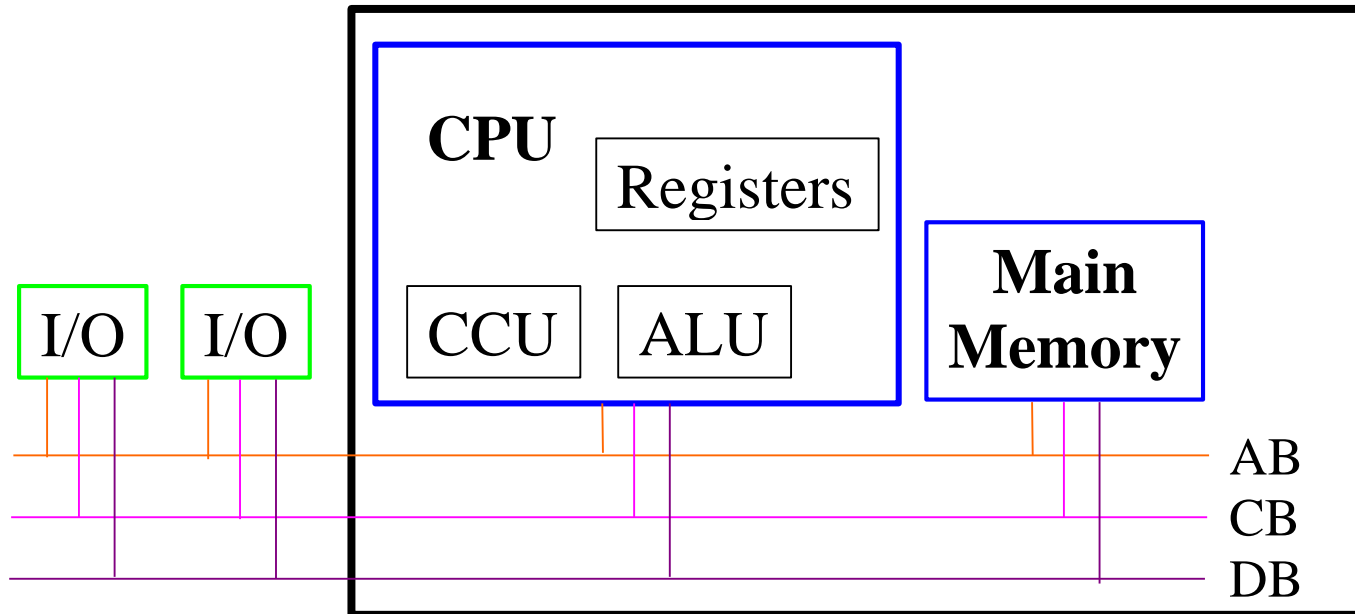


2. Strojový jazyk, jazyk symbolických instrukcí, assembler.



Zjednodušené blokové schéma počítače



CPU	Central Processing Unit (procesor)
ALU	Arithmetic and Logic Unit
CCU	Central Control Unit (řadič)
I/O	Input/Output Unit
AB, CB, DB	Address Bus, Control Bus, Data Bus

Významné registry

Accumulator (A)

Střádač

Instruction register (IR)

Instrukční registr

Instruction pointer register (IP)

Registr ukazatele instrukcí

Hlavní paměť (Main Memory) je většinou organizována po slabikách, tj. slabika je její nejmenší adresovatelnou jednotkou:

1 B = byte (slabika - osm bitů)

zkratka		hodnota
kB	kilobyte	$10^3 = 1\,000\text{ B}$
KiB	kibi byte	$2^{10} = 1\,024\text{ B}$
MB	megabyte	$10^6 = 1\,000\,000\text{ B}$
MiB	mebi byte	$2^{20} = 1\,048\,576\text{ B}$
GB	gigabyte	$10^9 = 1\,000\,000\,000\text{ B}$
GiB	gibi byte	$2^{30} = 1\,073\,741\,824\text{ B}$
TB	terabyte	$10^{12} = 1\,000\,000\,000\,000\text{ B}$
TiB	tebi byte	$2^{40} = 1\,099\,511\,627\,776\text{ B}$

Různý význam obsahu paměti:

1	1	1	1	1	1	0	1
s	exp			fraction			

1. Celé číslo bez znaménka ... 253
2. Celé číslo se znaménkem ... -3
3. Znak ASCII (čeština) ... ř
4. Desetinné číslo bez znaménka, například (podle polohy řádové tečky uvnitř zprava: 126.5, 63.25, 31.625, atd.
5. Desetinné číslo se znaménkem, například (podle polohy řádové tečky uvnitř zprava): -1.5, -0.75, -0.375, atd.
6. Reálné číslo, například $-1^s 2^{\text{exp}-4} f_3.f_2 f_1 f_0$: $-1^1 * 2^3 * 1.625 = -13.0$
7. Instrukce

Instrukce

- Instrukce jsou příkazy pro procesor, které bezprostředně řídí jeho činnost. Každá instrukce obsahuje operační kód a může obsahovat i adresy operandů, nebo přímo operandy.
- Operační kód jednoznačně určuje operaci a jeho součástí mohou být i adresy registrů, se kterými instrukce pracuje.
- Přes rozdíly v instrukčních souborech různých procesorů lze vysledovat několik základních typů instrukcí:
 - instrukce přenosové
 - instrukce aritmetické
 - instrukce logické
 - instrukce posuvů a rotací
 - instrukce skokové
 - instrukce řetězové
 - instrukce řídicí

Činnost počítače

1. Do instrukčního registru se uloží obsah paměťového místa, které je adresováno registrem ukazatele instrukcí.
2. Nastaví se nový obsah registru ukazatele instrukcí (ukazuje na následující instrukci).
3. Obsah instrukčního registru je dekodován, t.j. určí se požadovaná operace a adresy příslušných operandů.
4. Provede se určená operace (u instrukcí skokových se nastaví nový obsah registru ukazatele instrukcí).
5. Pokud nebyla provedená instrukce pokynem k zastavení procesoru, pokračuje se znovu od bodu 1.

Hypotetický počítač 1

Hypotetický počítač 1 (8-bitový střádač, paměť 32 B)

bit	7	6	5	4	3	2	1	0
	operační kód			adresa				

op. kód	symbolická instrukce	operace
000	NOP	prázdná operace
001	LOAD <i>adr</i>	uloží do střádače hodnotu z paměti (z adresy <i>adr</i>)
010	ADD <i>adr</i>	přičte k hodnotě ve střádači hodnotu z paměti (z adresy <i>adr</i>)
011	SAVE <i>adr</i>	uloží hodnotu ze střádače do paměti (na adresu <i>adr</i>)
100	NEG	změní znaménko hodnoty ve střádači
101	JMP <i>adr</i>	skočí na adresu <i>adr</i>
110	JN <i>adr</i>	skočí na adresu <i>adr</i> pokud je hodnota ve střádači záporná
111	HALT	zastaví výpočet

Příklad

Program ve strojovém kódu

adresa	obsah
00101	00110110
00110	11001010
00111	00110111
01000	11010000
01001	10101101
01010	00110111
01011	11001101
01100	10110011
01101	10000000
01110	01010110
01111	11010011
10000	00110110
10001	01111000
10010	11100000
10011	00110111
10100	01111000
10101	11100000
10110	00111000
10111	11110101
11000	xxxxxxxx

Nevýhody programování ve strojovém kódu:

- Programování je obtížná a nepřehledné.
- Program je prakticky nečitelný.
- Program je obtížně modifikovatelný.
- Program nemůže spolupracovat s jinými programy.

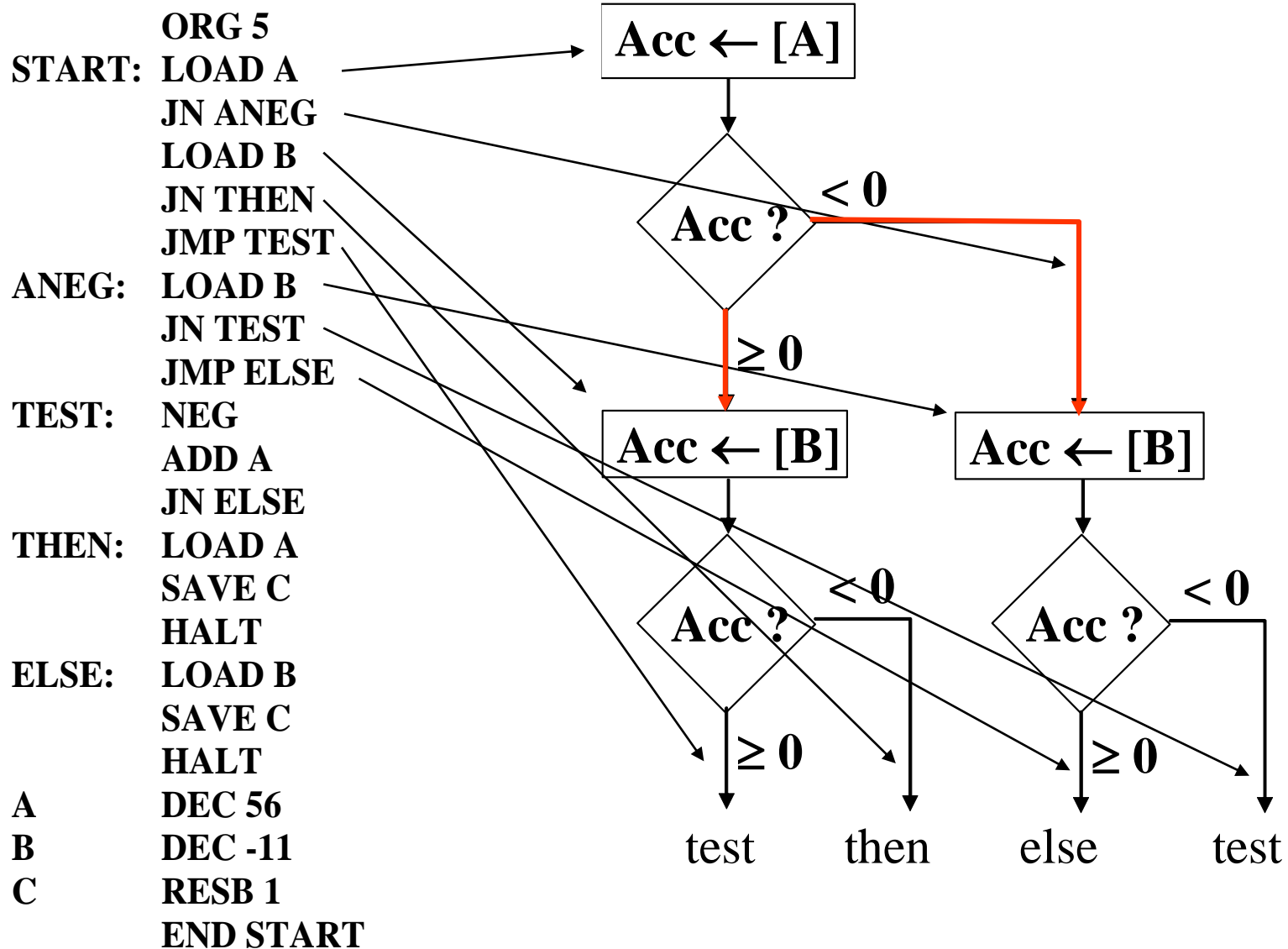
- Jazyk symbolických instrukcí (JSI) používá k programování symbolický zápis jednotlivých instrukcí.
- Programování v JSI je výrazně snadnější, program je (i když stále obtížně) čitelný a modifikovatelný a lze tvořit i spolupracující programy.
- Musíme mít k dispozici překladač JSI do strojového kódu (*assembler*), který ke své činnosti používá speciální pokyny (*direktivy*).

Direktivy pro náš hypotetický assembler:

ORG adr	definice počáteční adresy programu
END adr	označení konce programu a definice startovací adresy
DEC number	definice dekadické konstanty
RESB n	rezervace n slabik paměti

Příklad

if (a >= b) c = a; else c = b; (c = max(a,b))



Příklad

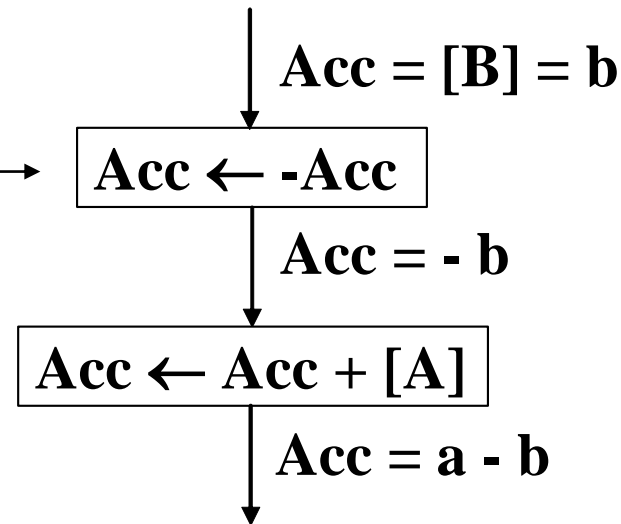
if (a >= b) c = a; else c = b; (c = *max*(a,b))

```

ORG 5
START: LOAD A
      JN ANEG
      LOAD B
      JN THEN
      JMP TEST
ANEG:  LOAD B
      JN TEST
      JMP ELSE
TEST:  NEG
      ADD A
      JN ELSE
THEN:  LOAD A
      SAVE C
      HALT
ELSE:  LOAD B
      SAVE C
      HALT

A      DEC 56
B      DEC -11
C      RESB 1
      END START

```



Překlad

strojový kód

```

      ORG 5
START: LOAD A
      JN ANEG
      LOAD B
      JN THEN
      JMP TEST
ANEG:  LOAD B
      JN TEST
      JMP ELSE
TEST:  NEG
      ADD A
      JN ELSE
THEN:  LOAD A
      SAVE C
      HALT
ELSE:  LOAD B
      SAVE C
      HALT
A      DEC 56
B      DEC -11
C      RESB 1
      END START

```

adresa	obsah
00101	001?????

Problém:

Překladač nezná hodnotu
reprezentovanou symbolem A
a nemůže proto pokračovat
v překladu !!!

Problém z předchozího snímku lze řešit dvěma způsoby:

1. Program je napsán tak, aby každý symbol byl před svým použitím již definován (používá se velmi zřídka).
2. Překlad probíhá ve dvou průchodech zdrojovým programem:
 - a) V prvním průchodu se vytváří tzv. tabulka symbolů, kterým se v průběhu překladu přiřazují hodnoty, které tyto symboly reprezentují.
 - b) Pokud je tabulka symbolů vytvořená v prvním průchodu jednoznačná (každému symbolu je přiřazena právě jediná hodnota) proběhne ve druhém průchodu vlastní překlad.

Překlad – 1. průchod

```

                    ORG 5
5   START: LOAD A
6           JN ANEG
7           LOAD B
8           JN THEN
9           JMP TEST
10  ANEG:  LOAD B
11           JN TEST
12           JMP ELSE
13  TEST:  NEG
14           ADD A
15           JN ELSE
16  THEN:  LOAD A
17           SAVE C
18           HALT
19  ELSE:  LOAD B
20           SAVE C
21           HALT
22  A      DEC 56
23  B      DEC -11
24  C      RESB 1
                    END START

```

tabulka symbolů

<u>symbol</u>	<u>adresa</u>
START	00101
A	10110
ANEG	01010
B	10111
THEN	10000
TEST	01101
ELSE	10011
C	11000

Překlad – 2. průchod

```

      ORG 5
START: LOAD A
      JN ANEG
      LOAD B
      JN THEN
      JMP TEST
ANEG:  LOAD B
      JN TEST
      JMP ELSE
TEST:  NEG
      ADD A
      JN ELSE
THEN:  LOAD A
      SAVE C
      HALT
ELSE:  LOAD B
      SAVE C
      HALT
A      DEC 56
B      DEC -11
C      RESB 1
      END START

```

instrukce

000	NOP
001	LOAD adr
010	ADD adr
011	SAVE adr
100	NEG
101	JMP adr
110	JN adrc
111	HALT

tabulka symbolů

START	00101
A	10110
ANEG	01010
B	10111
THEN	10000
TEST	01101
ELSE	10011
C	11000

strojový kód

adresa	obsah
00101	00110110
00110	11001010
00111	00110111
01000	11010000
01001	10101101
01010	00110111
01011	11001101
01100	10110011
01101	10000000
01110	01010110
01111	11010011
10000	00110110
10001	01111000
10010	11100000
10011	00110111
10100	01111000
10101	11100000
10110	00111000
10111	11110101
11000	xxxxxxxx

Příklad `if (a>=b) c=a; else c=b; (c=max(a,b))` – jiná varianta

Symbolický program

```

A      ORG 0
      DEC 56
B      DEC -11
C      RESB 1
START: LOAD A
      JN ANEG
      LOAD B
      JN THEN
      JMP TEST
ANEG:  LOAD B
      JN TEST
      JMP ELSE
TEST:  NEG
      ADD A
      JN ELSE
THEN:  LOAD A
      JMP CONT
ELSE:  LOAD B
CONT:  SAVE C
      HALT
      END START

```

Tabulka symbolů

symbol	adresa
A	00000
ANEG	01000
B	00001
C	00010
ELSE	10000
CONT	10001
START	00011
TEST	01011
THEN	01110

Příklad if (a>=b) c=a; else c=b; (c=max(a,b)) – jiná varianta

symbolický program

```

      ORG 0
A     DEC 56
B     DEC -11
C     RESB 1
START: LOAD A
      JN ANEG
      LOAD B
      JN THEN
      JMP TEST
ANEG:  LOAD B
      JN TEST
      JMP ELSE
TEST:  NEG
      ADD A
      JN ELSE
THEN:  LOAD A
      JMP CONT
ELSE:  LOAD B
CONT:  SAVE C
      HALT
      END START
  
```

instrukce

000	NOP
001	LOAD adr
010	ADD adr
011	SAVE adr
100	NEG
101	JMP adr
110	JN adrc
111	HALT

tabulka symbolů

START	00011
A	00000
ANEG	01000
B	00001
THEN	01110
TEST	01101
ELSE	10000
C	00011

strojový kód

adresa obsah

00000	00111000
00001	11110101
00010	xxxxxxxx
00011	00100000
00100	11001000
00101	00100001
00110	11001110
00111	10101011
01000	00100001
01001	11001011
01010	10110000
01011	10000000
01100	01000000
01101	11010000
01110	00100000
01111	10110001
10000	00100001
10001	01100010
10010	11100000

Hypotetický počítač 2

Hypotetický počítač (4 registry: R0, R1, R2 a SP, paměť 256 B).

Slabikové instrukce:

bit	7	6	5	4	3	2	1	0
	operační kód				reg1/-		reg2/-	

op. kód	symbolická instrukce	operace
0000	NOP	prázdná operace
0001	MOV reg1, reg2	přenesení do registru <i>reg1</i> hodnotu z registru <i>reg2</i>
0010	MOV [reg1], reg2	přenesení do slabiky paměti, jejíž adresa je v registru <i>reg1</i> , hodnotu z registru <i>reg2</i>
0011	MOV reg1, [reg2]	přenesení hodnotu ze slabiky paměti, jejíž adresa je v registru <i>reg2</i> , do registru <i>reg1</i>
0100	CMP reg1, reg2	porovná hodnoty v registrech <i>reg1</i> a <i>reg2</i> a nastaví příznaky <i>CF</i> , <i>OF</i>
0101	PUSH reg1	uloží obsah registru <i>reg</i> na zásobník: $SP = SP - 1$, $[SP] = reg$
0110	POP reg1	uloží obsah vrcholu zásobníku do registru <i>reg</i> : $reg = [SP]$, $SP = SP + 1$
0111	HALT	zastaví výpočet

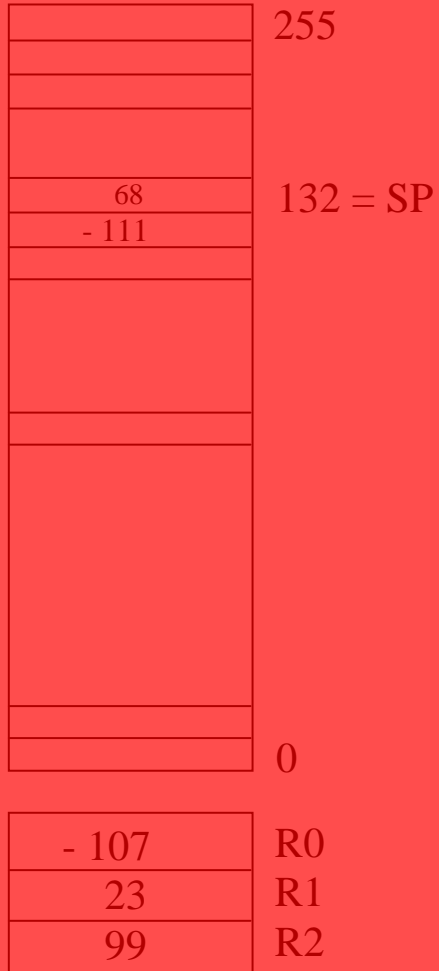
Dvouslabikové instrukce:

bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
	operační kód				reg1/-		reg2/-		adresa adr/číslo n							

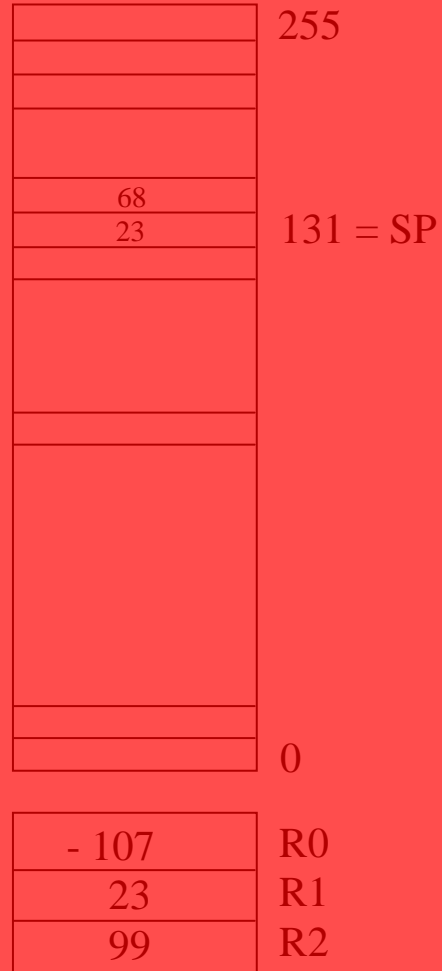
op. kód	symbolická instrukce	operace
1000	MOV reg1, [adr]	přenesení do registru reg operand ze slabiky paměti, jejíž adresa je adr
1001	MOV reg1, adr	přenesení do registru reg adresu adr
1010	MOV reg1, n	přenesení do registru reg číslo n
1011	CALL adr	skočí do procedury začínající na adrese adr : $SP = SP - 1$, $[SP] = IP$, $IP = adr$
1100	RET n	vrátí řízení volajícímu programu (návrat z procedury) a odstraní n slabik ze zásobníku: $IP = [SP]$, $SP = SP + n + 1$
1101	JMP adr	skočí na adresu adr
1110	JG adr	skočí na adresu adr , pokud při předchozím porovnání instrukcí <i>CMP</i> byl první operand větší než druhý (čísla se znaménky!!)
1111	MOV reg1, [reg2+ n]	přenesení do registru $reg1$ operand ze slabiky paměti, jejíž adresa je dána součtem obsahu registru $reg2$ a čísla n

Ukázka práce se zásobníkem:

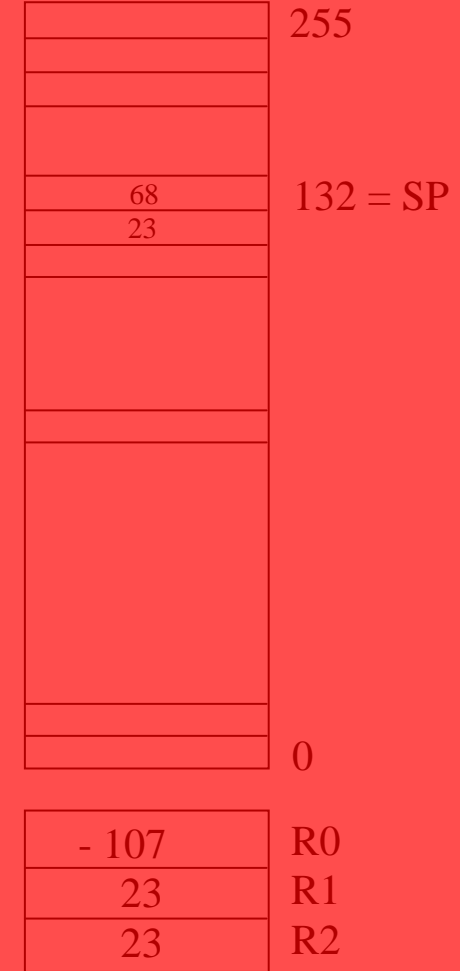
Výchozí stav (například):



Stav po PUSH R1



Stav po POP R2



Direktivy

ORG adr	definice první adresy programu
END adr	označení konce programu a definice startovací adresy
DEC number	definice dekadické konstanty
RESB n	rezervace n slabik paměti

Příklad

$e = \max(a,b,c,d)$

...

$POM1 = \max(A,B)$

$POM2 = \max(POM1,C)$

$E = \max(POM2,D)$

...

```
MOV R1,[r]           ; p = max(r,q)
MOV R2,[q]
MOV R0,p
CMP R1,R2
JG VETSI_r
MOV [R0],R2
JMP DALE
VETSI_r:
MOV [R0],R1
DALE:
```


a) Řešení pomocí opakující se posloupnosti instrukcí:

adresa paměti	symbolická instrukce nebo direktiva	poznámka
	ORG 0	
00	A DEC 56	; definice konstant
01	B DEC -11	
02	C DEC 21	
03	D DEC 13	
04	E RESB 1	
05	POM1 RESB 1	
06	POM2 RESB 1	
07	START:	; start výpočtu
	MOV R1,[A]	; porovnání čísel A a B
09	MOV R2,[B]	
0B	MOV R0,POM1	
0D	CMP R1,R2	
0E	JG VETSI_R1a	
10	MOV [R0],R2	
11	JMP DALE_a	
13	VETSI_R1a: MOV [R0],R1	
14	DALE_a:	; větší z čísel do POM1

a) Pokračování příkladu

adresa paměti	symbolická instrukce nebo direktiva	poznámka
	MOV R1,[POM1]	; porovnání čísel POM1 a C
16	MOV R2,[C]	
18	MOV R0,POM2	
1A	CMP R1,R2	
1B	JG VETSI_R1b	
1D	MOV [R0],R2	
1E	JMP DALE_b	
20	VETSI_R1b: MOV [R0],R1	
21	DALE_b: MOV R1,[POM2]	; větší z čísel do POM2 ; porovnání čísel POM2 a D
23	MOV R2,[D]	
25	MOV R0,E	
27	CMP R1,R2	
28	JG VETSI_R1c	
2A	MOV [R0],R2	
2B	JMP DALE_c	
2D	VETSI_R1c: MOV [R0],R1	
2E	DALE_c: HALT	; větší z čísel do E
	END START	; START → IP

b) Řešení pomocí funkce:

...

{A, B, adrPOM1} → STACK

CALL MAX

{C, POM1, adrPOM2} → STACK

CALL MAX

PUSH {D, POM2, adrE} → STACK

CALL MAX

...

{r, q, adr_p} → STACK

MOV R0,[r]

PUSH R0

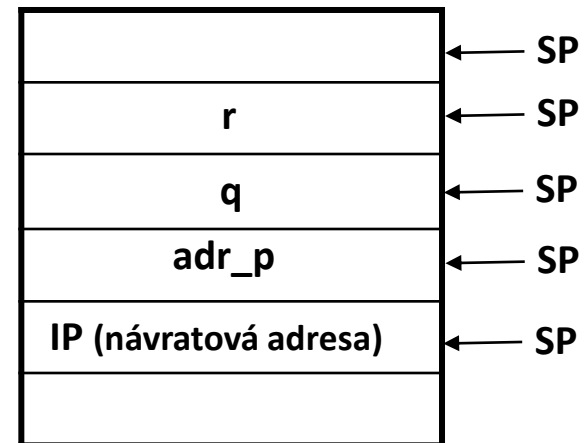
MOV R0,[q]

PUSH R0

MOV R0,p

PUSH R0

CALL MAX



b) Pokračování příkladu:

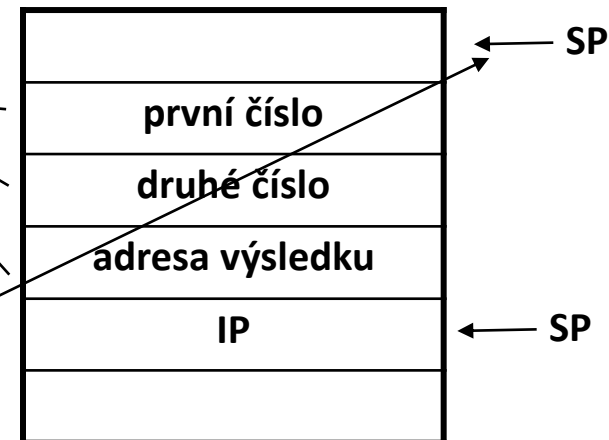
adresa paměti	symbolická instrukce nebo direktiva	poznámka
------------------	--	----------

ORG 0

00	A	DEC 56	; definice konstant
01	B	DEC -11	
02	C	DEC 21	
03	D	DEC 13	
04	E	RESB 1	
05	POM1	RESB 1	
06	POM2	RESB 1	

; procedura MAX předpokládá následující obsah zásobníku:

07	MAX:	MOV R0,[SP+1]
09		MOV R2,[SP+2]
0B		MOV R1,[SP+3]
0D		CMP R1,R2
0E		JG VETSI_R1
10		MOV [R0],R2
11		JMP DALE
13	VETSI_R1:	MOV [R0],R1
14	DALE:	RET 3

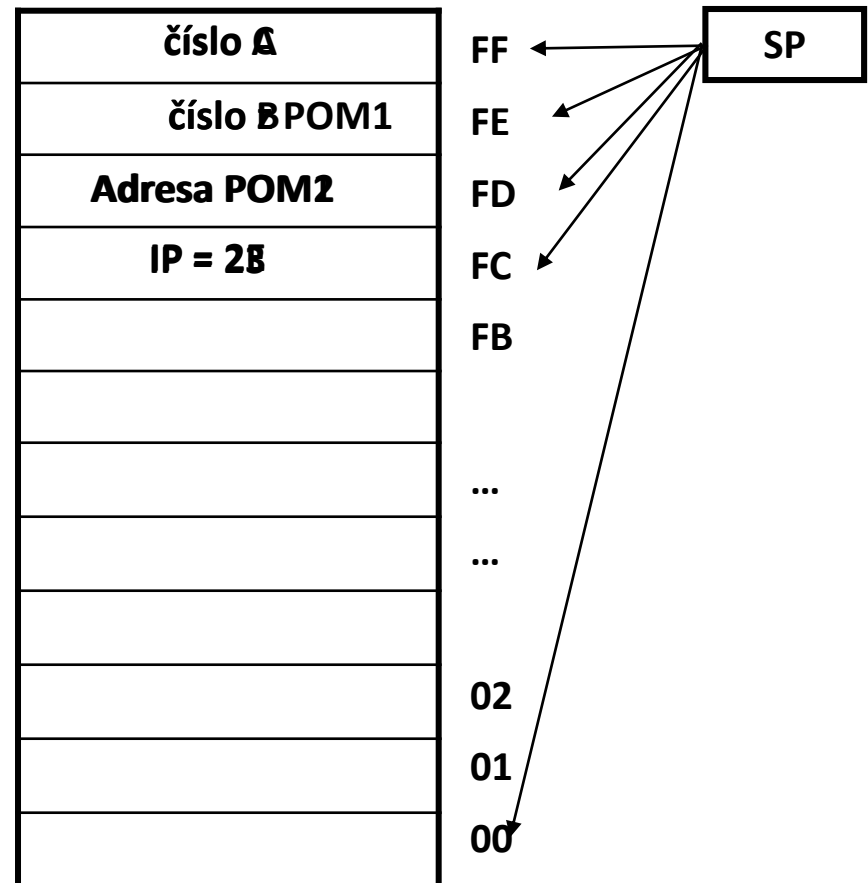


b) Pokračování příkladu

adresa symbolická instrukce poznámka paměti nebo
 direktiva

16	START:	MOV SP,0
18		MOV R0,[A]
1A		PUSH R0
1B		MOV R0,[B]
1D		PUSH R0
1E		MOV R0,POM1
20		PUSH R0
21		CALL MAX
23		MOV R0,[C]
25		PUSH R0
26		MOV R0,[POM1]
28		PUSH R0
29		MOV R0,POM2
2B		PUSH R0
2C		CALL MAX

zásobník

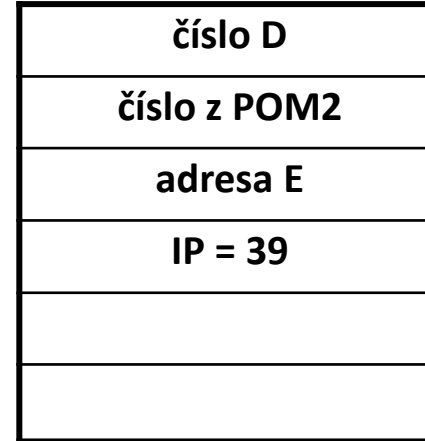


b) Pokračování příkladu

adresa paměti	symbolická instrukce nebo direktiva	poznámka
------------------	--	----------

2E	MOV R0, [D]	
30	MOV R1, [POM2]	
32	MOV R2, E	
34	PUSH R0	
35	PUSH R1	
36	PUSH R2	
37	CALL MAX	
39	HALT	
	END START	

zásobník



b) Řešení pomocí procedury, kdyby existovaly instrukce PUSH [adr] a PUSH adr:

START:

```
PUSH [A]  
PUSH [B]  
PUSH POM1  
CALL MAX
```

```
PUSH [C]  
PUSH [POM1]  
PUSH POM2  
CALL MAX
```

```
PUSH [D]  
PUSH [POM2]  
PUSH E  
CALL MAX
```

HALT

Ekvivalent v jazyce C:

```
    pom1 = max(a, b);  
    pom2 = max(c, pom1);  
    e = max(d, pom2);
```

Program s procedurami
je obvykle kratší,
ale mírně pomalejší.

Nová direktiva

name MACRO par

definice makra se jménem *name*
(*par* - seznam formálních parametrů)

...

... tělo makra

MEND

označení konce definice makra

c) Řešení pomocí makra:

symbolická instrukce
nebo direktiva

poznámka

ORG 0

MAX	MACRO P1,P2,P3	; počátek definice makra se jménem <i>max</i>
	LOCAL VETSI_R1,DALE	; definice lokálních návěští
	MOV R1,[P1]	
	MOV R2,[P2]	
	MOV R0,P3	
	CMP R1,R2	
	JG VETSI_R1	
	MOV [R0],R2	
	JMP DALE	
VETSI_R1:	MOV [R0],R1	
DALE:	MEND	; konec definice makra

A **DEC 56** **; definice konstant**

B **DEC -11**

C **DEC 21**

D **DEC 13**

E **RESB 1**

POM1 **RESB 1**

POM2 **RESB 1**

c) Pokračování příkladu

```

START:  MAX A,B,POM1           ;vlastní program
        MAX C,POM1,POM2
        MAX D,POM2,E
        HALT
        END START

```

Překlad výše uvedeného programu:

00	A	DEC 56	16		MOV R2,[POM1]
01	B	DEC -11	18		MOV R0,POM2
02	C	DEC 21	1A		CMP R1,R2
03	D	DEC 13	1B		JG @003
04	E	RESB 1	1D		MOV [R0],R2
05	POM1	RESB 1	1E		JMP @004
06	POM2	RESB 1	20	@003:	MOV [R0],R1
07	START:	MOV R1,[A]	21	@004:	
09		MOV R2,[B]			MOV R1,[D]
0B		MOV R0,POM1	23		MOV R2,[POM2]
0D		CMP R1,R2	25		MOV R0,E
0E		JG @001	27		CMP R1,R2
10		MOV [R0],R2	28		JG @005
11		JMP @002	2A		MOV [R0],R2
13	@001:	MOV [R0],R1	2B		JMP @006
14	@002:		2D	@005:	MOV [R0],R1
		MOV R1,[C]	2E	@006:	
					HALT

Reálný počítač – architektura Intel x86

Některé registry, instrukce a příznaky

31 ... 0	symbolická instrukce	operace
EAX	MOV reg1, reg2	přenesení do registru <i>reg1</i> hodnotu z registru <i>reg2</i>
EBX	MOV reg1, <i>imm</i>	přenesení do registru <i>reg1</i> konstantní hodnotu <i>imm</i>
ECX	ADD reg1, reg2	přičte k hodnotě v registru <i>reg1</i> hodnotu z registru <i>reg2</i> a výsledek uloží do <i>reg1</i> : $\text{reg1} = \text{reg1} + \text{reg2}$
EDX	ADD reg1, <i>imm</i>	přičte k hodnotě v registru <i>reg1</i> konstantní hodnotu <i>imm</i> a výsledek uloží do <i>reg1</i> : $\text{reg1} = \text{reg1} + \text{imm}$
ESP		
EBP		
ESI	SUB reg1, reg2	odečte od hodnoty v registru <i>reg1</i> hodnotu z registru <i>reg2</i> a výsledek uloží do <i>reg1</i> : $\text{reg1} = \text{reg1} - \text{reg2}$
EDI	SUB reg1, <i>imm</i>	odečte od hodnoty v registru <i>reg1</i> konstantní hodnotu <i>imm</i> a výsledek uloží do <i>reg1</i> : $\text{reg1} = \text{reg1} - \text{imm}$
registr příznaků EFLAGS		

C	Carry – přetečení bezznaménkové operace	O	Overflow – přetečení znaménkové operace
---	---	---	---