Funkce, zásobník, předávání parametrů a konvence volání

ISU-cv08

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

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

4. dubna 2025

Funkce

Funkce

Funkce označujeme návěštím, voláme instrukcí CALL a ukončujeme instrukcí RET:

- CALL – na zásobník uloží návratovou adresu instrukce z následujícího řádku a do programového čítače (EIP) nahraje adresu první instrukce z volané funkce.
- RET – návratovou adresu odstraní ze zásobníku, a nahraje ji do EIP.
- Na návěští funkce **NESMÍME** nikdy skákat – návrat z funkce by způsobil havárii!

```
call WriteInt32NewLine  ; ok    - volani funkce
jmp  WriteInt32NewLine  ; CHYBA - skok na funkci
```

Parametry funkce můžeme předávat třemi způsoby:

- V registrech – uložíme je do EAX, EBX, ECX, EDX (hodnoty) nebo ESI, EDI (adresy).
- V paměti – uložíme je do globálních proměnných (.data, .bss).
- V zásobníku – uložíme je na zásobník podle nějaké konvence volání.

Pokud funkce má nějakou návratovou hodnotu, tak ji vracíme v registru EAX:

- Každá funkce by měla končit právě jednou instrukcí RET.
- Každá instrukce by měla být součástí nějaké funkce.

Na vyzkoušení

Funkce main zavolá funkci Zdvojnasob která zdvojnásobí hodnotu v registru EAX:

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

section .text                ;kodovy segment
main:                        ;zacatek funkce main (zacatek programu)

    mov  eax, 10             ; EAX = 10
    call Zdvojnasob          ; zavolej funkci Zdvojnasob
    call WriteInt32NewLine   ; vypis (EAX = 20)

    mov  eax, 100            ; EAX = 100
    call Zdvojnasob          ; zavolej funkci Zdvojnasob
    call WriteInt32NewLine   ; vypis (EAX = 200)

    ret                      ; konec funkce main (konec programu)

Zdvojnasob:                  ;zacatek funkce Zdvojnasob
    add eax, eax             ; EAX = EAX + EAX
    ret                      ; konec funkce Zdvojnasob (navrat do main)
```

Na procvičení

Vyzkoušejte si:

- Zkopírujte si šablonu a implementujte funkci SumaPole která jako vstup dostane adresu pole (EAX) a počet jeho prvků (EBX), a jako výstup vrátí jejich sumu (EAX).
- Funkci ošetřete tak, aby ověřila že adresa pole je platná (není NULL, 0), a že počet jeho prvků je kladné číslo – pokud ne, tak funkce vrátí nulu.

Například:

- arr1, 5 => 150
- arr2, 3 => 600
- arr2, -1 => 0

Nápověda:

- Než začneme cyklus, parametry funkce musíme přesunout do správných registrů.
- Při průchodu polem se nám budou hodit instrukce LOOP a LODS (viz. cv07).

Šablona

```
%include 'rw32.inc'          ;knihovna pro vstup a vystup
section .data                ;inicializovany segment
    arr1 dd 10,20,30,40,50 ; pole peti 32b cisel
    arr2 dd 100, 200, 300  ; pole tri  32b cisel
section .text                ;kodovy segment
main:
    mov  eax, arr1           ; prvni vstup funkce (EAX = adresa pole)
    mov  ebx, 5              ; druhy vstup funkce (EBX = pocet prvku)
    call SumaPole            ; zavolej funkci SumaPole
    call WriteInt32NewLine ; vypis navratovou hodnotu (150)
    mov  eax, arr2           ; jine pole
    mov  ebx, 3              ; jine velikost
    call SumaPole            ; volani pro jine parametry
    call WriteInt32NewLine ; vypis navratovou hodnotu (600)
    mov  eax, arr2           ; platne pole
    mov  ebx, -1             ; neplatna velikost
    call SumaPole            ; volani pro neplatne parametry
    call WriteInt32NewLine ; vypis navratovou hodnotu (0)
    ret
  SumaPole:                  ; zacatek funkce SumaPole
    ;???
    ret                      ; konec   funkce SumaPole
```

Zásobník

Zásobník

Zásobník je paměťový segment ukládající dočasná data:

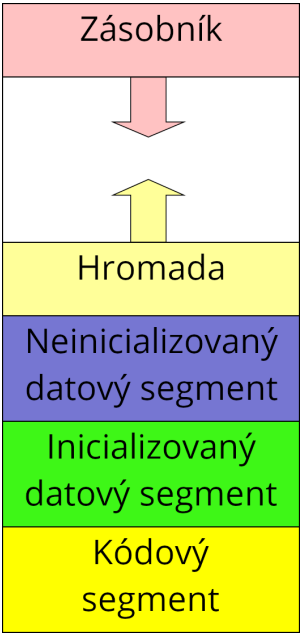
- Jeho velikost je, na rozdíl od datových segmentů (.data, .bss), proměnlivá a závisí na množství uložených položek.
- Maximální velikost zásobníku je omezena množstvím dostupné operační paměti (v 32b režimu až 4 GiB).
- Pokud nám na zásobníku dojde místo, nastává přetečení zásobníku a program havaruje.

Zásobník vždy roste **SMĚREM DOLŮ**:

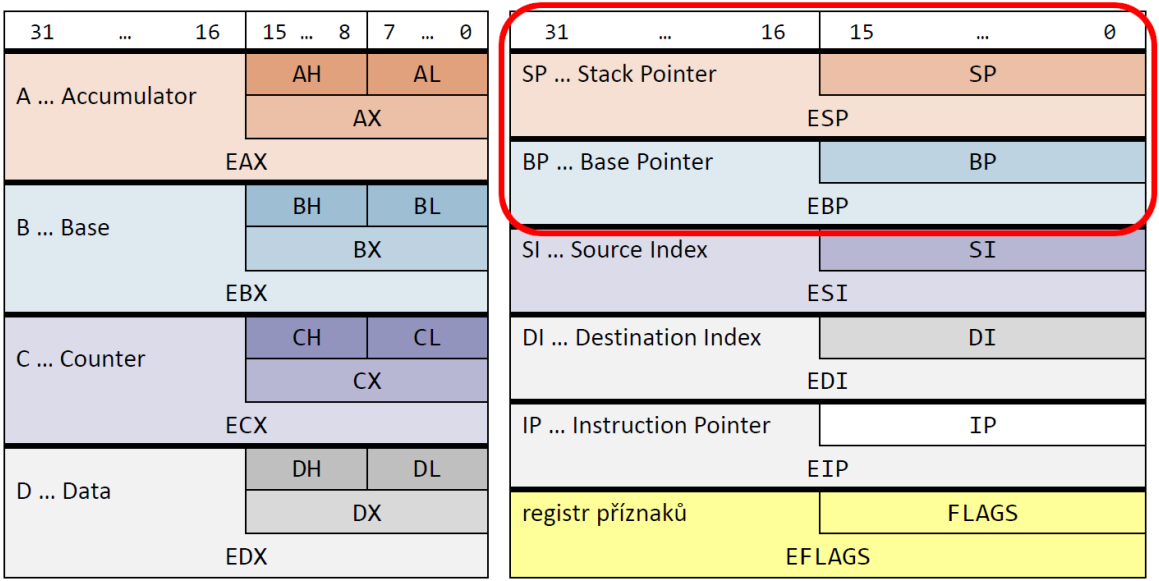
- Adresu jeho dna uchovává registr EBP (Base Pointer).
- Adresu jeho vrcholu uchovává registr ESP (Stack Pointer).
- Vrchol vždy bude mít stejnou nebo nižší adresu než dno!

Vrchol vždy ukazuje na poslední uloženou položku:

```
⊖ Stack content
[ESP+12] =        [0x000413AC] =        13, 0x0000000D   > rw32_exit
[ESP+08] =        [0x000413A8] =      3271, 0x00000CC7   > line 6
[ESP+04] =        [0x000413A4] =        10, 0x0000000A
[ESP] =           [0x000413A0] =        20, 0x00000014
```



Registry procesoru - vrchol a dno zásobníku



Ukládání na zásobník

Novou položku na zásobník uložíme instrukcí PUSH:

- Nejprve posune vrchol zásobníku (ESP), a pak na něj uloží novou položku.
- Na zásobník vždy ukládáme položky o velikosti 32b.

```
push eax        ; hodnotu EAX uloz na vrchol zasobniku
push dword 10   ; na vrchol zasobniku uloz hodnotu "10"
push ax         ; CHYBA - registr AX nema 32b
```

Položku uloženou na vrcholu zásobníku z něj odebereme instrukcí POP:

- Nejprve odebere poslední uloženou položku, a pak posune vrchol zásobníku.
- Položku můžeme smazat i posunutím vrcholu zásobníku.

```
pop  eax        ; hodnotu z vrcholu zasobniku odeber do EAX
add  esp, 4     ; smaz posledni ulozenou polozku
```

Vše co na zásobník uložíme z něj před ukončením funkce **MUSÍME** také odebrat:

- Jinak instrukce RET do EIP nahraje neplatnou adresu a program havaruje.

```
push dword 10   ; na vrchol zasobniku uloz hodnotu "10"
ret             ; CHYBA - do EIP nahraje adresu "10"
```

Zálohování registrů

Funkce by měla zálohovat a obnovovat hodnotu všech používaných registrů, vyjma výstupního registru EAX, a příznakového registru EFLAGS:

- Registry vždy obnovujeme v opačném pořadí něž ve kterém jsme je zálohovali.
- Registr EAX zálohujeme pouze pokud funkce nemá návratovou hodnotu.
- Registr EFLAGS můžeme zálohovat a obnovit instrukcemi PUSHFD a POPFD.

```
foo:
    push eax        ; zalohujeme registr EAX (pokud neni navratova hodnota)
    push ebx        ; zalohujeme registr EBX
    push ecx        ; zalohujeme registr ECX
    push edx        ; zalohujeme registr EDX
    pushfd          ; zalohujeme registr EFLAGS (obvykle neni potreba)

    ;telo funkce

    popfd           ; obnovujeme registr EFLAGS (obvykle neni potreba)
    pop  edx        ; obnovujeme registr EDX
    pop  ecx        ; obnovujeme registr ECX
    pop  ebx        ; obnovujeme registr EBX
    pop  eax        ; obnovujeme registr EAX (pokud neni navratova hodnota)
    ret
```

Na vyzkoušení

Sledujte obsah zásobníku při provádění následujících instrukcí:

```
section .text                ;  EAX  EBX  ; [ESP+0] [ESP+4] [ESP+8] [ESP+12]
main:                        ;    ?    ?  ; rw_exit       ?       ?        ?
    mov  eax, 10             ;   10    ?  ; rw_exit       ?       ?        ?
    mov  ebx, 20             ;   10   20  ; rw_exit       ?       ?        ?
    call foo                 ; volani funkce ulozi navratovu adresu
    add  eax, ebx            ;   30   20  ; rw_exit       ?       ?        ?
    call WriteInt32NewLine   ;   30   20  ; rw_exit       ?       ?        ?
    ret                      ; main se vraci do operacniho systemu

                             ;  EAX  EBX  ; [ESP+0] [ESP+4] [ESP+8] [ESP+12]
foo:                         ;   10   20  ; NAV_ADR rw_exit       ?        ?
    push eax                 ;   10   20  ;      10 NAV_ADR rw_exit        ?
    push ebx                 ;   10   20  ;      20      10 NAV_ADR rw_exit
    mov  eax, 100            ;  100   20  ;      20      10 NAV_ADR rw_exit
    mov  ebx, 200            ;  100  200  ;      20      10 NAV_ADR rw_exit
    add  eax, ebx            ;  300  200  ;      20      10 NAV_ADR rw_exit
    call WriteInt32NewLine   ;  300  200  ;      20      10 NAV_ADR rw_exit
    pop  ebx                 ;  300   20  ;      10 NAV_ADR rw_exit        ?
    pop  eax                 ;   10   20  ; NAV_ADR rw_exit       ?        ?
    ret                      ; navrat odebere a pouzije navratovou adresu
```

Na procvičení

Vyzkoušejte si:

- Zkopírujte si šablonu a implementujte funkci VypisRadek, která vypíše několik hvězdiček '*' a jeden konec řádku.
- Vstupem funkce je počet vypisovaných hvězdiček, předaný v registru EAX.

Nápověda:

- Znak (hodnotu AL dle ASCII) vypíšeme funkcí WriteChar, prázdný řádek vypíšeme funkcí WriteNewLine.
- Funkce VypisRadek musí zálohovat a obnovovat hodnoty všech používaných registrů, jinak rozbije cyklus funkce main.
- Pokud na výstupu místo čtverce máte trojúhelník, tak jste zapomněli že vstup funkce je v registru EAX.

Například:

- 0 =>
- 1 => *
- 2 => **
**
- 3 => ***
***
***
- 4 => ****
****
****
****
- 5 => *****
*****
*****
*****
*****

Šablona

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

section .text                ;kodovy segment
main:
    call ReadUInt8NewLine ; AL = vstup
    cmp  al, 0               ; AL porovnavame s nulou
    je   .skip               ; pokud AL==0 tak preskocime cyklus

    movzx eax, al            ; EAX=AL (bez-znamenkove rozsireni)
    mov  ecx, eax            ; ECX = pocet opakovani
  .for:
    call VypisRadek          ; zavolej funkci VypisRadek
    loop .for                ; opakuj volani funkce

  .skip:
    ret

VypisRadek:                  ; zacatek funkce VypisRadek
    ;???
    ret                      ; konec   funkce VypisRadek
```

Předávání parametrů

Zásobníkový rámec

Zásobníkový rámec (stack frame) je oblast mezi EBP a ESP, která na zásobníku ohraničuje místo které patří aktuální funkci:

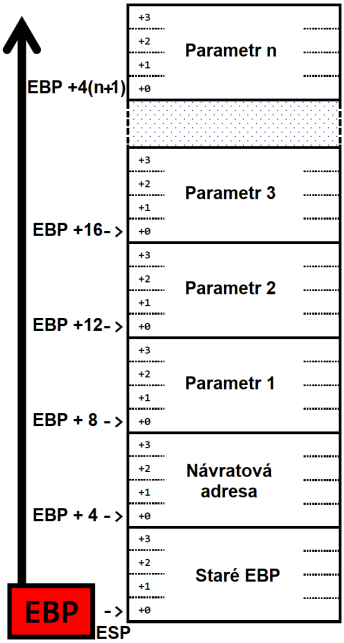
- Rámec nám umožňuje funkcím přes zásobník předávat parametry a alokovat pro ně lokální proměnné (viz. cv09).

Aby si funkce vzájemně nepřepisovaly data, každá funkce která zásobník používá si na něm musí označit svůj vlastní rámec:

- Na začátku funkce zálohujeme staré dno zásobníku (EBP), a potom adresu dna přepíšeme adresou vrcholu (ESP).
- Po vytvoření nového dna zálohujeme ostatní registry.
- Na konci funkce původního adresu dna zase obnovíme.

S výjimkou vytváření a rušení zásobníkového rámce, adresu dna zásobníku (EBP) **NIKDY** neupravujeme:

- Dno nám díky tomu na zásobníku vytvoří "pevný bod" který používáme pro adresování uložených hodnot.
- Funkce nesmí přepisovat hodnoty které nejsou součástí jejího zásobníkového rámce, ale může je číst.



Předávání parametrů přes zásobník

Volající funkce předávané parametry před voláním (CALL) uloží na zásobník (PUSH), a po volání je ze zásobníku obvykle také odstraní, posunutím vrcholu zásobníku (ESP):

```
main:
    push arr        ; predevame parametr (napriklad adresa pole)
    push dword 5    ; predavame parametr (napriklad delka  pole)
    call foo        ; volame funkci foo
    add  esp, 8     ; mazeme predane parametry (2*4B)
    ret
```

Volaná funkce (po vytvoření nového rámce) pak předané parametry může adresovat pomocí dna zásobníku, podobně jako položky pole: [EBP+8], [EBP+12], [EBP+16], atd.

```
foo:
    push ebp         ; zalohuj stare dno
    mov  ebp, esp    ; vytvor nove dno
    ;[ebp + 12]      ; predany parametr (napriklad adresa pole)
    ;[ebp +  8]      ; predany parametr (napriklad delka  pole)
    ;[ebp +  4]      ; navratova adresa do volajici funkci
    ;[ebp +  0]      ; zalohovana hodnota stareho dna
    pop  ebp         ; obnov stare dno
    ret              ; konec funkce
```

Na vyzkoušení

Funkce Secti spočítá součet dvou 32b hodnot předaných přes zásobník:

```
%include 'rw32.inc'          ;knihovna pro vstup a vystup
section .text                ;kodovy segment
main:
    push dword 20            ; predavame druhy parametr
    push dword 10            ; predavame prvni parametr
    call Secti               ; volame funkci soucet
    add  esp, 8              ; predane paremetry mazeme ze zasobniku
    call WriteInt32NewLine   ; vypis EAX (30)
    ret

Secti:
    push ebp                 ; zalohujeme stare dno
    mov  ebp, esp            ; vytvarime nove dno

    mov  eax, [ebp + 8]      ; EAX = prvni parametr
    add  eax, [ebp +12]      ; EAX = EAX + druhy parametr
                             ; (EAX je navaratova hednota funkce)
    pop  ebp                 ; obnovujeme stare dno
    ret
```

Na procvičení

Vyzkoušejte si:

- Zkopírujte si šablonu a implementujte funkci PocetZapornych která spočítá kolik pole, předané přes zásobník, obsahuje záporných čísel.
- Zálohování hodnoty používaných registrů (EAX, EBX, ECX, ...) a ověřování platnosti parametrů funkce (arr != 0, ECX > 0) můžete ignorovat.

Například:

- arr1, 4 => 2
- arr2, 5 => 3

```
Stack content
[ESP+16] = [EBP+16] = [0x000413CC] =          13, 0x0000000D  > rw32_exit
[ESP+12] = [EBP+12] = [0x000413C8] =        3763, 0x00000EB3  arr1
[ESP+08] = [EBP+08] = [0x000413C4] =           4, 0x00000004
[ESP+04] = [EBP+04] = [0x000413C0] =        3268, 0x00000CC4  > line 10
[ESP] =    [EBP] =    [0x000413BC] =  -84215046, 0xFAFAFAFA
```

Nápověda:

- Parametry předané přes zásobník jsou na adresách [EBP+8] a [EBP+12].
- Pro testování hodnoty položek pole se nám bude hodit instrukce SCAS (viz. cv07).

Šablona

```
%include 'rw32.inc'          ;knihovna pro vstup a vystup
section .data                ;inicializovany segment
    arr1 dw -1,-2, 3, 4      ; pole ctyr 16b cisel
    arr2 dw -1,-2,-3, 0, 0   ; pole peti 16b cisel

section .text                ;kodovy segment
main:
    push arr1                ; parametr funkce (adresa pole)
    push dword 4             ; parametr funkce (pocet prvku)
    call PocetZapornych      ; volame funkci
    add  esp, 8              ; mazeme predane parametry (2*4B)
    call WriteInt32NewLine   ; vypis navratovou hodnotu (2)

    push arr2                ; jine pole
    push dword 5             ; jina delka
    call PocetZapornych      ; volame funkci
    add  esp, 8              ; mazeme predane parametry (2*4B)
    call WriteInt32NewLine   ; vypis navratovou hodnotu (3)
    ret

PocetZapornych:              ; zacatek funkce PocetZapornych
    push ebp                 ; zalohujeme stare dno
    mov  ebp, esp            ; vytvarime nove dno
    ;???
    pop  ebp                 ; obnovujeme stare dno
    ret                      ; konec funkce PocetZapornych
```

Konvence volání

Konvence volání

Způsob předávání parametrů přes zásobník se řídí konvencemi volání:

- Konvence jsou odvozeny od implementace vyšších programovacích jazyků.
- Konvence určují také jestli parametry odstraňuje volající nebo volaná funkce.
- Volaná funkce parametry může odstranit použitím nepovinného operandu instrukce RET který se po návratu z funkce přičte k vrcholu zásobníku (ESP).

```
ret      ;konec funkce
ret  4   ;konec funkce a "add esp, 4" (odstran jeden 32b parametr)
```

Konvence volání	Parametry funkce	Zásobník uklízí	Dekorace jmen (pro jazyk C)	Použito v
pascal	zleva doprava	volaný	symbol	Pascal
cdecl	zprava doleva	volající	_symbol	Jazyk C
stdcall	zprava doleva	volaný	_symbol@4	Win32 API
fastcall	první dva parametry v ECX a EDX, zbytek zprava doleva	volaný	@symbol@8	různé

Na vyzkoušení

Volání funkce dle CDECL:

```
%include 'rw32.inc' ;knihovna
section .text       ;kodovy segment
main:
    push dword 20   ;druhy parametr
    push dword 10   ;prvni parametr
    call Soucet     ;zavolej soucet
    add  esp, 8     ;odstran param
    call WriteInt32 ;vypis vysledek
    ret

Soucet:             ;CDECL
    push ebp        ;zalohuj dno
    mov  ebp, esp   ;vytvor  dno

    mov  eax, [ebp + 8] ;prvni
    add  eax, [ebp +12] ;druhy

    pop  ebp        ;obnov dno
    ret             ;konec funkce
                    ;
```

Volání funkce dle STDCALL:

```
%include 'rw32.inc' ;knihovna
section .text       ;kodovy segment
main:
    push dword 20   ;druhy parametr
    push dword 10   ;prvni parametr
    call Soucet     ;zavolej soucet

    call WriteInt32 ;vypis vysledek
    ret

Soucet:             ;STDCALL
    push ebp        ;zalohuj dno
    mov  ebp, esp   ;vytvor  dno

    mov  eax, [ebp + 8] ;prvni
    add  eax, [ebp +12] ;druhy

    pop  ebp        ;obnov dno
    ret  8          ;konec funkce a
                    ;odstran param
```

Na vyzkoušení

Volání funkce dle PASCAL:

```
%include 'rw32.inc' ;knihovna
section .text       ;kodovy segment
main:
    push dword 10   ;prvni parametr
    push dword 20   ;druhy parametr
    call Soucet     ;zavolej soucet

    call WriteInt32 ;vypis vysledek
    ret

Soucet:             ;PASCAL
    push ebp        ;zalohuj dno
    mov  ebp, esp   ;vytvor  dno

    mov  eax, [ebp +12] ;prvni
    add  eax, [ebp + 8] ;druhy

    pop  ebp        ;obnov dno
    ret  8          ;konec funkce a
                    ;odstran param
```

Volání funkce dle FASTCALL:

```
%include 'rw32.inc' ;knihovna
section .text       ;kodovy segment
main:
    mov  ecx, 10    ;prvni parametr
    mov  edx, 20    ;druhy parametr
    call Soucet     ;zavolej soucet

    call WriteInt32 ;vypis vysledek
    ret

Soucet:             ;FASTCALL
    push ebp        ;zalohuj dno
    mov  ebp, esp   ;vytvor  dno

    mov  eax, ecx       ;prvni
    add  eax, edx       ;druhy

    pop  ebp        ;obnov dno
    ret  0          ;konec funkce a
                    ;odstran param
```

Na procvičení

Vyzkoušejte si:

- Zkopírujte si šablonu a implementujte funkci TrojuhelnikovaCisla která spočítá hodnotu několika prvních trojúhelníkových čísel definovaných rekurzivní rovnicí:

$$f(0) = 0 \qquad f(n) = n + f(n-1)$$

- Prvním parametrem funkce je pole arr,do kterého má funkce zapsat výsledky, druhým parametrem funkce je adresa proměnné len, která určuje jeho délku.
- Parametry funkce jsou předané přes zásobník, dle konvence PASCAL.
- Zálohování hodnoty používaných registrů (EAX, EBX, ECX, ...) a ověřování platnosti parametrů funkce (arr != 0, len > 0) můžete ignorovat.

Například:

- 0 1 3 6 10 15 21 28 36 45 55 66 78 91 105 120 136 ... 4950

Nápověda:

- Pozor, funkce dostává adresu proměnné len, ne její hodnotu.
- Nepokoušejte se o implementaci rekurze – novou hodnotu je jednodušší spočítat součtem aktuální iterace cyklu a předcházejícího výsledku.
- Pro ukládání hodnot do pole se nám bude hodit instrukce STOS (viz. cv07).

Šablona

```
%include 'rw32.inc'            ;knihovna pro vstup a vystup
section .bss                   ;neinicializovany segment
    arr  resd 100              ; pole sta 32b hodnot
section .data                  ;inicializovany segment
    len  dd   100              ; jedna 32b promenna s hodnotou sto

section .text                  ;kodovy segment
main:
    push arr                   ; predavame prvni parametr (pole arr)
    push len                   ; predavame druhy parametr (promenna len)
    call TrojuhelnikovaCisla   ; volame funkci

    mov  esi, arr              ; ESI = adresa vypisovaneho pole
    mov  ecx, [len]            ; ECX = delka pole (pocet opakovani)
    cld                        ;  DF = 0 (posun doprava)
  .for:
    lodsd                      ; EAX = aktualni prvek pole
    call WriteUInt32NewLine    ; vypis EAX
    loop .for                  ; opakuj cyklus

    ret
```