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

21. února 2024

Globální proměnné

Datové segmenty

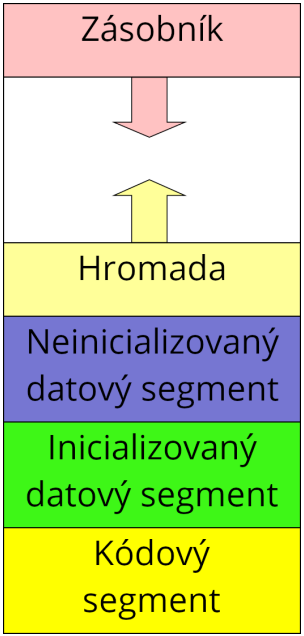
Paměťový prostor programu obsahuje dva datové segmenty:

- 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 se neukládají do datových segmentů ale do zásobníku (viz. cv08).

Každá proměnná musí mít:

- Identifikátor – označuje adresu proměnné v paměti počítače.
- Datový typ – určuje velikost (ale ne význam) proměnné, 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 neinicializovaném 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 hodnotou 20
8     M dd 30        ; jedna 32b promenna jmenem M s hodnotou 30
```

V ISU-HUBu musíte neinicializované proměnné v debuggeru nastavit ručně:

- Watches -> add watch -> JMENO PROMENNE -> OZUBENE KOLECKO -> pointer.

V SASMu musíte u proměnných v debuggeru nastavit jejich velikost:

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

Adresování proměnné

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

- V 32b režimu se operační 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 10         ; promenna jmenem X, velikosti  8b, s hodnotou 10
3      Y dw 20         ; promenna jmenem Y, velikosti 16b, s hodnotou 20
4      Z dd 30         ; promenna jmenem Z, velikosti 32b, s hodnotou 30
5
6  section .text       ;kodovy segment
7  main:
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     mov  ecx, [Z]   ; do registru ECX uloz hodnotu z promenne   Z
15     mov  [Z], ecx   ; do promenne   Z uloz hodnotu z registru ECX
16     ret
```

Vyzkoušejte si:

- Vytvořte si jednu neinicializovanou 16b proměnnou (X)
  a dvě inicializované 8b proměnné (Y = 10, Z = 20).
- Ze vstupu načtěte číslo do proměnné X a její hodnotu zobrazte v debuggeru.
- Spočítejte součet proměnných Y a Z a jeho výsledek vypište na výstup.

Omezení sběrnice

Procesor a paměť počítače jsou dvě samostatná zařízení propojená sběrnicí:

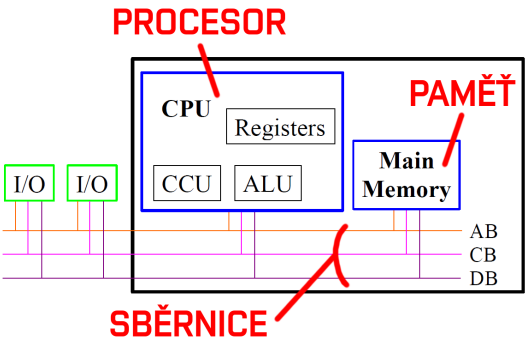
- Sběrnice **NEDOKÁŽE** adresovat více proměnných současně.
- Každá instrukce smí používat maximálně jedny hranaté závorky.

Například:

- Výpočet 32b součtu X = X + Y.

```
1  ;instrukce se dvojtymi zavorkami
2      add   [X], [Y] ; CHYBA
3
4  ;max jedny zavorky na instrukci
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):

- Kterou velikost instrukce použít odvozuje překladač – podle použitých 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 operandem není registr velikost instrukce musíme určit pseudo-instrukcí.

```
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
```

Vyzkoušejte si:

- Vytvořte si tři inicializované 16b proměnné (K = 10, L = 20, M = 30).
- Spočítejte nové hodnoty K = K+L+M, L = M+100, M = -M.
- Hodnoty všech proměnných si zobrazte v debuggeru.

Například:

- K = 10 => K = 60
  L = 20 => L = 130
  M = 30 => M = -30

Pole

Rezervování a definice polí

V neinicializovaném 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 obsah pole správně zobrazil v debuggeru, musíme nastavit počet jeho položek:

- V ISU-HUBu nastavujeme parametr Count.
- V SASMu nastavujeme parametr Array size.

Adresování polí

K adresování položek pole používáme indetifikátor a posuv (offset):

- Adresa první položky je shodná s adresou pole – její posuv je nula.
- Velikost posuvu vždy udáváme v **BAJTECH** (1, 2, 4), ne v počtu položek!

```
1  section .bss              ;ne-inicializovany 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 čtyř 32b čísel (dst).
- Vytvořte si inicializované pole čtyř 32b čísel (src = {10, 20, 30, 40}).
- Do položek pole dst zapište částečné sumy položek pole src.
- Obsah pole dst si zobrazte v debuggeru.

Například:

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

Indexové registry

Procesor obsahuje dva indexové registry (ESI a EDI) pro práci s řetězovými daty:

- ESI (Source Index) – ze které adresy v paměti budeme číst.
- EDI (Destination Index) – na kterou adresu v paměti budeme zapisovat.

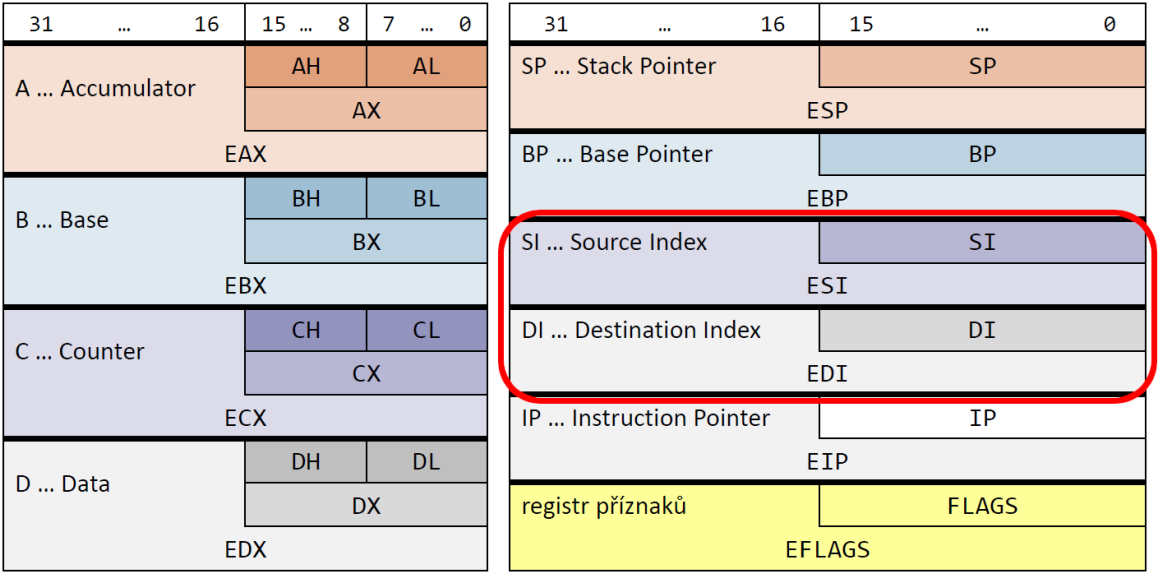
Nejjednodušším typem řetězových dat je textový řetězec:

- Řetězec je posloupnost 8b znaků zakončená znakem s hodnotu 0 (NUL).
- Znaky a řetězce ohraničujeme uvozovkami (`' '`, `" "`).
- Speciální řídící znaky zadáváme jejich hodnotou v tabulce ASCII, zadávání pomocí zpětného lomítka `'\0'` (0) nebo `'\n'` (10) nefunguje.

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

Ostatní typy řetězových dat (vektory čísel) si probereme později (viz. cv07).

Registry procesoru - indexové registry



Načítání a vypisování řetězců

Řetězec vypíšeme funkcí WriteString z knihovny rw32.inc:

- Funkce vypíše řetězec z adresy uložené v registru ESI.
- Protože do registrů ESI a EDI dáváme adresy, tak nepoužíváme hranaté závorky!

```
1   mov   esi, src            ; adresa promenne kterou budeme vypisovat
2   call  WriteString         ; volani funkce pro vypis retezce
```

Řetězec načteme funkcí ReadString z knihovny rw32.inc:

- Funkce na adresu uloženou v registru EDI načte až EBX znaků, a počet úspěšně načtených znaků zapíše do EAX.

```
3   mov   edi, dst            ; adresa promenne do ktere budeme nacitat
4   mov   ebx, 5              ; maximalni pocet nacitanych znaku
5   call  ReadString          ; volani funkce pro cteni retezce
```

Obě funkce mohou pro přehlednost ještě vypsat konec řádku:

```
6   call  ReadStringNewLine   ; nacti retezec a vypis konec radku
7   call  WriteStringNewLine  ; vypis retezec a konec radku
```

Vyzkoušejte si:

- Program ze vstupu načte řetězec až devíti znaků, a vypíše ho na výstup:

```
1  %include 'rw32.inc'    ;knihovna pro vstup a vystup
2
3  section .bss           ;ne-inicializovany segment
4      arr resb 10        ; rezervujeme misto pro 9 znaku a NUL
5
6  section .text          ;kodovy segment
7  main:
8      mov  edi, arr      ; EDI = adresa ciloveho pole (kam nacitame)
9      mov  ebx, 9        ; EBX = maximalni pocet nacitanych znaku
10     call ReadStringNewLine  ; nacti retezec a vypis konec radku
11
12     mov  esi, arr      ; ESI = adresa zdrojovheo pole (odkud vypisujeme)
13     call WriteStringNewLine ; vypis retezec a konec radku
14
15     ret
```

Vyzkoušejte si:

- Vytvořte si neinicializované pole, a ze vstupu do něj načtěte řetězec pěti znaků.
- Znaky řetězce přeskládejte do opačného pořadí, a vypište ho na výstup

Například:

- ABCDE => EDCBA