Externí funkce, lokální proměnné a desetinná čísla typu float

ISU-cv09

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

9. dubna 2025

Externí funkce

Externí funkce

Externí funkce jsou funkce implementované mimo náš zdrojový soubor:

- Například – funkce printf ze standardních knihoven jazyka C.
- Funkce z knihovny 'rw32.inc' nejsou externí, protože direktivou %include do našeho programu vkládáme její zdrojový kód (viz. rw32-2022.inc).

Externí funkce na začátku programu deklarujeme direktivou extern:

- Při jejich volání se řídíme konvencemi (viz. cv08, pro jazyk C konvence CDECL).
- Pokud funkce má návratovou hodnotu, tak bude vrácena v registru EAX (viz. cv08).

```
%include 'rw32.inc'          ;knihovna pro vstup a vystup
extern printf                ;deklarujeme externi funkci PRINTF

section .data                ;inicializovany segment
    arr db "Hello, World!", 10, 0 ; textovy retezec

section .text                ;kodovy segment
main:
    push arr                 ; parametr funkce predavame pres zasobnik
    call printf              ; volame externi funkci PRINTF
    add  esp, 4              ; mazeme predany parametr (1*4B)
    ret
```

Externí funkce v ISU-HUB

ISU-HUB podporuje volání jen u některých vybraných funkcí z jazyka C:

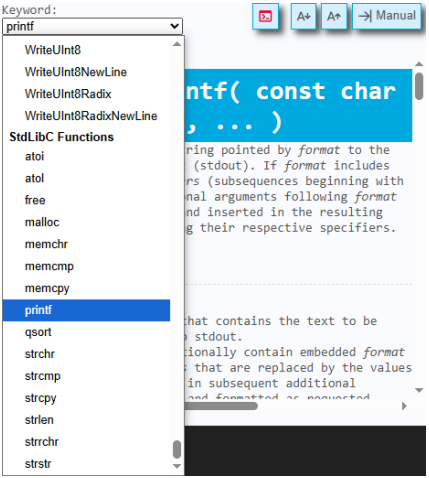
- Jiná vývojová prostředí (SASM) mohou vyžadovat dekoraci jmen, a direktivu extern může být potřeba nahradit za _extern nebo CEXTERN.
- Pozor – externí funkce **NEZÁLOHUJÍ** hodnoty registrů EAX, ECX a EDX!

```
%include 'rw32.inc' ;vkladame knihovnu
extern strlen       ;deklarujeme funkci

section .data       ;inicializovany segment
    arr db "Ahoj", 0; textovy retezec

section .text       ;kodovy segment
main:
    push arr        ; predavame parametr
    call strlen     ; volame funkci
    add  esp, 4     ; mazeme parametr (1*4B)

    call WriteInt32NewLine ; vypis EAX
    ;(navratova hodnota funkce STRLEN)
    ret
```



Na vyzkoušení

Program ze vstupu načte dvě 32b čísla a externí funkcí printf vypíše jejich součet:

```
%include 'rw32.inc'          ;knihovna pro vstup a vystup
extern printf                ; deklarujeme externi funkci PRINTF

section .data                ;inicializovany segment
    arr db "%i + %i = %i", 10, 0 ; textovy retezec
section .text                ;kodovy segment
main:
    call ReadInt32NewLine ; nacti prvni vstup (X)
    mov  ecx, eax         ; ECX = X
    call ReadInt32NewLine ; nacti druhy vstup (Y)
    mov  ebx, eax         ; EBX = Y
    add  eax, ecx         ; EAX = Y+X

    push eax              ; ctvrty parametr - %i (X+Y)
    push ebx              ; treti  parametr - %i (Y)
    push ecx              ; druhy  parametr - %i (X)
    push arr              ; prvni  parametr - vypisovany retezec
    call printf           ; volame externi funkci PRINTF
    add  esp, 16          ; mazeme predane parametry (4*4B)
    ret
```

Na procvičení

Vyzkoušejte si:

- Zkopírujte si šablonu a implementujte funkci BezpecnyMalloc, která ověří že jejím parametrem je kladné číslo, a pokud ano tak voláním externí funkce malloc dynamicky alokuje odpovídající množství bajtů paměti.
- Funkce je volána dle konvence CDECL (viz. cv08), její návratovou hodnotou je adresa alokované paměti, nebo nula (NULL) pokud k alokaci nedošlo.
- Úspěšnost alokace ošetřovat nemusíte – ISU-HUB při neúspěchu havaruje.
- Zálohování hodnot používaných registrů (EAX, EBX, ECX, ...) můžete ignorovat.

Například:

- 10 => 267248
- 0 => 0
- -10 => 0

```
⊖ Stack content
[ESP+12] = [EBP+12] = [0x000413BC] =          13, 0x0000000D   > rw32_exit
[ESP+08] = [EBP+08] = [0x000413B8] =          10, 0x0000000A
[ESP+04] = [EBP+04] = [0x000413B4] =        3272, 0x00000CC8   > line 12
[ESP] =    [EBP] =    [0x000413B0] =  -84215046, 0xFAFAFAFA
```

Nápověda:

- Aby funkce BezpecnyMalloc mohla číst parametry ze zásobníku, musí si vytvořit svůj vlastní zásobníkový rámec (viz. cv08).
- Pokud BezpecnyMalloc nevrátí správnou hodnotu, tak funkce free havaruje.
- Návratové hodnoty se vždy vracejí v registru EAX.

Šablona

```
%include 'rw32.inc'          ;knihovna pro vstup a vystup
extern malloc                ;deklarujeme externi funkci MALLOC
extern free                  ;deklarujeme externi funkci FREE
section .text                ;kodovy segment
main:
    call ReadInt32NewLine    ; EAX = vstup (kolik bajtu budeme alokovat)
    push eax                 ; EAX ulozime na zasobnik jako parametr funkce
    mov  eax, -1             ; (do EAX nastavujeme hodnotu -1 aby bylo)
                             ; (poznat jestli funkce BezpecnyMalloc)
                             ; (nastavuje spravnou navratovou hodnotu)
    call BezpecnyMalloc      ; volame funkci BEZPECNYMALLOC
    add  esp, 4              ; mazeme predany parametr

    call WriteInt32NewLine ;vypisujeme navratovou hodnotu funkce

    cmp  eax, 0              ; navratovou hodnotu funkce porovname s nulou
    je   .skip               ; pokud EAX == 0 tak skoc
    push eax                 ; navratovou hodnotu pouzij jako parametr
    call free                ; volame funkci FREE
    add  esp, 4              ; mazeme predany parametr
  .skip:
    ret
```

Lokální proměnné

Lokální proměnné

Lokální proměnné slouží pro ukládání dočasných dat:

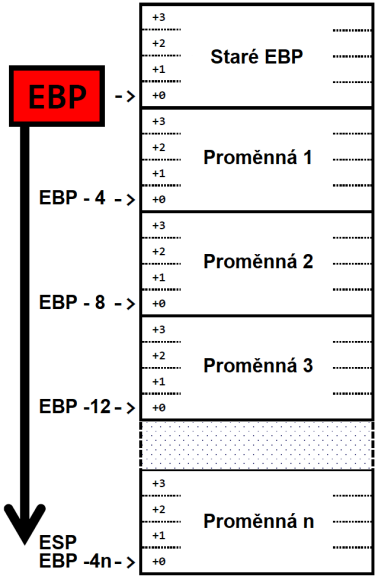
- Alokujeme je po vytvoření nového dna (EBP) posunutím vrcholu zásobníku (ESP) směrem dolů.
- Pro každou proměnnou alokujeme 32b paměti.

```
push ebp        ; zalohuj stare dno
mov  ebp, esp   ; vytvor nove dno
sub  esp, 8     ; alokujeme dve lokalni promenne
```

- Na konci funkce, před obnovením starého dna, alokované lokální proměnné musíme také uvolnit přepsáním adresy vrcholu (ESP) adresou dna (EBP).

```
mov  esp, ebp   ; uvolni vsechny lokalni promenne
pop  ebp        ; obnov stare dno
ret             ; konec funkce
```

- Pro vytvoření rámce a alokaci proměnných můžeme používat také instrukce ENTER a LEAVE (viz. p6s29).



Proměnné vs parametry

Proměnné adresujeme pomocí dna se záporným offsetem: [EBP–4], [EBP–8], atd. :

- Parametry funkce jsme adresovali s kladným offsetem: [EBP+8], [EBP+12], atd.
- Registry zálohujeme vždy až po vytvoření rámce a alokaci proměnných.

```
foo:
    push ebp          ; zalohuj stare dno
    mov  ebp, esp     ; vytvor nove dno
    sub  esp, 8       ; alokujeme dve lokalni promenne (2*4B)
    ; tady se zalohuji registry

    ; [EBP + 12]      ; druhy parametr funkce
    ; [EBP +  8]      ; prvni parametr funkce
    ; [EBP +  4]      ; navratova adresa do volajici funkce
    ; [EBP +  0]      ; zalohovana hodnota stareho dna
    ; [EBP -  4]      ; prvni lokalni promenna
    ; [EBP -  8]      ; druha lokalni promenna

    ; tady se obnovuji registry
    mov  esp, ebp     ; uvolni vsechny lokalni pormenne
    pop  ebp          ; obnov stare dno
    ret               ; konec funkce
```

Na vyzkoušení (1/2)

Funkce main pro dvě různá pole zavolá funkci PocetNul, která spočítá kolik pole obsahuje nul, a jako počítadlo při tom použije lokální proměnnou [EBP–4]:

```
%include 'rw32.inc'          ;knihovna pro vstup a vystup
section .data                ;inicializovany segment
    arr1 dd 0, 1, 2, 3       ; prvni pole
    arr2 dd 0, 1, 0, 1, 0    ; druhe pole

section .text                ;kodovy segment
main:
    push arr1                ; druhy parametr funkce (adresa pole)
    push dword 4             ; prvni parametr funkce (delka pole)
    call PocetNul            ; volani funkce PocetNul
    add  esp, 8              ; mazeme predane parametry (2*4B)
    call WriteInt32NewLine   ; vypisujeme navratovou hodnotu

    push arr2                ; jine pole
    push dword 5             ; jina delka
    call PocetNul            ; volani funkce
    add  esp, 8              ; mazani parametru
    call WriteInt32NewLine   ; vypis
    ret
```

Na vyzkoušení (2/2)

```
PocetNul:
    push ebp                    ; zalohuj stare dno
    mov  ebp, esp               ; vytvor nove dno
    sub  esp, 4                 ; alokuj misto pro jednu lokalni promennou

    mov  edi, [ebp + 12]        ; EDI = adresa prohledavaneho pole
    mov  ecx, [ebp +  8]        ; ECX = pocet opakovani (delka pole)
    mov  eax, 0                 ; EAX = hledana hodnota
    mov  [ebp - 4], dword 0     ; pocitadlo = 0 (pocatecni hodnota)
    cld                         ;  DF = 0 (posun doprava)

  .for:
    scasd                       ; EAX porovnej s aktualni polozkou pole [EDI]
    jne  .skip                  ; pokud EAX != [EDI] tak skoc
    inc  dword [ebp-4]          ; inkrementuj pocitadlo
  .skip:
    loop .for                   ; dekrementuj ECX a pokud ECX!=0 tak opakuj
    mov  eax, [ebp-4]           ; EAX = pocitadlo (navratova hodnota)

    mov  esp, ebp               ; uvolni lokalni promenne
    pop  ebp                    ; obnov stare dno
    ret
```

Na procvičení

Vyzkoušejte si:

- Zkopírujte si šablonu a implementujte funkci ObratVypis která ze vstupu načte několik 32b čísel, a v obráceném pořadí je vypíše na výstup.
- Parametry funkce jsou velikost a počet čísel, předané dle konvence CDECL.
- Pro uložení načtených čísel použijte lokální proměnné, ne instrukce PUSH a POP.
- Zálohování registrů a ověřování platnosti parametrů můžete ignorovat.

Například:

- 10, 20, 30, 40, 50 => 50, 40, 30, 20, 10

```
⊖ Stack content
[ESP+36] = [EBP+16] = [0x0004138C] =            13, 0x0000000D  > rw32_exit
[ESP+32] = [EBP+12] = [0x00041388] =            20, 0x00000014
[ESP+28] = [EBP+08] = [0x00041384] =             5, 0x00000005
[ESP+24] = [EBP+04] = [0x00041380] =          3265, 0x00000CC1  > line 8
[ESP+20] = [EBP] =    [0x0004137C] =     -84215046, 0xFAFAFAFA
[ESP+16] = [EBP-04] = [0x00041378] =             0, 0x00000000
[ESP+12] = [EBP-08] = [0x00041374] =             0, 0x00000000
[ESP+08] = [EBP-12] = [0x00041370] =             0, 0x00000000
[ESP+04] = [EBP-16] = [0x0004136C] =             0, 0x00000000
[ESP] =    [EBP-20] = [0x00041368] =             0, 0x00000000
```

Parametry

Lokalni promenne

Nápověda:

- Pozor – první lokální proměnná [EBP–4] je na konci pole.
  poslední lokální proměnná [EBP–N*4] je na začátku pole.
- Problém vyřešíme dvěma samostatnými cykly:
  První cyklus jde z leva do prava, čte ze vstupu a zapisuje do pole (STOS).
  Druhý cyklus jde z prava do leva, čte z pole a vypisuje na výstup (LODS).

Šablona

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

section .text                ;kodovy segment
main:
    push dword 20            ; velikost alokovaneho pole (N*4)
    push dword 5             ; pocet nacitanych cisel    (N)
    call ObratVypis          ; volame funkci
    add  esp, 8              ; predany parametr mazeme ze zasobniku
    ret

ObratVypis:
    push ebp                 ; zalohuj stare dno
    mov  ebp, esp            ; vytvor nove dno
    sub  esp, [ebp +12]      ; alokujeme misto pro lokalni promenne (N*4B)

    ;???

    mov  esp, ebp            ; uvolnujeme lokalni promenne
    pop  ebp                 ; obnovujeme stare dno
    ret
```

Desetinná čísla typu float

Desetinná čísla

Celá čísla (např. 10, datový typ int) jsou ukládána v doplňkovém kódu (viz. cv01):

- První bit čísla určuje znaménko, a můžeme ho obrátit dvojkovým doplňkem.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

Desetinná čísla (např. 10.0, datový typ float) jsou ukládána s plovoucí řádovou čárkou dle standardu IEEE 754, podle kterého se zápis čísla rozděluje na tři části:

- Znaménko (1b, S) – pro kladná čísla 0, pro záporná čísla 1.
- Exponent (8b, E) – udává mocninu základu čísla, a je uložen v kódu transformované nuly (viz. p1s40) s posunem o hodnotu 127.
- Mantisa (23b, M) – představuje desetinnou část čísla, a až na výjimky je normalizovaná – první jednička je implicitní a do mantisy se nepíše.

0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Pro normalizované desetinné číslo uložené jako 32b float platí rovnice:

$$X = -1^{S} * 2^{E-127} * (1.M)$$

Normalizovaná čísla

Znaménko určuje jestli je číslo kladné nebo záporné:

- 1.0 = 0 01111111 00000000000000000000000
- -1.0 = 1 01111111 00000000000000000000000

Exponent určuje mocninu základu čísla (2^{E-127}):

- 0.5 = 0 01111110 00000000000000000000000
- 1.0 = 0 01111111 00000000000000000000000
- 2.0 = 0 10000000 00000000000000000000000

Jednotlivé bity mantisy, představují polovinu, čtvrtinu, osminu, šestnáctinu, atd. :

- 1.0 = 0 01111111 00000000000000000000000
- 1.25 = 0 01111111 01000000000000000000000
- 1.5 = 0 01111111 10000000000000000000000
- 1.75 = 0 01111111 11000000000000000000000
- 2.0 = 0 10000000 00000000000000000000000
- 2.25 = 0 10000000 00100000000000000000000

Převod z desítkové soustavy na 32b float

Například, převod čísla -5.375 z desítkové soustavy do plovoucí řádové čárky:

- Znaménko určíme podle toho jestli je číslo kladné (S = 0) nebo záporné (S = 1).

-5.375 => S = 1

- Absolutní hodnotu čísla převedeme z desítkové do dvojkové (viz. cv01).

$$(5.375)_{10} = (101.011)_2$$

- Posunutím desetinné tečky číslo normalizujeme do podoby "1.mantisa".

$(101.011)_2$ => 1.01011

- Exponent spočítáme z posuvu dle rovnice "E-127 = posuv".

E-127 = 2 => E = $(129)_{10}$ = $(10000001)_2$

- Znaménko, exponent a mantisu spojíme, a zprava doplníme nulami na 32b.

	Zn.	Exponent	Mantisa
-5.375 =	1	10000001	01011000000000000000

Převod z 32b float do desítkové soustavy

Například, převod čísla 11000000101011000000000000000 do desítkové soustavy:

- Z exponentu spočítáme velikost posuvu, dle rovnice "E-127 = posuv".

$$E = (10000001)_2 = (129)_{10} \quad => \quad 129\text{-}127 = \text{posuv} = 2$$

- Z mantisy vytvoříme normalizované číslo ve formátu "1.mantisa".

$$1.01011$$

- Posunutím desetinné tečky číslo de-normalizujeme.

$$1.01011 \quad => \quad (101.011)_2$$

- Převodem z dvojkové do desítkové (viz. cv01) získáme absolutní hodnotu.

$$(101.011)_2 = (5.375)_{10}$$

- Podle znaménka určíme jestli je číslo kladné (S = 0) nebo záporné (S = 1).

Zn.	Exponent	Mantisa
1	10000001	01011000000000000000 = -5.375

Na procvičení

Desetinné číslo převeďte z desítkové soustavy na 32b float:

- -3.0 = ???
- 2.25 = ???
- 31.0 = ???
- -0.875 = ???

Desetinné číslo převeďte z 32b float do desítkové soustavy:

- ??? = 0 10000001 11100000000000000000000
- ??? = 0 10000101 01000000000000000000000
- ??? = 1 10000010 00000100000000000000000
- ??? = 0 01111011 00000000000000000000000

Speciální hodnoty

Speciální hodnoty jsou definovány pomocí následujících konstant:

- $+0.0$ = 0 00000000 00000000000000000000000 je kladná nula
- -0.0 = 1 00000000 00000000000000000000000 je záporná nula
- $+\infty$ = 0 11111111 00000000000000000000000 je kladné nekonečno
- $-\infty$ = 1 11111111 00000000000000000000000 je záporné nekonečno
- qNaN = ? 11111111 1 cokoliv kromě samých nul je tiché nečíslo
- sNaN = ? 11111111 0 cokoliv kromě samých nul je signalizující nečíslo
- $\pm???$ = ? 00000000 cokoliv kromě samých nul je denormalizované číslo

Datové typy desetinných čísel

Assembler u desetinných čísel rozeznává několik datových typů:

Velikost		Segment			Jméno		přesnost
bity	Bajty	.bss	.data	.text	typ	velikost	
32	4	resd	dd	dword	float	double-word	single
64	8	resq	dq	qword	double	quad-word	double
80	10	rest	dt	tword	long double	ten-word	extended

Aby se číslo v paměti uložilo jako desetinné, musí obsahovat desetinnou tečku:

- Pozor – celá a desetinná čísla mají jiný rozsah platných velikostí!
- Celá – 8b, 16b, 32b, 64b.
- Desetinná – 32b, 64b, 80b.

```
section .data
    K db 10      ;  8b cele       cislo jmenem K s hodnotou 10
    L dw 20      ; 16b cele       cislo jmenem L s hodnotou 20
    M dd 30      ; 32b cele       cislo jmenem M s hodnotou 30
    N dq 40      ; 64b cele       cislo jmenem N s hodnotou 40

    R dd 50.0    ; 32b desetinne cislo jmenem R s hodnotou 50.0
    S dq 60.0    ; 64b desetinne cislo jmenem S s hodnotou 60.0
    T dt 70.0    ; 80b desetinne cislo jmenem T s hodnotou 70.0
```

Načítání a vypisování typu float

32b desetinné číslo (float) načteme a vypíšeme funkcemi ReadFloat a WriteFloat:

```
call ReadFloat                  ; do registru EAX nacti 32b float
call WriteFloat                 ; z  registru EAX vypis 32b float
```

V kódovém segmentu floaty musíme obalit makrem __float32__() z překladače NASM:

```
mov  eax, __float32__(0.5)      ; do registru EAX uloz 32b float (0.5)
mov  eax, 0.5                   ; CHYBA - samotna desetinna cisla v
                                ;         kodovem segmentu nelze pouzit
```

Speciální hodnoty vytvoříme pomocí doplňujících maker:

```
mov eax, __float32__(__Infinity__)    ; EAX = kladne nekonecno
mov ebx, __float32__(-__Infinity__)   ; EBX = zaporne nekonecno
mov ecx, __float32__(__QNaN__)        ; ECX = tiche necislo
mov edx, __float32__(__SNaN__)        ; EDX = signalizujici necislo
```

Při ukládání do proměnné nebo na zásobník musíme použít pseudo-instrukci dword:

```
mov  [X], dword __float32__(10.0)     ; promenna X = 10.0
push dword __float32__(__Infinity__) ; na zasobnik uloz kladne nekonecno
```

Na vyzkoušení

Desetinná čísla **NELZE** počítat pomocí běžných celočíselných instrukcí (viz. cv11):

```
%include 'rw32.inc'                 ;knihovna pro vstup a vystup
section .text                       ;kodovy segment
main:                                     ;normalizovana cisla
    mov  eax, __float32__(1.0)            ; 0|01111111|00000000000000000000000
    mov  eax, __float32__(2.0)            ; 0|10000000|00000000000000000000000
    mov  eax, __float32__(3.0)            ; 0|10000000|10000000000000000000000
    mov  eax, __float32__(10.0)           ; 0|10000010|01000000000000000000000
    mov  eax, __float32__(0.1)            ; 0|01111011|10011001100110011001101

                                          ;specialni hodnoty
    mov  eax, __float32__(__Infinity__);0|11111111|00000000000000000000000
    mov  eax, __float32__(0.0)            ; 0|00000000|00000000000000000000000
    mov  eax, __float32__(-0.0)           ; 1|00000000|00000000000000000000000

    mov  eax, __float32__(1.0)  ; EAX = 0|01111111|00000000000000000000000
    add  eax, eax               ; EAX = EAX + EAX (celociselny soucet)
    call WriteFloatNewLine      ; EAX = 0|11111110|00000000000000000000000
    ret                         ;170141183460469231731687303715884105728.0
```