Logické instrukce, programový čítač, příznaky a podmínky

ISU-cv05

Ing. Jakub Husa Ph.D.

Vysoké učení technické v Brně, Fakulta informačních technologií
Božetěchova 1/2. 612 66 Brno - Královo Pole
ihusa@fit.vut.cz

14. března 2025

Logické instrukce

Logické instrukce (AND, OR, XOR, NOT) implementují bitové operace (&, |, ^, ~):

- Pozor, nejsou to logické operátory (&&, ||, !), tak jak je známe z jazyka C!

```
1  and  eax, ebx ;EAX = EAX & EBX  ;EAX a zaroven EBX  ;logicky soucin
2  or   eax, ebx ;EAX = EAX | EBX  ;EAX a   nebo  EBX  ;logicky soucet
3  xor  eax, ebx ;EAX = EAX ^ EBX  ;bud EAX nebo  EBX  ;exkluzivni soucet
4  not  eax      ;EAX = ~EAX       ;invertuj EAX       ;logicka negace
```

Bitové operace umožňují nastavit (maskovat) hodnotu konkrétních bitů:

- Pozor na rozdíl mezi logickou (NOT) a aritmetickou (NEG) negací (viz. cv02).

```
5  ;AND umoznuje vynutit nuly
6  mov al, 0b00110011
7  and al, 0b00001111
8  ;   AL=   00000011
9
10 ;OR  umoznuje vynutit jednicky
11 mov bl, 0b00110011
12 or  bl, 0b00001111
13 ;   BL=   00111111
```

```
14 ;XOR umoznuje vynutit zmenu
15 mov cl, 0b00110011
16 xor cl, 0b00001111
17 ;   CL=   00111100
18
19 ;NOT invertuje vsechny bity
20 mov dl, 0b00110011
21 not dl
22 ;   DL=   11001100
```

"To be, or not to be, that is the question:"

Shakespeare – Hamlet (Act 3, Scene 1)

```
%include 'rw32.inc'          ;knihovna pro vstup a vystup

section .text                ;kodovy segment
main:
    mov   eax, 0x2B          ; EAX = 2B
    not   eax                ; EAX = not 2B
    or    eax, 0x2B          ; EAX = 2B or not 2B
    call WriteInt32NewLine   ; vypis EAX
    ret
```

"Negative one — answereth the processor."

Intel Architecture Software Developer's Manual
(Chapter 3, pages 457–460)

Vyzkoušejte si:

- Vytvořte si dostatečně dlouhé pole a načtěte do něj řetězec tří znaků.
- Předpokládejte že načtené znaky jsou nějaká malá a velká písmena.
- Hodnoty znaků řetězce upravte tak, aby jeho první písmeno bylo malé, druhé velké, třetí změnilo svoji velikost, a upravený řetězec vypište.

Například:

- abc => aBC
  ABC => aBc

'A' = 65 = 01000001
'B' = 66 = 01000010
'C' = 67 = 01000011

'a' = 97 = 01100001
'b' = 98 = 01100010
'c' = 99 = 01100011

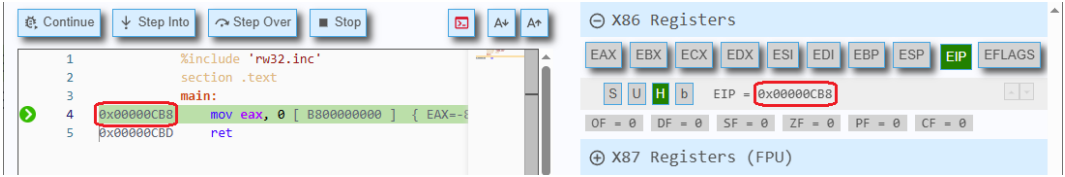
Nápověda:

- Velká a malá písmena se od sebe v tabulce ASCII liší hodnotou jednoho bitu.
- Před voláním funkcí ReadString a WriteString musíme do EBX a ECX nastavit počet načítaných a vypisovaných znaků, a do indexových registrů EDI a ESI nastavit adresu vypisovaného řetězce (viz. cv03).

Programový čítač

Programový čítač obsahuje adresu aktuální instrukce:

- Hodnota čítače je uložena v registru EIP (Extended Instruction Pointer).
- Jeho počáteční hodnotou je adresa první instrukce z funkce main.



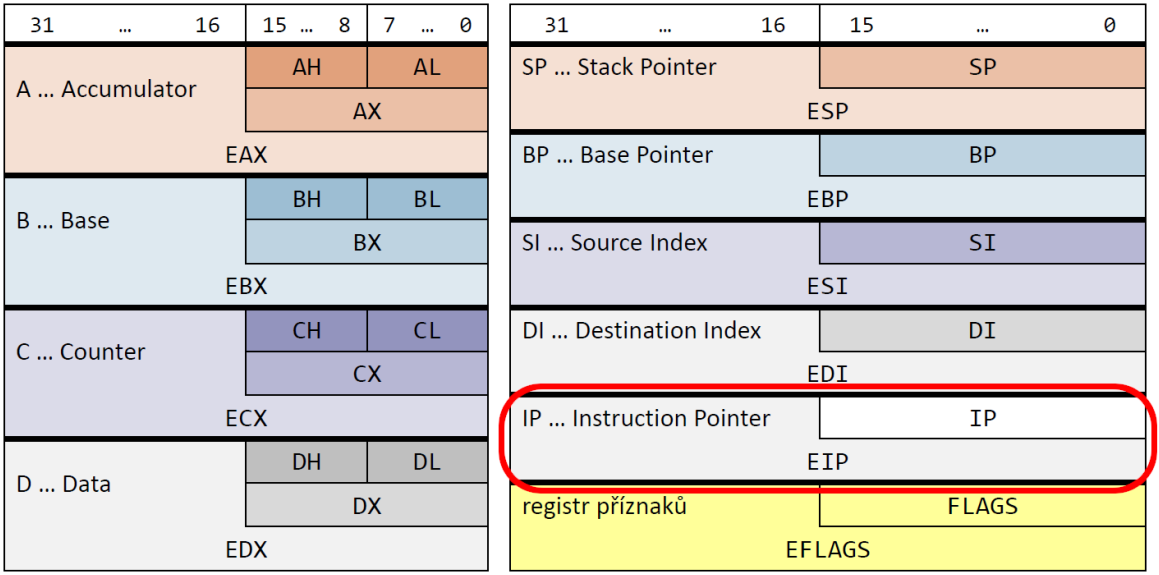
- Při provedení libovolné instrukce se do EIP, jako vedlejší efekt, automaticky nahraje adresa instrukce z následujícího řádku.

S obsahem registru EIP nelze manipulovat běžnými instrukcemi:

- Hodnotu čítače můžeme změnit skokem (Jump) na nějaké návěští (Label).
- Skoky a návěští umožňují psát podmínky, cykly (viz. cv07) a funkce (viz. cv08).

```
1    mov   eip, 0          ; CHYBA - do EIP nelze zapsat instrukci MOV
2    mov   eip, .label     ; CHYBA - do EIP nelze zapsat instrukci MOV
3    jmp   0               ; CHYBA - skok na neplatnou adresu
4    jmp   .label          ; do EIP nahraj adresu instrukce na navesti .LABEL
```

Registry procesoru - programový čítač



Návěští

Návěští (Label) v kódovém segmentu označuje adresu následující instrukce:

- Návěští vytvoříme pomocí identifikátoru následovaného dvojtečkou (:).
- Názvy návěští jsou case-sensitive – na psaní malých a velkých písmen záleží!

Pokud identifikátor začíná tečkou (.) návěští je lokální, jinak je globální:

- Globální návěští musejí být unikátní v rámci celého programu.
- Lokální návěští musejí být unikátní v rámci jednoho globálního návěští.

```
1  section .text    ;kodovy segment
2  main:            ; globalni  navesti  jmenem   MAIN  (nesmi se opakovat)
3    .if:           ; lokalni   navesti  jmenem   .IF   (muze  se opakovat)
4      mov eax, 0   ; instrukce oznacena navestim .IF
5    .else:         ; lokalni   navesti  jmenem   .ELSE (muze  se opakovat)
6      mov ebx, 0   ; instrukce oznacena navestim .ELSE
7      ret
8
9  foo:             ; globalni  navesti  jmenem   FOO   (nesmi se opakovat)
10   .if:           ; lokalni   navesti  jmenem   .IF   (muze  se opakovat)
11     mov ecx, 0   ; instrukce oznacena navestim .IF
12   .else:         ; lokalni   navesti  jmenem   .ELSE (muze  se opakovat)
13     mov edx, 0   ; instrukce oznacena navestim .ELSE
14     ret
```

Nepodmíněný skok

Instrukce JMP (JuMP) je nepodmíněný skok který se vždy provede:

- Jejím operandem je návěští instrukce jejíž adresa se nahraje do registru EIP.

Nepodmíněný skok dolů umožňuje část kódu vynechat:

- Například – v podmínce chceme po provedení těla if vynechat tělo else.

```
1     mov   eax, 10              ;EAX = 10
2     jmp   .label               ;skoc na navesti .LABEL
3     mov   eax, 20              ;EAX = 20 (bude preskoceno)
4   .label:                      ;lokalni navesti jmenem .LABEL
5     call  WriteInt32NewLine    ;vypis EAX (hodnota 10)
```

Nepodmíněný skok nahoru umožňuje část kódu opakovat:

- Například – v cyklu while se po provedení těla chceme vrátit k jeho hlavičce.
- Pozor, nepodmíněný skok nahoru může způsobit nekonečný cyklus!

```
6     mov   eax, 0               ;EAX = 0
7   .label:                      ;lokalni navesti jmenem .LABEL
8     call  WriteInt32NewLine    ;vypis EAX
9     inc   eax                  ;EAX = EAX + 1
10    jmp   .label               ;skoc na navesti .LABEL
```

Příznaky

Příznaky

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

- Příznaky jsou nastavovány jako vedlejší efekt provedení některých instrukcí.
- ± – příznak se nastaví podle výsledku.
  ' ' – příznak se nenastaví.
  ? – příznak bude nedefinovaný.

Příznaky se ukládají do registru EFLAGS:

- S obsahem příznakového registru nelze manipulovat běžnými instrukcemi.

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

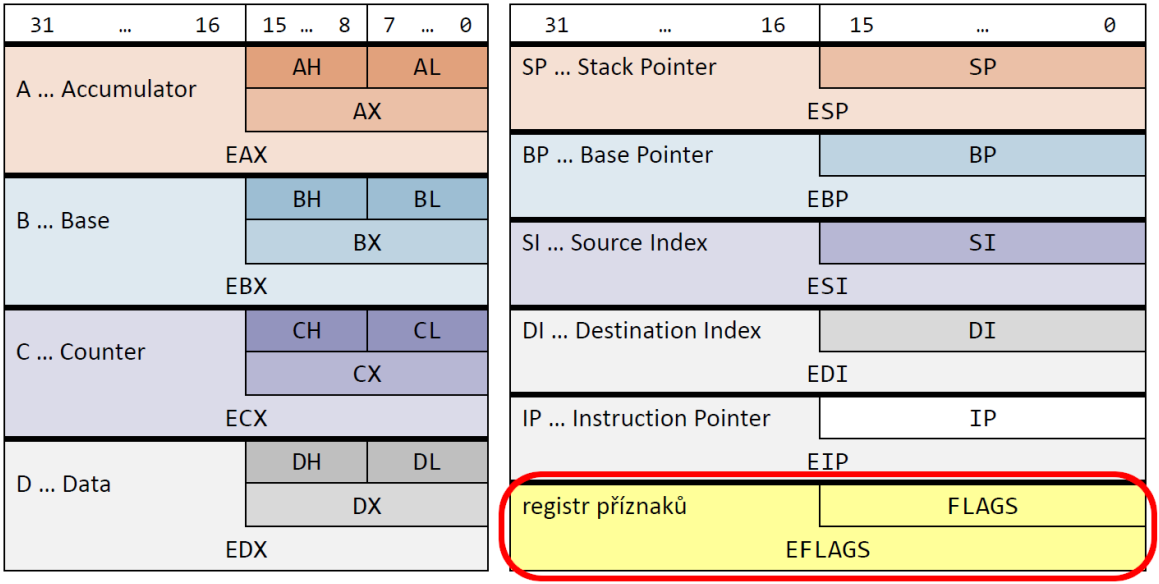
V ISU nás zatím budou zajímat jen příznaky:

- 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*								±

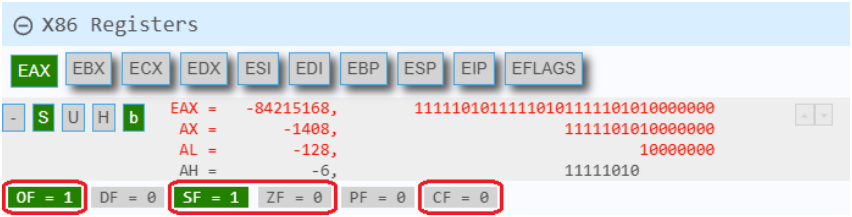
ARITHMETIC		Flags							
Name	Comment	O	D	I	T	S	Z	A	P	C
ADD	Add	±				±	±	±	±	±
ADC	Add with Carry	±				±	±	±	±	±
SUB	Subtract	±				±	±	±	±	±
SBB	Subtract with borrow	±				±	±	±	±	±
DIV	Divide (unsigned)	?				?	?	?	?	?
IDIV	Signed Integer Divide	?				?	?	?	?	?
MUL	Multiply (unsigned)	±				?	?	?	?	±
IMUL	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									

Registry procesoru - registr příznaků



Sledujte hodnoty příznaků při provádění následujících instrukcí:

```
section .text          ;  registr AL     |    priznaky    |
main:                  ;  DEC    BIN     | OF CF SF ZF    |
                       ;-----------------|----------------|
    mov   al, 126      ;  126 01111110   |  ?  ?  ?  ?    |
    add   al,   1      ;  127 01111111   |  0  0  0  0    |
    add   al,   1      ; -128 10000000   |  1  0  1  0    |
    add   al,   1      ; -127 10000001   |  0  0  1  0    |
    add   al, 126      ;   -1 11111111   |  0  0  1  0    |
    add   al,   1      ;    0 00000000   |  0  1  0  1    |
    add   al,   1      ;    1 00000001   |  0  0  0  0    |
    ret
```



Podmíněné skoky

Podmíněný skok je skok který se provede pouze při splnění nějaké podmínky:

- Pokud podmínka splněna nebyla, program pokračuje na dalším řádku.
- Pro každou ze všech možných podmínek existuje samostatná instrukce.

Skok může být podmíněn hodnotou nějakého konkrétního příznaku (JO, JC, JS, JZ):

```
1 jo    .label ;skoc pokud OF == 1 (Jump if Overflow)
2 jc    .label ;skoc pokud CF == 1 (Jump if Carry)
3 js    .label ;skoc pokud SF == 1 (Jump if Sign)
4 jz    .label ;skoc pokud ZF == 1 (Jump if Zero)
```

Podmínka skoku může být negovaná (JNO, JNC, JNS, JNZ):

```
5 jno   .label ;skoc pokud OF == 0 (Jump if Not Overflow)
6 jnc   .label ;skoc pokud CF == 0 (Jump if Not Carry)
7 jns   .label ;skoc pokud SF == 0 (Jump if Not Sign)
8 jnz   .label ;skoc pokud ZF == 0 (Jump if Not Zero)
```

Program ze vstupu načte dvě bez-znaménková 8b čísla (X a Y),
a pokud Y není nula tak spočítá a vypíše jejich podíl:

```
%include 'rw32.inc'          ;knihovna po vstup a vystup

section .text                ;kodovy segment
main:
    call  ReadUInt8NewLine   ; AL = X            ;nacti delenec
    mov   bl, al             ; AL = X, BL = X    ;zkopiruj ho do BL
    call  ReadUInt8NewLine   ; AL = Y, BL = X    ;nacti delitel
    xchg  bl, al             ; AL = X, BL = Y    ;vymen delenec a delitel
    movzx ax, al             ; bez-znamenkove rozsireni nulami z 8b na 16b
                             ; AX = X, BL = Y
    add   bl, 0              ; pomoci souctu nastav priznaky
                             ; AX = X, BL = Y
    jz    .skip              ; pokud ZF==1 tak skoc na navesti .SKIP
    div   bl                 ; 8b deleni bez znamenka
                             ; AL = X/Y, AH = X%Y
    call  WriteUInt8NewLine  ; vypis podil
  .skip:                     ; navesti jmenem .SKIP
    ret
```

Vyzkoušejte si:

- Ze vstupu si načtěte dvě 8b znaménková čísla a vypište jejich součet.
- Program upravte tak, aby se výpis přeskočil pokud při výpočtu došlo k přetečení.
- Program upravte tak, aby vypisoval absolutní hodnotu součtu.

Například:

- 10, 20 => 30
  100, 50 =>
  10, -20 => 10

Nápověda:

- Podle příznaku přetečení (Overflow) se rozhodují skoky JO a JNO.
- Podle příznaku záporného čísla (Sign) se rozhodují skoky JS a JNS.
- Absolutní hodnotu implementujeme (ne)přeskočením aritmetické negace.

Podmínky

Aritmetické porovnání

Příznaky můžeme nastavit instrukcí aritmetického porovnávání CMP (CoMPare):

- Spočítá aritmetický rozdíl (SUB), ale nezapíše výsledek, pouze nastaví příznaky.

```
%include 'rw32.inc'              ;knihovna pro vstup a vystup
section .data                    ;inicializovany segment
    jeDeset    db "Nactene cislo je deset", 10, 0     ; prvni retezec
    neniDeset  db "Nactene cislo neni deset", 10, 0   ; druhy retezec
section .text                    ;kodovy segment
main:
    mov  esi, jeDeset            ; do ESI nastav prvni retezec
    call ReadInt8NewLine         ; do AL nacti 8b cislo se znamenkem
                                 ;   (instrukce call priznaky nenastavuje)
    cmp  al, 10                  ; nactene cislo porovnej s desitkou
    jz   .skip                   ; pokud ZF==1 tak skoc na navesti .SKIP
    mov  esi, neniDeset          ; jinak, zmen vypisovany retezec
  .skip:
    call WriteStringASCIIZ       ; vypis retezec z adresy nastavene v ESI
    ret
```

Aritmetické porovnání příznaky připraví pro použití aritmetických skoků:

- Aritmetické skoky se rozhodují podle několika příznaků současně, tak aby podmínka odpovídala jednomu z relačních operátorů (>, >=, <, <=, ==, !=).

Aritmetické skoky

```
1  cmp  eax, ebx; porovnani umozni provest nasledujici aritmeticke skoky
2
3  je   .label  ; EAX == EBX  Jump if Equal     (se znamenkem i bez znamenka)
4  jne  .label  ; EAX != EBX  Jump if Not Equal (se znamenkem i bez znamenka)
5
6  jg   .label  ; EAX >  EBX  Jump if Greater                  (se znamenkem)
7  jl   .label  ; EAX <  EBX  Jump if Lesser                   (se znamenkem)
8  jge  .label  ; EAX >= EBX  Jump if Greater or Equal         (se znamenkem)
9  jle  .label  ; EAX <= EBX  Jump if Lesser  or Equal         (se znamenkem)
10 jng  .label  ; EAX <= EBX  Jump if Not Greater             (se znamenkem)
11 jnl  .label  ; EAX >= EBX  Jump if Not Lesser              (se znamenkem)
12 jnge .label  ; EAX <  EBX  Jump if Not Greater or Equal    (se znamenkem)
13 jnle .label  ; EAX >  EBX  Jump if Not Lesser  or Equal    (se znamenkem)
14
15 ja   .label  ; EAX >  EBX  Jump if Above                   (bez znamenka)
16 jb   .label  ; EAX <  EBX  Jump if Below                   (bez znamenka)
17 jae  .label  ; EAX >= EBX  Jump if Above or Equal          (bez znamenka)
18 jbe  .label  ; EAX <= EBX  Jump if Below or Equal          (bez znamenka)
19 jna  .label  ; EAX <= EBX  Jump if Not Above               (bez znamenka)
20 jnb  .label  ; EAX >= EBX  Jump if Not Below               (bez znamenka)
21 jnae .label  ; EAX <  EBX  Jump if Not Above or Equal      (bez znamenka)
22 jnbe .label  ; EAX >  EBX  Jump if Not Below or Equal      (bez znamenka)
```

Podmínky typu if-else

Podmínku typu if-else zapíšeme kombinací podmíněných a nepodmíněných skoků:

- Pokud podmínka platí, skočíme na návěští then.
- Pokud podmínka neplatí, skočíme na návěští else.
- Po provedení těla then skočíme na návěští end.

Protože při nesplnění podmínky program automaticky přejde na další řádek, tak některá návěští a skoky z programu můžeme vynechat:

- Například – podmínka porovná jestli EAX a EBX mají stejnou (Equal) hodnotu:

```
1   .if:              ; hlavicka podminky .IF              (muzeme vynechat)
2     cmp  eax, ebx   ; aritmeticke porovnani
3     je   .then      ; pokud EAX==EBX skoc na .THEN       (muzeme vynechat)
4     jne  .else      ; pokud EAX!=EBX skoc na .ELSE
5   .then:            ; navesti .THEN                      (muzeme vynechat)
6     ;prvni telo
7     jmp  .end       ; po provedeni tela skoc na .END
8   .else:            ; navesti .ELSE
9     ;druhe telo
10    jmp  .end       ; po provedeni tela skoc na .END     (muzeme vynechat)
11  .end:             ; konec podminky if-else
```

Podmínky typu if-else-if

Zřetězenou podmínku typu if-else-if vytvoříme několika podmíněnými skoky:

- Příkaz bude mít několik těl – každé z nich s vlastní podmínkou.
- Skoky nemění hodnotu příznaků – porovnávání nemusíme opakovat.

Vyzkoušejte si:

- Program porovná jestli je EAX větší (Greater), menší (Lesser), nebo rovno (Equal) 0.

```
1   .if:
2     cmp  eax, 0          ; aritmetickym porovnanim nastavime priznaky
3     jg   .then           ; pokud plati prvni podminka skocime na .THEN
4     jl   .elseif         ; pokud plati druha podminka skocime na .ELSEIF
5     je   .else           ; pokud plati treti podminka skocime na .ELSE
6   .then:
7     ;cislo je kladne     ; telo .THEN
8     jmp  .end            ; po provedeni tela skocime na konec
9   .elseif:
10    ;cislo je zaporne    ; telo .ELSEIF
11    jmp  .end            ; po provedeni tela skocime na konec
12  .else:
13    ;cislo je nula       ; telo .ELSE
14    jmp  .end            ; po provedeni tela skocime na konec
15  .end:
```

Složené podmínky

Složenou podmínku vytvoříme pomocí několika skoků na stejné tělo:

- Pokud stačí aby platila jedna podmínka (||), dva skoky povedou na then.
- Pokud musejí platit obě podmínky (&&), dva skoky povedou na else.

Je EAX menší jak 10 nebo větší jak 20?

```
.if:
  cmp  eax, 10 ;nastavime priznaky
  jl   .then   ;pokud plati prvni
  cmp  eax, 20 ;nastavime priznaky
  jg   .then   ;pokud plati druha
  jmp  .else   ;pokud neplatily
.then:
  ;telo THEN   ;po provedeni tela
  jmp  .end    ;skocime na konec
.else:
  ;telo ELSE   ;po provedeni tela
  jmp  .end    ;skocime na konec
.end:
```

Je EAX větší jak 10 a menší jak 20?

```
.if:
  cmp  eax, 10 ;nastavime priznaky
  jng  .else   ;pokud neplati
  cmp  eax, 20 ;nastavime priznaky
  jnl  .else   ;pokud neplati
  jmp  .then   ;pokud platily obe
.then:
  ;telo THEN   ;po provedeni tela
  jmp  .end    ;skocime na konec
.else:
  ;telo ELSE   ;po provedeni tela
  jmp  .end    ;skocime na konec
.end:
```

Program ze vstupu načte dvě 8b čísla a vypíše to větší z nich:

- Pro znaménková čísla.

```
%include 'rw32.inc'
section .text
main:
    call ReadInt8NewLine
    mov  bl, al
    call ReadInt8NewLine
  .if:            ;zacatek podminky
    cmp  al, bl   ;porovnej AL a BL
    jg   .then    ;pokud AL >  BL
    jng  .else    ;pokud AL <= BL
  .then:          ;telo then
    call WriteInt8NewLine
    jmp  .end     ;skoc na konec
  .else:          ;telo else
    xchg al, bl   ;vymen AL a BL
    call WriteInt8NewLine
    jmp  .end     ;skoc na konec
  .end:           ;konec podminky
    ret
```

- Pro bez-znaménková čísla.

```
%include 'rw32.inc'
section .text
main:
    call ReadUInt8NewLine
    mov  bl, al
    call ReadUInt8NewLine
  .if:            ;zacatek podminky
    cmp  al, bl   ;porovnej AL a BL
    ja   .then    ;pokud AL >  BL
    jna  .else    ;pokud AL <= BL
  .then:          ;telo then
    call WriteUInt8NewLine
    jmp  .end     ;skoc na konec
  .else:          ;telo else
    xchg al, bl   ;vymen AL a BL
    call WriteUInt8NewLine
    jmp  .end     ;skoc na konec
  .end:           ;konec podminky
    ret
```

Vyzkoušejte si:

- Vytvořte si čtyři 16b proměnné (X, Y a V, Z).
- Znaménkově spočítejte rovnici, tak aby se to větší z čísel X a Y dělilo tím menším.
- Podíl uložte do V, zbytek do Z, a jejich obsah si zkontrolujte v debuggeru.
- Pokud rovnici nelze spočítat, program se ukončí a proměnné V a Z budou nula.
- Zamyslete se nad tím které tři instrukce bychom museli změnit, pokud bychom úlohu počítali bez-znaménkově.

$$\frac{X}{Y} \quad \text{nebo} \quad \frac{Y}{X}$$

Například:

$X = 100$	$X = -3$	$X = 100$
$Y = -3$	$Y = 100$	$Y = 0$
$V = -33$	$V = -33$	$V = 0$
$Z = 1$	$Z = 1$	$Z = 0$

Nápověda:

- Při 16b dělení má dělenec 32b a musí být umístěn v registrech DX:AX (viz. cv04).

Logické porovnání

Příznaky můžeme nastavit také instrukcí logického porovnávání TEST (TEST):

- Spočítá logický součin (AND), ale nezapíše výsledek, pouze nastaví příznaky.
- Instrukci můžeme použít pro testování hodnoty konkrétního bitu.
- Pokud je testovaný bit nastaven na jedničku, tak ZF = 0, jinak ZF = 1.

```
%include 'rw32.inc'              ;knihovna pro vstup a vystup
section .data                    ;inicializovany segment
    jeZaporne    db "Nactene cislo je zaporne", 10, 0     ; prvni retezec
    neniZaporne db "Nactene cislo neni zaporne", 10, 0    ; druhy retezec
section .text                    ;kodovy segment
main:
    mov  esi, jeZaporne          ; do ESI nastav prvni retezec
    call ReadInt8NewLine         ; do AL nacti 8b cislo se znamenkem
                                 ;   (instrukce call priznaky nenastavuje)
    test  al, 0b10000000         ; otestuj prvni bit nacteneho cisla
                                 ;   (pokud je nactene cislo zaporne)
                                 ;   (tak zacinalo jednickou, a ZF = 0)
    jnz  .skip                  ; pokud ZF==0 tak skoc na navesti .SKIP
    mov  esi, neniZaporne        ; jinak, zmen vypisovany retezec
  .skip:
    call WriteStringASCIIZ       ; vypis retezec z adresy nastavene v ESI
    ret
```

Na procvičení

Vyzkoušejte si:

- Vytvořte si dvě inicializované 32b proměnné (X a Y).
- Zkontrolujte hodnoty obou proměnných a vypište řetězec "obe cisla jsou suda" nebo "alespon jedno cislo je liche".
- Sudá (even) a lichá (odd) čísla poznáme podle hodnoty jejich nejnižšího bitu. Nebo (složitěji) podle hodnoty zbytku po dělení dvěma.

Například:

- 1000000, 2000000 => obe cisla jsou suda
  1000001, 2000000 => alespon jedno cislo je liche
  1000000, 2000001 => alespon jedno cislo je liche
  1000001, 2000001 => alespon jedno cislo je liche

Nápověda:

- Pro logické porovnávání používáme instrukci TEST, která příznaky nastaví jako by byl proveden logický součin (&).