Globální proměnné, pole a indexové registry

ISU-cv03

Ing. Jakub Husa

Vysoké Učení Technické v Brně, Fakulta informačních technologií
Božetěchova 1/2. 612 66 Brno - Královo Pole
ihusa@fit.vutbr.cz

23. února 2023

Globální proměnné

Datové segmenty

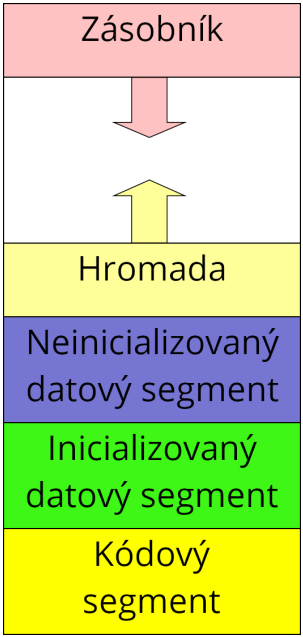
Paměťový prostor programu obsahuje dva datové segmenty:

- Neinicializovaný – označovaný jako section .bss obsahuje globální proměnné s neznámou počáteční hodnotou.
- Inicializovaný – označovaný jako section .data obsahuje globální proměnné se známou počáteční hodnotou.
- Lokální proměnné vytváříme na zásobníku (viz. cv08).

Každá proměnná musí mít:

- Identifikátor – označuje adresu proměnné v paměti počítače.
- Datový typ – určuje velikost proměnné v paměti počítače, a v každém segmentu se označuje jinou pseudo-instrukcí.

Velikost		Segment			Registry
bity	Bajty	.bss	.data	.text	
8	1	resb	db	byte	AH AL BH BL ...
16	2	resw	dw	word	AX BX CX DX
32	4	resd	dd	dword	EAX EBX ECX EDX
64	8	resq	dq	qword	???



Rezervování a definice proměnné

V ne-inicializovaném segmentu rezervujeme název, velikost a počet položek:

```
1 section .bss       ;ne-inicializovany datovy segment
2     X resb 1       ; jedna  8b promenna jmenem X
3     Y resw 1       ; jedna 16b promenna jmenem Y
4     Z resd 1       ; jedna 32b promenna jmenem Z
```

V inicializovaném segmentu definujeme název, velikost a počáteční hodnotu:

```
5 section .data      ;inicializovany datovy segment
6     K db 10        ; jedna  8b promenna jmenem K s hodnotou 10
7     L dw 20        ; jedna 16b promenna jmenem L s 16b hodnotou 20
8     M dd 30        ; jedna 32b promenna jmenem M s 32b hodnotou 30
```

Aby se proměnné správně zobrazily v debuggeru musíme nastavit jejich velikost:

- Byte (b), Word (w), Double-word (d), Quad-word (q).

Na cvičeních pracujeme s procesory z rodiny x86 v 32b režimu:

- V tomto režimu se paměť počítače chová jako jedno 4 GiB velké pole.
- Proměnné adresujeme přímým adresováním jako položky paměti počítače pomocí identifikátoru a hranatých závorek.

```
1  section .data       ;inicializovany datovy segment
2      X db 0          ; promenna jmenem X, velikosti  8b, s hodnotou 0
3      Y dw 0          ; promenna jmenem Y, velikosti 16b, s hodnotou 0
4
5  section .text       ;kodovy segment
6  _main:              ; zacatek funkce main (zacatek programu)
7
8      mov   al, [X]   ; do registru AL uloz hodnotu z promenne  X
9      mov  [X],  al   ; do promenne  X uloz hodnotu z registru AL
10
11     mov   bx, [Y]   ; do registru BX uloz hodnotu z promenne  Y
12     mov  [Y],  bx   ; do promenne  Y uloz hodnotu z registru BX
13
14     ret             ; konec funkce main (konec programu)
```

Vyzkoušejte si:

- Vytvořte si jednu neinicializovanou 32b proměnnou (X).
- Ze vstupu načtěte jedno 32b číslo, uložte ho do proměnné, a její hodnotu zobrazte v debuggeru.

Vyzkoušejte si:

- Vytvořte si dvě inicializované 8b proměnné (Y = 10 a Z = 20).
- Součet obou proměnných vypište na výstup.

Omezení sběrnice

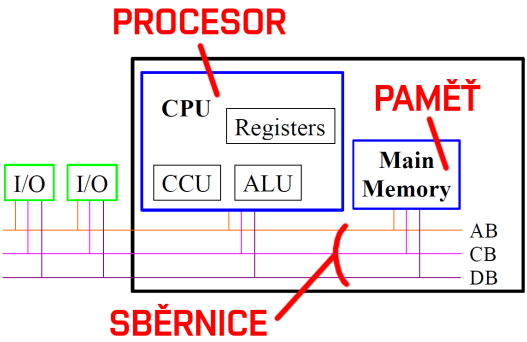
Procesor a operační paměť jsou dvě samostatná zařízení propojená sběrnicí:

- Sběrnice nedokáže adresovat více proměnných současně.
- V každé instrukci můžeme používat maximálně jedny hranaté závorky.

Například, součet $X = X + Y$:

```
1 ;CHYBA - nelze pouzit vicero zavorek
2     add   [X], [Y]
3
4 ;spravne - pouzivame registry
5     mov   eax, [Y] ;EAX = Y
6     add   [X], eax ;   X = X + EAX
```

Zjednodušené blokové schéma počítače



CPU	Central Processing Unit (procesor)
ALU	Arithmetic and Logic Unit
CCU	Central Control Unit (řadič)
I/O	Input/Output Unit
AB, CB, DB	Address Bus, Control Bus, Data Bus

Omezení překladače

Většina instrukcí existuje ve třech velikostech (8b, 16b, 32b):

- Správnou velikost určuje překladač podle velikosti používaných registrů.

```
1  mov   al,  10          ; 8b instrukce MOV
2  mov   ax,  10          ;16b instrukce MOV
3  mov  eax,  10          ;32b instrukce MOV
4  mov   al, eax          ;CHYBA - dva registry ruzne velikosti
5  mov  [X],  10          ;CHYBA - zadny registr, nelze odvodit velikost
```

Pokud registry nepoužíváme velikost instrukce si musíme určit sami:

```
6  section .data
7      X db 0             ; 8b promenna X s pocatecni hodnotou 0
8      Y dw 0             ;16b promenna Y s pocatecni hodnotou 0
9      Z dd 0             ;32b promenna Z s pocatecni hodnotou 0
10
11 section .text
12     mov byte  [X], 10 ;pouzij  8b instrukci MOV
13     mov word  [Y], 10 ;pouzij 16b instrukci MOV
14     mov dword [Z], 10 ;pouzij 32b instrukci MOV
```

Na procvičení

Vyzkoušejte si:

- Vytvořte si tři inicializované 16b proměnné (K = 10, L = 20, M = 30).
- Napište program který hodnoty změní na K = K+L+M, L = M+100, M = -M
- hodnoty všech proměnných zobrazte v debuggeru.

Pole

V ne-inicializovaném segmentu rezervujeme název, velikost a počet položek:

```
1 section .bss                  ;NE-inicializovany datovy segment
2     arrX resb 4               ; pole ctyr  8b promennych jmenem arrX
3     arrY resw 4               ; pole ctyr 16b promennych jmenem arrY
4     arrZ resd 4               ; pole ctyr 32b promennych jmenem arrZ
```

V inicializovaném segmentu definujeme název, velikost a počáteční hodnoty:

```
5 section .data                 ;inicializovany datovy segment
6     arrK db 10, 20, 30, 40 ; pole varK s  8b hodnotami 10, 20, 30, 40
7     arrL dw 10, 20, 30, 40 ; pole varL s 16b hodnotami 10, 20, 30, 40
8     arrM dd 10, 20, 30, 40 ; pole varM s 32b hodnotami 10, 20, 30, 40
```

Aby se položky pole správně zobrazily v debuggeru musíme nastavit Array size.

Položky pole adresujeme identifikátorem posunutým o konstantu:

- Položky pole jsou vždy indexovány od nuly.
- Velikost posuvu udáváme v bajtech (1, 2, 4) ne položkách!

```
1  section .bss              ;neinicializovany segment
2      arrX resb 4           ; pole ctyr  8b polozek
3      arrY resw 4           ; pole ctyr 16b polozek
4
5  section .text             ;kodovy segment
6  _main:
7      mov  [arrX + 0],  al ; prvni   8b polozka je na adrese +0
8      mov  [arrX + 1],  bl ; druha   8b polozka je na adrese +1
9      mov  [arrX + 2],  cl ; treti   8b polozka je na adrese +2
10     mov  [arrX + 3],  dl ; ctvrta  8b polozka je na adrese +3
11
12     mov  [arrY + 0],  ax ; prvni  16b polozka je na adrese +0
13     mov  [arrY + 2],  bx ; druha  16b polozka je na adrese +2
14     mov  [arrY + 4],  cx ; treti  16b polozka je na adrese +4
15     mov  [arrY + 6],  dx ; ctvrta 16b polozka je na adrese +6
16     ret
```

Vyzkoušejte si:

- Vytvořte si neinicializované pole tří 32b čísel (arr).
- Napište program který ze vstupu načte dvě 32b čísla (R a S), a do pole zapíše hodnoty R, S, a R-S.
- Obsah pole zobrazte v debuggeru.

Vyzkoušejte si:

- Vytvořte si neinicializované pole čtyř 16b čísel (dst).
- Vytvořte si inicializované pole čtyř 16b čísel (src = {10, 20, 30, 40}).
- Napište program který do položek pole dst zapíše částečné sumy pole src.
- Obsahy obou polí zobrazte v debuggeru.

Například:

- src = {10, 20, 30, 40} => dst = {10, 30, 60, 100}

Indexové registry

Procesor obsahuje dva indexové registry pro práci s řetězovými daty:

- Indexové registry používáme pro nepřímé adresování a umisťujeme do nich identifikátor (adresu, ukazatel) proměnné.
- Vypisujeme přes registr ESI (Source Index), čteme přes EDI (Destination Index).

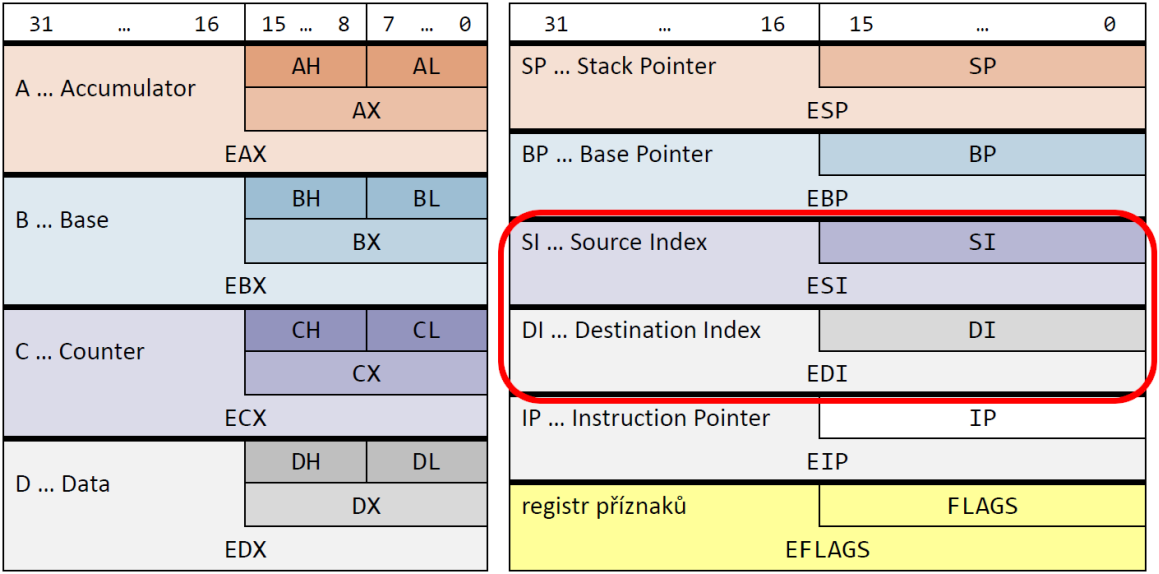
Nejjednodušším typem řetězových dat je textový řetězec:

- Řetězec je posloupnost 8b znaků zakončená znakem s hodnotu 0 (NULL).
- Do proměnné (pole) řetězec zadáváme pomocí dvojitých uvozovek.
- Řídící znaky zadáváme jejich hodnotou ASCII, zadávání znaků pomocí zpětného lomítka, jako '\0' (0) nebo '\n' (10) nefunguje.

```
1 section .data                     ;posloupnost znaku nasledovana
2     arr db "Hello World!", 10, 0 ;koncem radku (10) a koncem retezce (0)
```

Ne-textové řetězce si probereme společně s řetězovými instrukcemi (viz. cv09).

Indexové registry



Řetězce vypisujeme funkcí WriteString:

- Adresu vypisovaného řetězce zadáváme do ESI.
- Protože zadáváme adresu ne hodnotu, tak nepoužíváme hranaté závorky!

```
1  section .data
2      arrX db "Hello World!", 10, 0 ;vypisovany retezec
3  section .text
4      mov  esi, arrX        ;adresa promenne kterou budeme vypisovat
5      call WriteString      ;volani funkce pro vypis retezce
```

Řetězce načítáme funkcí ReadString:

- Adresu načítaného řetězce zadáváme do EDI.
- Maximální počet načítaných znaků zadáváme do registru EBX.
- Pozor, počet úspěšně načtených znaků funkce zapíše do registru EAX!

```
6  section .bss
7      arrY resb 6           ;rezervuj misto pro 5 znaku a NULL
8  section .text
9      mov  edi, arrY        ;adresa promenne do ktere budeme nacitat
10     mov  ebx, 5           ;maximalni pocet nacitanych znaku
11     call ReadString       ;volani funkce pro cteni retezce
```

Program vyzve uživatele aby zadal svoje jméno, a načte řetězec (max 20 znaků):

```
1  %include "rw32-2018.inc" ;knihova pro vstup a vystup
2
3  section .bss             ;neinicializovany segment
4      jmeno resb 21        ;rezervuj misto pro 20 znaku a NULL
5
6  section .data            ;inicializovany segment
7      vyzva db "Zadejte vase jmeno:", 10, 0 ;vypisovany retezec
8
9  section .text            ;kodovy segment
10 _main:
11     mov  esi, vyzva      ;adresa promenne kterou budeme vypisovat
12     call WriteString     ;vypis retezec
13
14     mov  edi, jmeno      ;adresa promenne do ktere budeme nacitat
15     mov  ebx, 20         ;nacitat budeme maximalne 20 znaku
16     call ReadString      ;nacti retezec
17
18     ret
```

Vyzkoušejte si:

- Vytvořte si dostatečně dlouhé pole a načtěte do něj řetězec čtyř znaků.
- V řetězci vyměňte první a poslední znak, a upravený řetězec vypište.

Například:

- ABCD => DBCA