

# Registr příznaků, posuvy, rotace, násobení a dělení

ISU-cv04

**Ing. Jakub Husa**

Vysoké Učení Technické v Brně, Fakulta informačních technologií  
Božetěchova 1/2. 612 66 Brno - Královo Pole  
[ihusa@fit.vutbr.cz](mailto:ihusa@fit.vutbr.cz)



28. února 2023

## Registr příznaků

**Příznak (flag)** je pravdivostní hodnota poskytující informace o výsledku provedené operace:

- Příznaky jsou nastavovány jako vedlejší efekt **některých** instrukcí.
- Ovlivňují některé výpočty a umožňují provádět **podmíněné skoky** (viz. **cv05**).

Příznaky jsou uchovávány v registru **EFLAGS**:

- S registrem nelze manipulovat na přímo.

`mov eflags, 0 ;CHYBA - do EFLAGS nelze zapsat`

Konkrétními příznaky jsou například:

- **Overflow (OF)** – došlo k **přetečení**?
- **Carry (CF)** – došlo k **přenosu**?
- **Sign (SF)** – je výsledek **záporný**?
- **Zero (ZF)** – je výsledek **nula**?

TRANSFER		Flags									
Name	Comment	O	D	I	T	S	Z	A	P	C	
MOV	Move (copy)										
XCHG	Exchange										
SHL	Shift logical left (= SAL)	i				±	±	?	±	±	
SHR	Shift logical right	i				±	±	?	±	±	
SAL	Shift arithmetic left (= SHL)	i				±	±	?	±	±	
SAR	Shift arithmetic right	i				±	±	?	±	±	
RCL	Rotate left through Carry	i								±	
RCR	Rotate right through Carry	i								±	
ROL	Rotate left	i								±	
ROR	Rotate right	i								±	

ARITHMETIC		Flags									
Name	Comment	O	D	I	T	S	Z	A	P	C	
ADD	Add	±				±	±	±	±	±	
ADC	Add with Carry	±				±	±	±	±	±	
SUB	Subtract	±				±	±	±	±	±	
SBB	Subtract with borrow	±				±	±	±	±	±	
DIV	Divide (unsigned)	?				?	?	?	?	?	
IDIV	Signed Integer Divide	?				?	?	?	?	?	
MUL	Multiply (unsigned)	±				?	?	?	?	±	
IMUL	Signed Integer Multiply	±				?	?	?	?	±	
INC	Increment	±				±	±	±	±		
DEC	Decrement	±				±	±	±	±		
NEG	Negate (two-complement)	±				±	±	±	±	±	
CBW	Convert byte to word										
CWD	Convert word to double	±				±	±	±	±	±	
CWDE	Conv word extended double										

31	...	16	15 ... 8	7 ... 0
A ... Accumulator			AH	AL
			AX	
			EAX	
B ... Base			BH	BL
			BX	
			EBX	
C ... Counter			CH	CL
			CX	
			ECX	
D ... Data			DH	DL
			DX	
			EDX	

31	...	16	15	...	0
SP ... Stack Pointer			SP		
ESP					
BP ... Base Pointer			BP		
EBP					
SI ... Source Index			SI		
ESI					
DI ... Destination Index			DI		
EDI					
IP ... Instruction Pointer			IP		
EIP					
registr příznaků			FLAGS		
EFLAGS					

Vyzkoušejte si:

- Sledujte jak se budou měnit hodnoty příznaků v následujícím programu.

1	section .text	; registr AL		priznaky				
2	_main:	; DEC BIN		OF	CF	SF	ZF	
3		; -----		-----				
4	mov al, 126	; 126 01111110		?	?	?	?	
5	add al, 1	; 127 01111111		0	0	0	0	
6	add al, 1	; -128 10000000		1	0	1	0	
7	add al, 1	; -127 10000001		0	0	1	0	
8	add al, 126	; -1 11111111		0	0	1	0	
9	add al, 1	; 0 00000000		0	1	0	1	
10	add al, 1	; 1 00000001		0	0	0	0	
11	ret							

Instrukce **sčítání** a **odčítání** nastavují příznak **CF**:

- Příznak můžeme promítnout do dalšího výpočtu.
- Inkrementace a dekrementace příznak **CF** nenastavují.

**ADD** – add.

```
1 mov al, -1 ;AL= 11111111, CF= ?
2 add al, 1 ;AL= 00000000, CF= 1
3 add al, 1 ;AL= 00000001, CF= 0
```

**SUB** – subtract.

```
4 mov al, 0 ;AL= 00000000, CF= ?
5 sub al, 1 ;AL= 11111111, CF= 1
6 sub al, 1 ;AL= 11111110, CF= 0
```

**ADC** – add with carry.

```
7 mov al, -1 ;AL= 11111111, CF= ?
8 adc al, 1 ;AL= 00000000, CF= 1
9 adc al, 1 ;AL= 00000010, CF= 0
```

**SBB** – subtract with borrow.

```
10 mov al, 0 ;AL= 00000000, CF= ?
11 sbb al, 1 ;AL= 11111111, CF= 1
12 sbb al, 1 ;AL= 11111101, CF= 0
```

**INC** – increment.

```
13 mov al, -1 ;AL= 11111111, CF= ?
14 inc al ;AL= 00000000, CF= ?
```

**DEC** – decrement.

```
15 mov al, 0 ;AL= 00000000, CF= ?
16 dec al ;AL= 11111111, CF= ?
```

Vyzkoušejte si:

- Proveďte výpočet který současně nastaví příznaky OF, CF a ZF.
- Obsah registru EFLAGS zkontrolujte v debuggeru.

Vyzkoušejte si:

- Napište program který ze vstupu načte dvě 8b čísla, a provede jejich součet.
- Pokud výpočtu došlo k přenosu z nejvyššího řádu vypište jedničku.
- Pokud k přenosu nedošlo vypište nulu.

Například:

- 0, 0 => 0  
1, -1 => 1

# Posuvy a rotace



Bitové posuvy (SHL, SHR, SAL, SAR) umožňují pohybovat s bity uvnitř registru:

- Jako operandy uvádíme **co posouváme** a **o kolik bitů** (**konstanta** nebo **CL**).
- Bitový posuv o jednu pozici je v podstatě násobení nebo dělení dvěma.
- Při posunu **doleva** se prázdné bity v obou směrech doplňují **nulami**.
- Při posunu **doprava** se prázdné bity doplňují **nulami** nebo **prvním bitem**.

SHL – shift left.

```

1  mov  al,  15 ;AL = 00001111 = 15
2  shl  al,   1 ;AL = 00011110 = 30
3  mov  al, -16 ;AL = 11110000 = -16
4  shl  al,   1 ;AL = 11100000 = -32
    
```

SHR – shift right.

```

5  mov  al,  15 ;AL = 00001111 = 15
6  shr  al,   1 ;AL = 00000111 = 7
7  mov  al, -16 ;AL = 11110000 = -16
8  shr  al,   1 ;AL = 01111000 = 120
    
```

SAL – shift arithmetic left.

```

9  mov  al,  15 ;AL = 00001111 = 15
10 sal  al,   1 ;AL = 00011110 = 30
11 mov  al, -16 ;AL = 11110000 = -16
12 sal  al,   1 ;AL = 11100000 = -32
    
```

SAR – shift arithmetic right.

```

13 mov  al,  15 ;AL = 00001111 = 15
14 sar  al,   1 ;AL = 00000111 = 7
15 mov  al, -16 ;AL = 11110000 = -16
16 sar  al,   1 ;AL = 11111000 = -8
    
```

Bitové rotace (ROL, ROR, RCL, RCR) se podobají posuvům:

- Jako operandy uvádíme **co rotujeme** a **o kolik bitů** (**konstanta** nebo **CL**).
- Bity které registr opouštějí se do něj vracejí z druhé strany.
- Přenášený bit se buď **kopíruje do**, nebo **rotuje skrz** příznak **CF**.

**ROL** – rotate left.

```
1 mov al, 102 ;AL= 01100110, CF= ?
2 rol al, 1 ;AL= 11001100, CF= 0
3 rol al, 1 ;AL= 10011001, CF= 1
4 rol al, 1 ;AL= 00110011, CF= 1
```

**ROR** – rotate right.

```
5 mov al, 102 ;AL= 01100110, CF= ?
6 ror al, 1 ;AL= 00110011, CF= 0
7 ror al, 1 ;AL= 10011001, CF= 1
8 ror al, 1 ;AL= 11001100, CF= 1
```

**RCL** – rotate through carry left.

```
9 mov al, 102 ;AL= 01100110, CF= ?
10 rcl al, 1 ;AL= 1100110?, CF= 0
11 rcl al, 1 ;AL= 100110?0, CF= 1
12 rcl al, 1 ;AL= 00110?01, CF= 1
```

**RCR** – rotate through carry right.

```
13 mov al, 102 ;AL= 01100110, CF= ?
14 rcr al, 1 ;AL= ?0110011, CF= 0
15 rcr al, 1 ;AL= 0?011001, CF= 1
16 rcr al, 1 ;AL= 10?01100, CF= 1
```

Vyzkoušejte si:

- Do registru **EAX** nahrajte hodnotu **0x12345678**.
- Posuvy a rotacemi obsah registru upravte tak abyste z něj mohli vypsát hodnoty **0x12347856**, **0x48756123** a **0x23480075**.
- Pro výpis můžete použít funkci **WriteHex32**.

# Násobení

Čísla můžeme násobit **znaménkově** (IMUL) nebo **bez-znaménkově** (MUL):

- Při násobení čísel dochází ke **zvětšení** datového typu.
- Instrukce násobení má pouze jeden operand, číslo **kterým násobíme**.
- Číslo **které násobíme** a umístění **výsledku** jsou dány velikostí instrukce.

```
1 imul bl      ; AX = AL * BL
2 imul bx      ; DX:AX = AX * BX
3 imul ebx     ; EDX:EAX = EAX * EBX
```

Výsledek **16b** násobení je rozdělen mezi dva registry (DX:AX):

- Horní polovina bude uložena v **DX** a spodní polovina v **AX**.
- Poloviny výsledku můžeme spojit použitím posuvu nebo rotace.

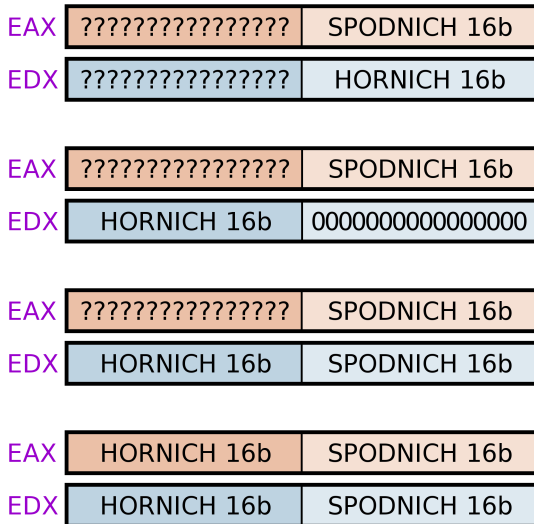
Procesor **neumí** násobit konstantou:

```
4 imul [X]      ;CHYBA - nelze urcit velkost instrukce
5 imul byte [X] ;ok      - AX = AL * [X]
6 imul byte 10   ;CHYBA - nelze nasobit konstantou
```

```

1 ;registr AX vynasob registrem BX
2     imul    bx
3
4
5
6 ;registr EDX posun o 16b doleva
7     shl     edx, 16
8
9
10
11 ;zkopiruj spodnich 16b z AX do DX
12     mov     dx,  ax
13
14
15
16 ;zkopiruj vseh 32b z EDX do EAX
17     mov     eax, edx

```



Vyzkoušejte si:

- Program ze vstupu načte dvě 16b čísla a vypíše jejich znaménkový násobek:

```
1 %include "rw32-2018.inc" ;knihovna pro vstup a vystup
2
3 section .text           ;kodovy segment
4 _main:
5     call ReadInt16       ;do AX nacti prvni vstup
6     mov  bx, ax          ;prvni vstup zkopirujeme do BX
7     call ReadInt16       ;do AX nacti druhy vstup
8
9     imul bx              ;DX:AX = AX * BX
10    shl  edx, 16          ;registr EDX posun o 16b doleva
11    mov  dx, ax           ;zkopiruj spodnich 16b vysledku z AX do DX
12    mov  eax, edx         ;zkopiruj vseh 32b vysledku z EDX do EAX
13
14    call WriteInt32       ;vypis EAX
15    ret
```

Vyzkoušejte si:

- Vytvořte si dvě 8b proměnné ( $X = 100$ ,  $Y = 200$ ) a jednu 16b proměnnou ( $Z = 300$ ).
- Bez-znaménkově spočítejte následující rovnici a vypište její výsledek:

$$f = X * Y * Z$$



**Dělení**

Čísla můžeme dělit **znaménkově** (IDIV) nebo **bez-znaménkově** (DIV):

- Při dělení čísel dochází ke **zmenšení** datového typu.
- Instrukce dělení má pouze jeden operand, číslo **kterým dělíme**.
- Číslo **které dělíme** a umístění **podílu** a **zbytku** jsou dány velikostí instrukce.

```
1 idiv bl      ; AL = AX / BL , AH = AX % BL
2 idiv bx      ; AX = DX:AX / BX , DX = DX:AX % BX
3 idiv ebx     ; EAX = EDX:EAX / EBX , EDX = EDX:EAX % EBX
```

Dělené **32b** číslo je umístěno ve dvou registrech (**DX:AX**):

- Horní polovina musí být uložena v **DX** a spodní polovina v **AX**.
- Do horní poloviny děleného čísla bud umístěn **zbytek**, do spodní **podíl**.
- Dělené číslo můžeme rozmístit posuvy a rotacemi, nebo **rozšířením**.

Pokud se **podíl** nebo **zbytek** nevejdou do registrů, nastane chyba:

- Pokus o **dělení nulou** způsobí chybu.
- Procesor **neumí** dělit konstantou.

```
4 idiv byte 10 ;CHYBA - nelze delit konstantou
```

Rozšíření (**CBW**, **CWD**, **CDQ**, **CWDE**) umožňuje **zvětšit** datový typ znaménkového čísla:

- Při počítání **bez znaménka** číslo nerozšiřujeme, ale vždy doplňujeme **nulami**.
- Instrukce rozšíření nepřijímají žádné operandy.

```

1  cbw      ; AL =>      AX   (Convert Byte to Word)
2  cwd      ; AX =>  DX:AX   (Convert Word to Dword)
3  cdq      ; EAX => EDX:EAX (Convert Dword to Qword)
4  cwde     ; AX =>      EAX  (Convert Word to Dword Extended)
    
```

**Kladné číslo** – rozšiřuje **nulami**.

```

5          ; |      AX = ???      |
6  mov  al, 15 ; |????????|00001111|
7          ; |AH = ???|AL = 15|
8          ; |-----|
9          ; |      AX = 15      |
10  cbw      ; |00000000|00001111|
11          ; |AH = 0|AL = 15|
    
```

**Záporné číslo** – rozšiřuje **jedničkami**.

```

12         ; |      AX = ???      |
13  mov  al,-16 ; |????????|11110000|
14         ; |AH = ???|AL = -16|
15         ; |-----|
16         ; |      AX = -16      |
17  cbw      ; |11111111|11110000|
18         ; |AH = -1|AL = -16|
    
```

Vyzkoušejte si:

- Program ze vstupu načte dvě 16b čísla, první z nich rozšíří, vydělí je se znaménkem, a vypíše jejich podíl:

```
1 %include "rw32-2018.inc" ;knihovna pro vstup a vystup
2
3 section .text           ;kodovy
4 _main:
5     call ReadInt16       ;do AX nacti delene cislo
6     mov  cx, ax          ;s delenym cislem uhneme do CX
7     call ReadInt16       ;do AX nacti delici cislo
8     mov  bx, ax          ;s delicim cislem uhneme do BX
9     mov  ax, cx          ;delene cislo vratime z CX do AX
10
11     cwd                 ;delenemu cislu rozsirime znamenko
12     idiv bx             ;cisla spolu podelime
13
14     call WriteInt16      ;vypis
15     ret
```

Vyzkoušejte si:

- Napište program který ze vstupu načte dvě 32b čísla (X a Y), a vypíše podíl Z, bez-znaménkově definovaný následující rovnicí.

$$Z = \frac{X * Y}{X + Y}$$

Například:

$$X = 100 \quad Y = 300$$

$$Z = \frac{100 * 300}{100 + 300} = \frac{30000}{400} = 75$$

Vyzkoušejte si:

- Vytvořte si čtyři 8b proměnné ( $K=10$ ,  $L=20$ ,  $M=2$ ,  $N=3$ ).
- Vytvořte si dvě neinicializované 16b proměnné ( $V$  a  $Z$ ).
- **Znaménkově** spočítejte výsledek následující rovnice, podíl uložte do proměnné  $V$ , zbytek do proměnné  $Z$ , a obsah obou proměnných zobrazte v **debuggeru**.

$$f = \frac{K + L + 100}{M * N}$$

Například:

$$f = \frac{10 + 20 + 100}{2 * 3} = \frac{130}{6}$$

$$V = 21 \quad Z = 4$$