Cykly a řetězové instrukce

ISU-cv07

Ing. Jakub Husa

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

26. března 2024

Cykly

Cykly typu while

Skokem vzad (nahoru) v programu vytvoříme cyklus:

- U cyklu typu while-do se na začátku těla ptáme jestli se cyklus má ukončit.

```
1  while:
2    cmp   eax, 0   ; porovnanim hodnot nastavime priznaky
3    je    end      ; podminkou rozhodujeme o ukonceni cyklu
4  do:
5    ;telo WHILE
6    jmp   while    ; ne-podminenym skokem se vracime na podminku
7  end:
```

- U cyklu typu do-while se na konci těla ptáme jestli se cyklus má opakovat.

```
8  do:
9    ;telo WHILE
10 while:
11   cmp   eax, 0   ; porovnanim hodnot nastavime priznaky
12   jne   do        ; podminkou rozhodujeme o opakovani cyklu
13 end:
```

Program bude ze vstupu načítat 32b čísla tak dlouho dokud nebyla zadána 0, a pak vypíše jejich sumu:

```
1  %include 'rw32.inc'          ;knihovna pro vstup a vystup
2
3  section .text                ;kodovy segment
4  main:
5      mov  ebx, 0              ; EBX = pocatecni hodnota sumy
6
7    do:
8      call ReadInt32NewLine    ; EAX = vstup
9      add  ebx, eax            ; suma = suma + vstup
10   while:
11     cmp  eax, 0              ; porovnanim nastavime priznaky
12     jne  do                  ; pokud EAX != 0 cyklus se bude opakovat
13   end:
14
15     mov  eax, ebx            ; EAX = suma
16     call WriteInt32NewLine   ; vypis
17     ret
```

Vyzkoušejte si:

- Ze vstupu si načtěte dvě 8b znaménková čísla (X a Y).
- Vypište všechny hodnoty v rozsahu od menšího do většího z nich (včetně).

Například:

- 5, 10 => 5 6 7 8 9 10
  5, 5 => 5
  5, -5 => -5 -4 -3 -2 -1 0 1 2 3 4 5

Cyklus typu for

Cyklus typu for implementujeme instrukcí LOOP (smyčka):

- Instrukce nejprve dekrementuje (DEC) registr ECX, a pokud jeho hodnota není rovna 0, tak provede skok na uvedené návěští (JNZ).

Vyzkoušejte si:

- Program ve smyčce vypíše 32b čísla od 10 do 1 (včetně):

```
1  %include 'rw32.inc'          ;knihovna pro vstup a vystup
2
3  section .text                ;kodovy segment
4  main:
5      mov  ecx, 10             ; ECX = pocet opakovani
6    for:
7      mov  eax, ecx            ; EAX = ECX
8      call WriteInt32NewLine   ; vypis EAX
9      loop for                 ; smycka opakovani
10                              ; DEC ECX; JNZ for
11     ret
```

Adresování paměti

Při adresování paměti můžeme v hranatých závorkách použít až dva 32b registry, a jeden z těchto registrů (vyjma ESP) můžeme násobit číslem 1/2/4/8:

- Pozor – registr od adresy **NEJDE** odečíst, ale může mít zápornou hodnotu!

```
1  [arr + 100]   ;ok
2  [arr + eax]   ;ok
3  [ebx]         ;ok
4  [arr + ecx*2];ok
5  [edx + edx*2];ok
6  [esp + ebp*4];ok
7  [edi*8 - 100];ok
```

```
8  [arr - 100]     ;ok
9  [arr - eax]     ;CHYBA - registr nelze odecitat
10 [bx]            ;CHYBA - registr musi byt 32b
11 [arr + ecx*3]   ;CHYBA - lze nasobit jen cisly 1/2/4/8
12 [edx*2 + edx*4];CHYBA - nelze nasobit oba registry
13 [ebp + esp*4]   ;CHYBA - nelze nasobit registr ESP
14 [arr + esi + edi*8 - 100] ;ok
```

Viz. rovnice pro výpočet efektivní adresy v 32b chráněném režimu (p3s24):

- Displacement – adresa proměnné nebo pole, posunutá o konstantu.

EA = base + index * 2^{scale} + displacement

base = {EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI}

index = {EAX, EBX, ECX, EDX, EBP, ESI, EDI} **(chybí zde ESP!!!)**

displacement = {d8, d32}

scale = {0, 1, 2, 3}

Program v paměti rezervuje pole deseti 32b čísel, a jeho položky inicializuje hodnotami od 100 do 1000 (obsah pole si zkontrolujte v debuggeru):

```
1  %include 'rw32.inc'          ;knihovna pro vstup a vystup
2
3  section .bss                 ;ne-inicializovany datovy segment
4      arr resd 10              ; pole deseti 32b cisel
5
6  section .text                ;kodovy segment
7  main:
8      mov  eax, 100            ; EAX = hodnota prvku pole
9      mov  ebx, 0              ; EBX = index prvku pole
10     mov  ecx, 10             ; ECX = pocet opakovani
11
12   for:
13     mov  [arr+ebx*4], eax    ; arr[EBX] = EAX
14     add  eax, 100            ; hodnotu  prvku zvys o 10
15     inc  ebx                 ; index    prvku zvys o  1
16     loop for                 ; pocet opakovani sniz o 1
17                              ; a pokud neni 0 tak opakuj
18     ret
```

Vyzkoušejte si:

- Vytvořte si inicializované pole pěti 16b bez-znaménkových čísel.
- Projděte přes všechny prvky pole a vypište jejich maximum.

Například:

- 1, 4, 2, 5, 3 => 5
  10000, 40000, 20000, 50000, 30000 => 50000

Řetězové instrukce

Řetězové instrukce - MOVS

Řetězové instrukce existují pro usnadnění práce s poli (vektory, řetězová data):

- Používají nepřímé adresování pomocí registrů ESI a EDI.
- Velikost položek pole (a posuvu) neurčujeme pseudo-instrukcemi (byte/word/dword), ale je přímo součástí názvu instrukce (B/W/D).

Z pole do pole kopírujeme instrukcemi MOVSB, MOVSW, MOVSD (Move String):

- Instrukce zkopíruje jednu položku z adresy v ESI na adresu v EDI a adresy v ESI a EDI posune na další položku.

```
1 movsb   ;zkopiruj  8b z adresy ESI do adresy EDI
2 movsw   ;zkopiruj 16b z adresy ESI do adresy EDI
3 movsd   ;zkopiruj 32b z adresy ESI do adresy EDI
```

Směr posuvu řídíme příznakem DF (Direction Flag):

- Jeho hodnotu nastavíme instrukcemi CLD a STD.
- Pozor – před voláním funkce ReadSring vždy musíme nastavit DF na 0!

```
4 cld     ;Clear Direction Flag  (DF = 0)  posun doprava
5 std     ; Set  Direction Flag  (DF = 1)  posun doleva
```

Program zkopíruje pět 8b položek z pole src do pole dst:

```
1  %include 'rw32.inc'          ;knihovna pro vstup a vystup
2
3  section .bss                 ;ne-inicializovany datovy segment
4      dst resb 5               ;  pole peti 8b hodnot
5
6  section .data                ;inicializovany datovy segment
7      src db 10,20,30,40,50 ; definujeme pole peti 8b hodnot
8
9  section .text                ;kodovy segment
10 main:
11     mov  esi, src            ; ESI = zdrojove pole
12     mov  edi, dst            ; EDI =  cilove  pole
13     mov  ecx, 5              ; ECX = pocet opakovani (polozek pole)
14     cld                      ;  DF = 0 (posun doprava)
15   for:
16     movsb                    ; zkopiruj 8b z adresy ESI na adresu EDI
17                              ; a oba je zvys (DF==0) o jednicku (8b)
18     loop for                 ; dekrementuj ECX a pokud neni
19                              ; nula tak skoc na navesti for
20     ret
```

Z pole do registru kopírujeme instrukcemi LODSB, LODSW, LODSD (Load String):

- Instrukce zkopíruje data z adresy v ESI do registru AL/AX/EAX a adresu v ESI posune na další položku.

```
1 lodsb   ;zkopiruj  8b z adresy ESI do  AL
2 lodsw   ;zkopiruj 16b z adresy ESI do  AX
3 lodsd   ;zkopiruj 32b z adresy ESI do EAX
```

Z registru do pole kopírujeme instrukcemi STOSB, STOSW, STOSD (Store String):

- Instrukce zkopíruje data z registru AL/AX/EAX na adresu v EDI a adresu v EDI posune na další položku.

```
4 stosb   ;zkopiruj  8b z  AL do adresy EDI
5 stosw   ;zkopiruj 16b z  AX do adresy EDI
6 stosd   ;zkopiruj 32b z EAX do adresy EDI
```

Program z prava do leva vypíše pět 32b položek z pole arr:

```
1  %include 'rw32.inc'          ;knihovna pro vstup a vystup
2
3  section .data                ;inicializovany datovy segment
4      arr dd 10,20,30,40,50    ; pole peti 32b hodnot
5
6  section .text                ;kodovy segment
7  main:
8      mov  esi, arr + 16       ; ESI = cilove pole (adresa posledniho prvku)
9      mov  ecx, 5              ; ECX = pocet opakovani
10     std                      ;  DF = 1 (posun doleva)
11   for:
12     lodsd                    ; zkopiruj 32b z adresy v ESI do EAX
13                              ; a ESI sniz (DF==1) o ctyrku (32b)
14     call WriteInt32NewLine   ; vypis obsah registru EAX
15     loop for                 ; dekrementuj ECX a pokud neni
16                              ; nula tak skoc na navesti for
17     ret
```

Vyzkoušejte si:

- Vytvořte si inicializované pole pěti 16b bez-znaménkových čísel.
- Všechny položky pole vydělte číslem deset, a výsledný obsah pole si zobrazte v debuggeru.

Například:

- 11, 22, 33, 44, 55 => 1, 2, 3, 4, 5
  10000, 20000, 30000, 40000, 50000 => 1000, 2000, 3000, 4000, 5000

Pole s polem porovnáme instrukcemi CMPSB, CMPSW, CMPSD (Compare String):

- Instrukce porovná data z adresy v ESI s daty na adrese v EDI, a adresy v ESI a EDI posune na další položku.

```
1 cmpsb   ;porovnej  8b z adresy ESI a adresy EDI
2 cmpsw   ;porovnej 16b z adresy ESI a adresy EDI
3 cmpsd   ;porovnej 32b z adresy ESI a adresy EDI
```

Registr s polem porovnáme instrukcemi SCASB, SCASW, SCASD (Scan String):

- Instrukce porovná data z registru AL/AX/EAX s daty na adrese v EDI, a adresu v EDI posune na další položku.

```
4 scasb   ;porovnej  8b z  AL a adresy EDI
5 scasw   ;porovnej 16b z  AX a adresy EDI
6 scasd   ;porovnej 32b z EAX a adresy EDI
```

Řetězové porovnání nastavuje stejné příznaky jako instrukce CMP (OF, CF, SF, ZF):

- Umožňuje nám použít aritmetické skoky (JG, JL, JA, JB, JE, JNE, ...).

Program spolu porovná dva řetězce (pole 8b hodnot) a vypíše na kolikátém znaku od konce se od sebe odlišují – pokud jsou oba řetězce stejné tak vypíše 0:

```
1  %include 'rw32.inc'          ;knihovna pro vstup a vystup
2  section .data                ;inicializovany datovy segment
3      src db 'Hello, World!', 10, 0 ; zdrojove pole
4      dst db 'Hello, world!', 10, 0 ;  cilove  pole
5  section .text                ;kodovy segment
6  main:
7      mov  esi, src            ; ESI = zdrojove pole
8      mov  edi, dst            ; EDI = cilovehe pole
9      mov  ecx, 15             ; ECX = pocet opakovani (delka pole)
10     cld                      ;  DF = 0 (posun doprava)
11   for:
12     cmpsb                    ; porovnej 8b znaky na adresach ESI a EDI
13                              ;    (a posun se na dalsi polozku)
14     jne skip                 ; pokud nejsou stejne tak ukonci cyklus
15     loop for                 ; dekrementuj ECX a pokud neni nula tak opakuj
16   skip:
17     mov  eax, ecx            ; EAX = zbyvajici pocet opakovani
18     call WriteInt32NewLine ; vypis EAX
19     ret
```

Prefix opakování - REP

Pro opakování řetězové instrukce můžeme místo smyčky použít prefix REP (Repeat):

- REP dekrementuje ECX, a pokud není 0 tak instrukci bude opakovat.
- Pozor – prefix opakování můžeme umístit pouze před řetězovou instrukci!

Vyzkoušejte si:

- Program zkopíruje pět 8b položek z pole src do pole dst.
- Pomocí instrukce LOOP.

```
1  main:
2      mov  esi, src;ESI = zdroj
3      mov  edi, dst;EDI = cil
4      mov  ecx, 5  ;ECX = pocet
5      cld         ;DF = 0 (doprava)
6    for:
7      movsb       ;zkopiruj 8b
8      ; z adresy ESI na adresu EDI
9      ; posun je na dalsi prvek
10     loop for    ;dekrementuj ECX
11     ; a pokud ECX != 0 tak skoc
12     ret
```

- Pomocí prefixu REP.

```
13 main:
14     mov  esi, src;ESI = zdroj
15     mov  edi, dst;EDI = cil
16     mov  ecx, 5  ;ECX = pocet
17     cld         ;DF = 0 (doprava)
18
19     rep movsb    ;zkopiruj 8b
20     ; z adresy ESI na adresu EDI
21     ; posun je na dalsi prvek
22     ; dekrementuj ECX
23     ; a pokud ECX != 0 tak opakuj
24     ret
```

Podmíněné prefixy opakování - REPE a REPNE

Při porovnávání můžeme použít podmíněné prefixy opakování REPE a REPNE:

- Dekrementují ECX, a instrukci budou opakovat pouze pokud ECX není 0, a pokud při porovnávání byla (REPE) nebo nebyla (REPNE) nalezena shoda.

```
1 repe   ;Repeat while   Equal   (ZF == 1)  opakuj pokud shoda  JE
2 repne  ;Repeat while Not Equal (ZF == 0)  opakuj pokud shoda NENI
```

Program spolu porovná dva řetězce (pole 8b hodnot) a vypíše na kolikátém znaku od konce se od sebe odlišují – pokud jsou oba řetězce stejné tak vypíše 0:

```
3  main:
4      mov  esi, src          ; ESI = zdrojove pole
5      mov  edi, dst          ; EDI = cilovehe pole
6      mov  ecx, 16 ;<-POZOR!; ECX = pocet opakovani (delka pole +1) <-POZOR!
7      cld                    ;  DF = 0 (posun doprava)
8      repe cmpsb             ; porovnej 8b znaky na adresach ESI a EDI, posun
9                             ; se na dalsi polozku, dekrementuj ECX, a pokud
10                            ; ECX != 0 a polozky se shoduji, tak opakuj
11     mov  eax, ecx          ; EAX = zbyvajici pocet opakovani
12     call WriteInt32NewLine; vypis EAX
13     ret
```

Vyzkoušejte si:

- Vytvořte si inicializované pole obsahující řetězec 'Hello, World!'.
- Funkcí ReadChar ze vstupu načtěte jeden znak, jeho poslední výskyt v poli nahraďte pomlčkou '–' (45), a vypište upravený řetězec.

Například:

- `H => -ello, World!`
  `e => H-llo, World!`
  `l => Hello, Wor-d!`
  `o => Hello, W-rld!`
  `x => Hello, World!`