Logické instrukce, programový čítač a podmínky

ISU-cv05

Ing. Jakub Husa

Vysoké Učení Technické v Brně, Fakulta Informačních Technologií
Božetěchova 1/2. 612 66 Brno - Královo Pole
ihusa@fit.vut.cz

6. března 2024

Logické instrukce

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

- Bitové operace nám umožňují nastavit (maskovat) hodnotu jednotlivých bitů.
- Pozor, nejde o logické operátory (&&, ||, !) které znáte z jazyka C!

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

Vyzkoušejte si:

```
5  ;AND vynucuje nulu
6  mov al, 0b00110011 ;AL = 00110011
7  and al, 0b00001111 ;AL = 00000011
8
9  ;OR  vynucuje jednicku
10 mov bl, 0b00110011 ;BL = 00110011
11 or  bl, 0b00001111 ;BL = 00111111
```

```
12 ;XOR vynucuje zmenu
13 mov cl, 0b00110011 ;CL = 00110011
14 xor cl, 0b00001111 ;CL = 00111100
15
16 ;NOT invertuje bity
17 mov dl, 0b00110011 ;DL = 00110011
18 not dl             ;DL = 11001100
```

Vyzkoušejte si:

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

Například:

- abc => aBC
  ABC => aBc

Programový čítač

Programový čítač

Programový čítač obsahuje adresu instrukce která má být procesorem provedena:

- Hodnota čítače se ukládá do registru EIP (Extended Instruction Pointer).
- Jeho počáteční hodnotou je adresa první instrukce z funkce main.
- Při provedení libovolné instrukce se do něj (jako vedlejší efekt) automaticky nahraje adresa instrukce z následujícího řádku.
- S obsahem registru EIP nelze manipulovat běžnými instrukcemi.

```
1     mov  eip, 0     ;CHYBA - do registru EIP nelze zapsat
```

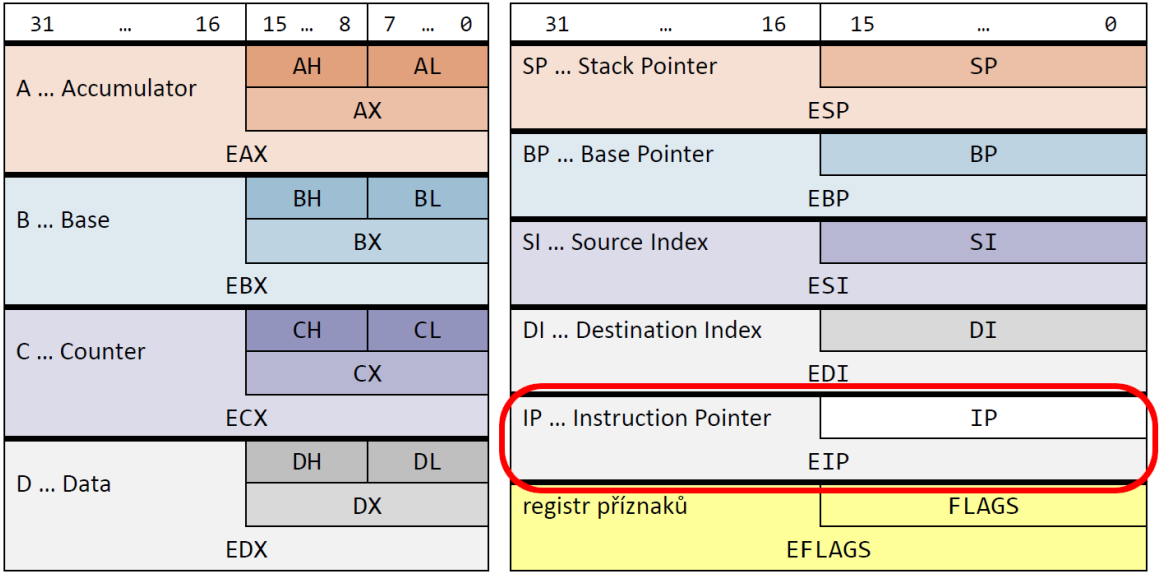
Hodnotu čítače můžeme změnit skokem (Jump) na nějaké návěští (Label):

- Návěští vytvoříme pomocí identifikátoru následovaného dvojtečkou (:).

```
2 label:              ;navesti jmenem LABEL
3     mov  eax, 10    ;instrukce oznacena navestim LABEL
```

- Názvy návěští jsou case-sensitive – na psaní malých a velkých písmen záleží!
- Návěští nám umožňují implementovat funkce, podmínky a cykly.

Registry procesoru - programový čítač



Nepodmíněný skok

Nepodmíněný skok (JMP) je skok který se provede vždy:

- Operandem skoku je návěští instrukce jejíž adresa se má nahrát do registru EIP.
- Nepodmíněný skok vpřed (dolů) nám umožní část kódu vynechat.

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

- Nepodmíněný skok vzad (nahoru) vytvoří nekonenčý cyklus.

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

- Pozor, skokem **NESMÍME** přejít na návěští funkce (viz. cv08)!

```
11   jmp   main                 ;CHYBA - skok na navesti funkce main
```

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:

```
1  %include 'rw32.inc'          ;knihovna po vstup a vystup
2
3  section .text                ;kodovy segment
4  main:
5      call ReadUInt8NewLine ; AL = X             ;nacti delenec
6      mov  bl, al           ; AL = X, BL = X     ;zkopiruj ho do BL
7      call ReadUInt8NewLine ; AL = Y, BL = X     ;nacti delitel
8      xchg bl, al           ; AL = X, BL = Y     ;vymen delenec a delitel
9      mov  ah, 0            ; AX = X, BL = Y     ;delenec rozsir z 8b na 16b
10                           ;   (bez-znamenkove rozsireni nulami)
11     add  bl, 0            ; AX = X, BL = Y     ;nastav priznaky
12                           ;   (jako vedlejsi efekt instrukce ADD)
13     jz   skip             ; pokud ZF==1 tak skoc na navesti SKIP
14                           ;   (vysledek souctu byl nula)
15     div  bl               ; AL = X/Y, AH = X%Y, BL = Y
16                           ;   (8b deleni bez znamenka)
17     call WriteUInt8NewLine; vypis podil
18   skip:                   ; navesti jmenem SKIP
19     ret
```

Vyzkoušejte si:

- Ze vstupu načtěte dvě znaménková 8b čísla (X a Y) a vypište jejich součet.
- Program upravte tak aby se výpis přeskočil pokud při součtu došlo k přetečení.
- Program upravte tak aby se znaménko záporného výsledku obracelo na kladné.

Například:

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

Podmínky

Příznaky můžeme nastavit instrukcemi pro porovnávání (CMP, TEST):

- Instrukce provedou výpočet ale nezapíší výsledek – pouze nastaví příznaky.

```
1 cmp  eax, ebx  ; nastav priznaky  EAX - EBX  (aritmeticke porovnani)
2 test eax, ebx  ; nastav priznaky  EAX & EBX  (  logicke   porovnani)
```

Aritmetické porovnání (CMP) příznaky připraví pro aritmetické skoky:

- Pro znaménková čísla.

```
3  je   lbl ;EAX == EBX (Equal)
4  jg   lbl ;EAX >  EBX (Greater)
5  jl   lbl ;EAX <  EBX (Lesser)
6  jge  lbl ;EAX >= EBX (.. or Equal)
7  jle  lbl ;EAX <= EBX (.. or Equal)
8
9  jne  lbl ;EAX != EBX (Not Equal)
10 jng  lbl ;EAX <= EBX (Not Greater)
11 jnl  lbl ;EAX >= EBX (Not Lesser)
12 jnge lbl ;EAX <  EBX (.. or Equal)
13 jnle lbl ;EAX >  EBX (.. or Equal)
```

- Pro bez-znaménková čísla.

```
14 je   lbl ;EAX == EBX (Equal)
15 ja   lbl ;EAX >  EBX (Above)
16 jb   lbl ;EAX <  EBX (Below)
17 jae  lbl ;EAX >= EBX (.. or Equal)
18 jbe  lbl ;EAX <= EBX (.. or Equal)
19
20 jne  lbl ;EAX != EBX (Not Equal)
21 jna  lbl ;EAX <= EBX (Not Above)
22 jnb  lbl ;EAX >= EBX (Not Below)
23 jnae lbl ;EAX <  EBX (.. or Equal)
24 jnbe lbl ;EAX >  EBX (.. or Equal)
```

Podmínky typu if-else

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

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

Vyzkoušejte si:

- Podmínka porovná jestli EAX a EBX mají stejnou (Equal) hodnotu:

```
1  if:                      ; zacatek podminky if-else
2    cmp  eax, ebx          ; aritmetickym porovnanim nastavime priznaky
3    je   then              ; pokud podminka   byla splnena skocime na THEN
4    jne  else              ; pokud podminka nebyla splnena skocime na ELSE
5  then:
6    ;cisla byla stejna     ; telo THEN
7    jmp  end               ; po provedeni tela skocime na konec
8  else:
9    ;cisla byla rozdilna   ; telo ELSE
10   jmp  end               ; po provedeni tela skocime na konec
11 end:                     ; konec podminky if-else
```

Zkrácený zápis podmínky

Při nesplnění podmínky program vždy automaticky přejde na další řádek:

- Některé skoky a návěští tak z podmínky můžeme vynechat.
- Výsledný zápis bude kratší ale méně přehledný.

Vyzkoušejte si:

- Podmínka porovná jestli EAX a EBX mají stejnou (Equal) hodnotu:

```
1     cmp  eax, ebx          ; aritmetickym porovnanim nastavime priznaky
2     jne  else              ; pokud podminka nebyla splnena skocime na ELSE
3     ;cisla byla stejna     ; jinak program automaticky prejde na telo THEN
4     jmp  end               ; po provedeni tela THEN skocime na konec
5   else:
6     ;cisla byla rozdilna ; po provedeni tela ELSE program automaticky
7   end:                     ; prejde na konec
```

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

- 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
```

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:
```

Vyzkoušejte si:

- Ze vstupu načtěte dvě znaménková 8b čísla (X a Y).
- Spočítejte a vypište výsledek následující rovnice:

$$f = \frac{1000}{|X * Y|}$$

- Program upravte tak aby vypsal "NaN" pokud by při podílu došlo k dělení nulou.

Například:

$$10, 20 => \frac{1000}{200} = 5$$

$$10, -30 => \frac{1000}{300} = 3$$

$$10, 0 => \frac{1000}{0} = \text{NaN}$$

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?

```
1  if:
2    cmp  eax, 10 ;nastavime priznaky
3    jl   then    ;pokud plati prvni
4    cmp  eax, 20 ;nastavime priznaky
5    jg   then    ;pokud plati druha
6    jmp  else    ;pokud neplatily
7  then:
8    ;telo THEN   ;po provedeni tela
9    jmp  end     ;skocime na konec
10 else:
11   ;telo ELSE   ;po provedeni tela
12   jmp  end     ;skocime na konec
13 end:
```

Je EAX větší jak 10 a menší jak 20?

```
14 if:
15   cmp  eax, 10 ;nastavime priznaky
16   jng  else    ;pokud neplati
17   cmp  eax, 20 ;nastavime priznaky
18   jnl  else    ;pokud neplati
19   jmp  then    ;pokud platily obe
20 then:
21   ;telo THEN   ;po provedeni tela
22   jmp  end     ;skocime na konec
23 else:
24   ;telo ELSE   ;po provedeni tela
25   jmp  end     ;skocime na konec
26 end:
```

Vyzkoušejte si:

- Vytvořte si dvě inicializované 16b 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 rozeznáte podle hodnoty jejich nejnižšího bitu.

Například:

- 10, 20 => obe cisla jsou suda
  11, 20 => alespon jedno cislo je liche
  10, 21 => alespon jedno cislo je liche
  11, 21 => alespon jedno cislo je liche