Funkce, zásobník, předávání parametrů a rekurze

ISU-cv08

Ing. Jakub Husa

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

4. dubna 2024

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!

```
1 call WriteInt32NewLine  ; ok     - volani funkce
2 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 návratovou hodnotu vracíme ji v registru EAX:

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

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

```
1  %include 'rw32.inc'          ;knihovna pro vstup a vystup
2
3  section .text                ;kodovy segment
4  main:                        ;zacatek funkce main (zacatek programu)
5
6      mov  eax, 10             ; EAX = 10
7      call Zdvojnasob          ; zavolej funkci Zdvojnasob
8      call WriteInt32NewLine   ; vypis (EAX = 20)
9
10     mov  eax, 100            ; EAX = 100
11     call Zdvojnasob          ; zavolej funkci Zdvojnasob
12     call WriteInt32NewLine   ; vypis (EAX = 200)
13
14     ret                      ; konec funkce main (konec programu)
15
16 Zdvojnasob:                  ;zacatek funkce Zdvojnasob
17     add eax, eax             ; EAX = EAX + EAX
18     ret                      ; konec funkce Zdvojnasob (navrat do main)
```

Na procvičení

Vyzkoušejte si:

- Vytvořte si inicializované pole 32b znaménkových čísel.
- Napište funkci SumaPole která jako vstup dostane adresu pole (ESI) a počet jeho prvků (EAX), a jako výstup vrátí jejich sumu (EAX).
- Funkci SumaPole zavolejte z funkce main a vypište její výsledek.

Například:

```
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 32b znamenkovych cisel
5
6  section .text                ;kodovy segment
7  main:
8      mov  esi, arr            ; do ESI nastav adresu pole (vstup funkce)
9      mov  eax, 5              ; do EAX nastav pocet prvku (vstup funkce)
10     call SumaPole            ; zavolej funkci SumaPole
11     call WriteInt32NewLine ; vypis EAX (navratova hodnota)
12     ret
```

Zásobník

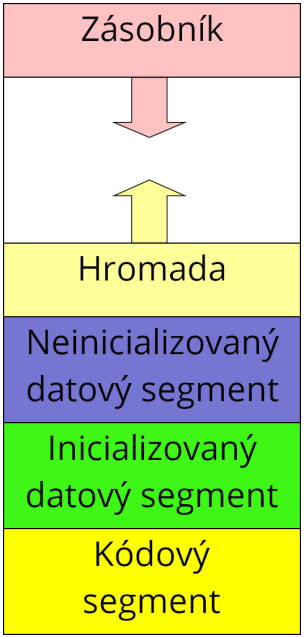
Zásobník

Zásobník je paměťový segment ukládající dočasná data:

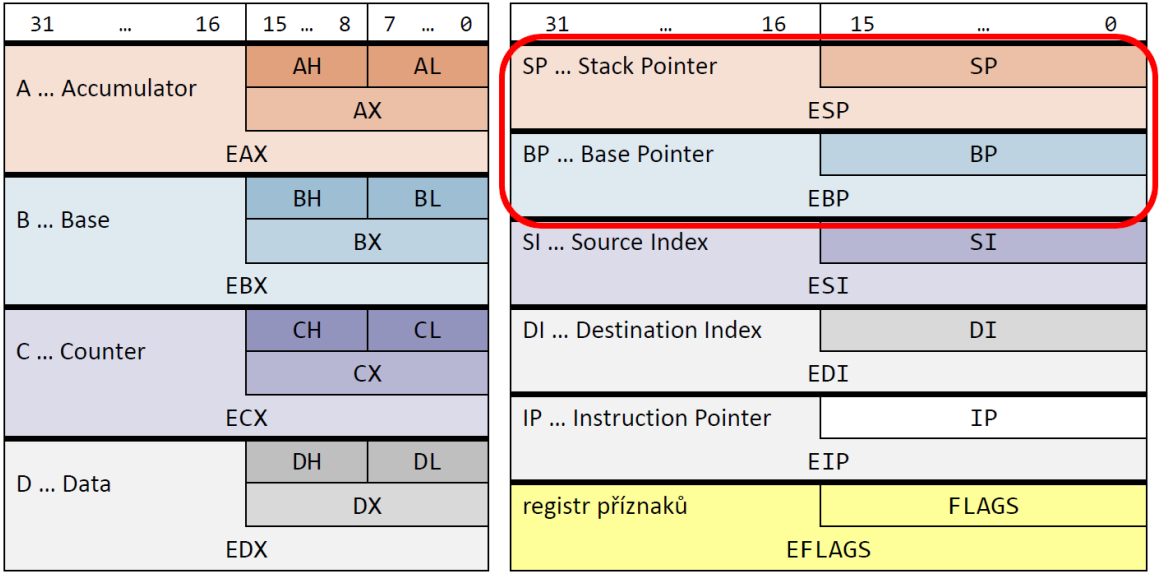
- Jeho velikost je, na rozdíl od datových segmentů (.data, .bss), proměnlivá a závisí na množství uložených položek.
- Maximální velikost zásobníku je omezena množstvím dostupné operační paměti (v 32b režimu 4 GiB).
- Pokud nám na zásobníku dojde místo, nastává přetečení zásobníku a program havaruje.

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!



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 bychom měli ukládat pouze položky o velikosti 32b.

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

Položku uloženou na zásobníku z něj odebereme instrukcí POP:

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

```
4 pop  eax        ; hodnotu z vrcholu zasobniku odeber do EAX
5 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.

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

Sledujte jak se bude měnit obsah zásobníku při provádění následujících instrukcí:

```
1  section .text                  ;kodovy segment
2
3                                 ;  EAX   ;  [ESP+0] [ESP+4] [ESP+8] [ESP+12]
4  main:                          ;   ?    ;     ?       ?       ?       ?
5      mov  eax, 111              ;  111   ;     ?       ?       ?       ?
6      push eax                   ;  111   ;    111      ?       ?       ?
7      push dword 222             ;  111   ;    222     111      ?       ?
8      call foo
9                                 ;  333   ;    222     111      ?       ?
10     mov  dword [esp+4], 444    ;  333   ;    222     444      ?       ?
11     add  esp, 4                ;  333   ;    444      ?       ?       ?
12     pop  eax                   ;  444   ;     ?       ?       ?       ?
13     ret
14                                ;  EAX   ;  [ESP+0] [ESP+4] [ESP+8] [ESP+12]
15 foo:                           ;  111   ;  NAV_ADR   222     111       ?
16     push dword 333             ;  111   ;    333   NAV_ADR   222     111
17     pop  eax                   ;  333   ;  NAV_ADR   222     111       ?
18     ret
```

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.
- Pokud funkce nemá návratovou hodnotu tak registr EAX zálohujeme také.
- Registr EFLAGS zálohujeme a obnovujeme instrukcemi PUSHFD a POPFD.

```
1  foo:
2      push ebx        ; zalohujeme registr EBX
3      push ecx        ; zalohujeme registr ECX
4      push edx        ; zalohujeme registr EDX
5      pushfd          ; zalohujeme registr EFLAGS (obvykle neni potreba)
6
7      ;telo funkce
8
9      popfd           ; obnovujeme registr EFLAGS (obvykle neni potreba)
10     pop  edx        ; obnovujeme registr EDX
11     pop  ecx        ; obnovujeme registr ECX
12     pop  ebx        ; obnovujeme registr EBX
13     ret
```

Vyzkoušejte si:

- Napište program který ze vstupu načte 8b číslo (X) a X-krát zavolá funkci VypisRadek.
- Funkce VypisRadek vypíše X hvězdiček a jeden konec řádku.
- Vstupem funkce bude hodnota X, předávaná v registru EAX.
- Funkce musí zálohovat hodnoty všech používaných registrů.

```
1  %include 'rw32.inc'          ;knihovna pro vstup a vystup
2
3  section .text                ;kodovy segment
4  main:
5      call ReadUInt8NewLine ;  AL = vstup
6      and  eax, 255         ; bez-znam rozsireni AL->EAX
7      mov  ecx, eax         ; ECX = pocet opakovani
8    for:
9      call VypisRadek       ; zavolej funkci VypisRadek
10     loop for              ; volani X-krat opakuj
11     ret
```

Například:

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

Předávání parametrů

Zásobníkový rámec

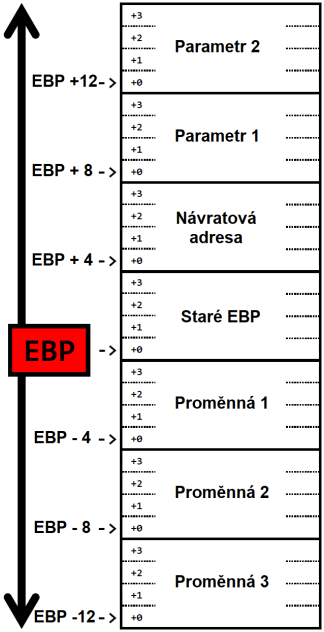
Zásobníkový rámec (stack frame) je oblast mezi EBP a ESP, která na zásobníku ohraničuje místo které patří aktuální funkci:

- Zásobníkový rámec nám umožňuje funkcím přes zásobník předávat parametry a alokovat pro ně lokální proměnné.

Každá funkce si na zásobníku vytváří svůj vlastní rámec:

- Na začátku funkce nejprve zálohujeme staré dno, a pak zkopírováním ESP do EBP vytvoříme nové dno.
- Uvnitř funkce pak hodnotu EBP už nikdy neupravujeme!
- Ostatní registry zálohujeme až po vytvoření nového dna.
- Na konci funkce obnovujeme staré dno volající funkce.

```
1  foo:
2      push ebp          ; zalohuj stare dno
3      mov  ebp, esp     ; vytvor nove dno
4      ;telo funkce
5      pop  ebp          ; obnov stare dno
6      ret               ; konec funkce
```



Předávání parametrů přes zásobník

Způsob předávání parametrů se řídí konvencemi volání:

- Dnes budeme používat konvenci CDECL z jazyka C.

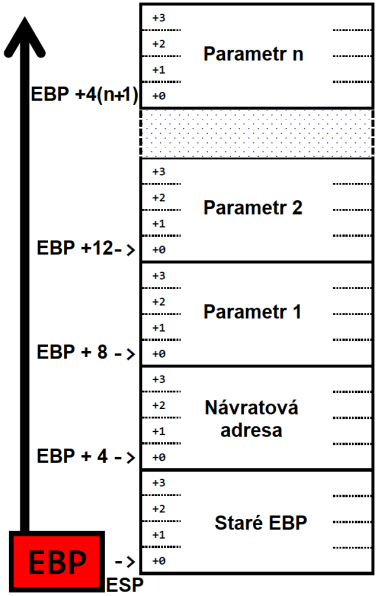
Volající funkce předávané parametry uloží na zásobník v pořadí z prava do leva (PUSH):

- Volání funkce na něj uloží návratovou adresu (CALL).
- Volaná funkce na něj uloží staré dno (PUSH).
- Protože se adresa nového dna (EBP) už nikdy nezmění, můžeme ji používat k adresování předaných parametrů.

```
1  [ebp + 16]     ; treti parametr
2  [ebp + 12]     ; druhy parametr
3  [ebp +  8]     ; prvni parametr
4  [ebp +  4]     ; navratova adresa
5  [ebp +  0]     ; stare dno
```

Parametry které po volání funkce zůstanou na zásobníku z něj podle konvence CDECL odstraňuje volající funkce:

- Volaná funkce svoje parametry nesmí přepisovat protože nejsou uvnitř jejího zásobníkového rámce!



Funkce Vynasob spočítá 32b násobek dvou parametrů předávaných přes zásobník:

```
1  main:
2      push dword 20            ; predevame druhy parametr
3      push dword 10            ; predavame prvni parametr
4      call Vynasob             ; volame funkci soucet
5      add  esp, 8              ; oba paremetry odstranujeme ze zasobniku
6      call WriteInt32NewLine   ; vypis (EAX = 30)
7      ret
8
9  Vynasob:
10     push ebp                 ; zalohujeme stare dno
11     mov  ebp, esp            ; vytvarime nove dno
12
13     push edx                 ; registr EDX zalohujeme az po vytvoreni dna
14     mov  eax, [ebp + 8]      ; EAX = prvni parametr
15     imul dword [ebp +12]     ; EDX:EAX = EAX * druhy parametr
16     pop  edx                 ; registr EDX obnovujeme pred obnovenim dna
17
18     pop  ebp                 ; obnovujeme stare dno
19     ret
```

Vyzkoušejte si:

- Vytvořte si inicializované pole 16b znaménkových čísel.
- Napište funkci PocetZapornych která přes zásobník dostane počet prvků a adresu pole, a která spočítá kolik toto pole obsahuje záporných čísel.
- Funkci PocetZapornych zavolejte z funkce main, a vypište její výsledek.

Například:

```
1  %include 'rw32.inc'          ;knihovna pro vstup a vystup
2  section .data                ;inicializovany datovy segment
3      arr dw 1, -2, 3, -4, 5
4  section .text                ;kodovy segment
5  main:
6      push arr                 ; adresa pole (druhy parametr)
7      push dword 5             ; pocet prvku (prvni parametr)
8      call PocetZapornych      ; volame funkci
9      add  esp, 8              ; predavane parametry maze volajici funkce
10     call WriteInt32NewLine   ; vypis vysledek
11     ret
```

Rekurze

Rekurze nastává ve chvíli kdy některá z funkcí volá sebe sama:

- Při každém volání se na zásobník ukládá nová návratová adresa.
- Pokud funkce dostává parametry předávané přes zásobník, tak se na něm při každém volání vytváří i nový zásobníkový rámec.

Na rozdíl od cyklu, rekurze vždy musí mít nějakou ukončující podmínku:

- Rekurze bez ukončující podmínky vždy způsobí přetečení zásobníku.

```
1  %include 'rw32.inc'          ;knihovna pro vstup a vystup
2  section .text                ;kodovy segment
3  main:
4      mov  eax, 0              ; EAX = pocitadlo
5      call func                ; zavolej funkci func
6      ret
7
8  func:
9      inc  eax                 ; inkrementuj pocitadlo
10     call WriteInt32NewLine   ; vypis EAX
11     call func                ; zavolej sebe sama
12     ret                      ; konec funkce (nikdy se nestane)
```

Funkce main zavolá funkci RekurzivniVypis s jedním parametrem předávaným přes zásobník, která pomocí rekurze vypíše odpovídající počet levých a pravých závorek:

```
1  %include 'rw32.inc'          ;knihovna pro vstup a vystup
2
3  section .text                ;kodovy segment
4  main:
5      call ReadUInt32NewLine; nacitame vstup
6      push eax                 ; predavame paremetr
7      call RekurzivniVypis     ; volame funkci
8      add  esp, 4              ; predany parametr mazeme ze zasobniku
9      ret
```

Na vyzkoušení (2/2)

```
10 RekurzivniVypis:
11     push ebp                ; zalohujeme stare dno
12     mov  ebp, esp           ; vytvarime nove dno
13
14     cmp  [ebp+8], dword 0   ; parametr funkce porovnavame s nulou
15     je   skip               ; pokud je parametrem nula (ukoncujici podminka)
16                             ;   tak skocime na konec, pred obnoveni dna
17
18     mov  al, '('            ; AL = vypisovany znak
19     call WriteChar          ; prvni vypis
20
21     mov  eax, [ebp+8]       ; EAX = parametr funkce
22     dec  eax                ; parametr funkce snizime o jednicku
23     push eax                ; novy parametr ulozime na zasobnik
24     call RekurzivniVypis    ; funkce zavola sebe sama (s novym parametrem)
25     add  esp, 4             ; predany parametr mazeme ze zasobniku
26
27     mov  al, ')'            ; AL = vypisovany znak
28     call WriteChar          ; druhy vypis
29   skip:
30     pop  ebp                ; obnovujeme stare dno
31     ret
```

Na procvičení

Vyzkoušejte si:

- Ze vstupu načtěte jedno 32b číslo, a pomocí cyklu vypočítejte hodnotu N-tého pyramidového čísla:

$$f(0) = 0 \qquad f(n) = n^2 + f(n-1)$$

- Vstupem funkce je parametr N předávaný přes zásobník.
- Horních 32b výsledku násobení můžete ignorovat.

Například:

- 0 => 0
- 1 => 1
- 2 => 5
- 3 => 14
- 4 => 30
- 5 => 55

```
1  %include 'rw32.inc'          ;knihovna pro vstup a vystup
2
3  section .text                ;kodovy segment
4  main:
5      call ReadUInt32NewLine   ; nacitame vstup
6      push eax                 ; predavame parametr
7      call PyramidoveCislo     ; volame funkci
8      add  esp, 4              ; predany parametr mazeme ze zasobniku
9      call WriteUInt32NewLine  ; vypisujeme vysledek
10     ret
```

Rekurzivní řešení

```
1  PyramidoveCislo:
2      push ebp                  ; zalohujeme stare dno
3      mov  ebp, esp             ; vytvarime nove dno
4      push ebx                  ; zalohujeme registr EBX (zalohovat bychom meli i EDX)
5    if:
6      cmp  [ebp + 8], dword 0   ; je parametr N roven nule? (ukoncujici podminka)
7      je   then                 ; pokud ano tak skocime na THEN
8      jne  else                 ; pokud ne  tak skocime na ELSE
9    then:
10     mov  eax, 0               ; EAX = 0 (navratova hodnota)
11     jmp  end                  ; skoc na konec
12   else:
13     mov  eax, [ebp + 8]       ; EAX = N
14     mul  eax                  ; EDX:EAX = N*N (EDX budeme ignorovat)
15     mov  ebx, eax             ; EBX = N*N
16     mov  eax, [ebp + 8]       ; EAX = N
17     dec  eax                  ; EAX = N-1
18     push eax                  ; predavame parametr (N-1)
19     call PyramidoveCislo      ; volame funkci, po ktere EAX = f(N-1)
20     add  esp, 4               ; predany parametr mazeme ze zasobniku
21     add  eax, ebx             ; EAX = f(N-1) + N^2
22   end:
23     pop  ebx                  ; obnovujeme registr EBX (obnovovat bychom meli i EDX)
24     pop  ebp                  ; obnovujeme stare dno
25     ret
```