Globální proměnné, pole a indexové registry

ISU-cv03

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

28. února 2025

Globální proměnné

Datové segmenty

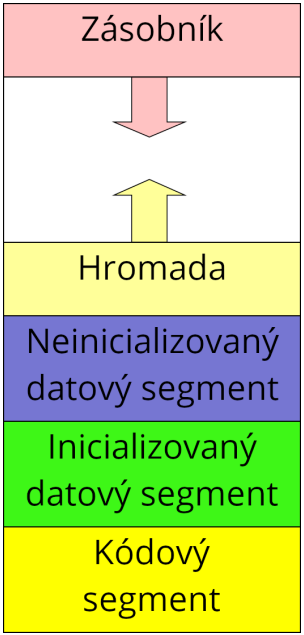
Paměťový prostor programu obsahuje dva datové segmenty:

- 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, a probereme si je později (viz. cv08).

Každá proměnná musí mít identifikátor a datový typ:

- Identifikátor – označuje adresu proměnné v paměti počítače.
- Datový typ – určuje velikost proměnné, ale ne její sémantiku (zda jde o číslo s nebo bez znaménka, ukazatel, znak, etc.).
- Typ v každém segmentu označujeme 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	nejsou



Inicializované proměnné

V inicializovaném segmentu definujeme (Define) název, velikost a počáteční hodnotu:

- Inicializované proměnné v debuggeru uvidíme automaticky (Auto-watches).

```
1  section .data      ;inicializovany datovy segment
2      R db 10        ; promenna jmenem R velikosti  8b s hodnotou 10
3      S dw 20        ; promenna jmenem S velikosti 16b s hodnotou 20
4      T dd 30        ; promenna jmenem T velikosti 32b s hodnotou 30
5      U dq 40        ; promenna jmenem U velikosti 64b s hodnotou 40
```

V 32b režimu (viz. cv02) se operační paměť počítače chová jako jedno 4 GiB velké pole:

- Proměnné vždy adresujeme jako položky paměti počítače, pomocí jejich identifikátoru a hranatých závorek.

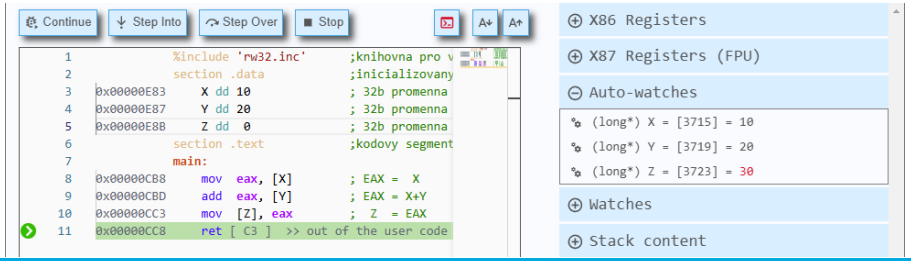
```
6  section .text              ;kodovy segment
7  main:
8      mov   al, [R]          ; do registru  AL uloz hodnotu z promenne R
9      call WriteInt8NewLine  ; vypis obsah registru  AL
10     mov   ax, [S]          ; do registru  AX uloz hodnotu z promenne S
11     call WriteInt16NewLine ; vypis obsah registru  AX
12     mov  eax, [T]          ; do registru EAX uloz hodnotu z promenne T
13     call WriteInt32NewLine ; vypis obsah registru EAX
14     ret
```

Na vyzkoušení

Program sečte hodnoty 32b proměnných X a Y a výsledek uloží do 32b proměnné Z:

- Výslednou hodnotu proměnné Z si zkontrolujte v debuggeru (Auto-watches).

```
%include 'rw32.inc'        ;knihovna pro vstup a vystup
section .data              ;inicializovany segment
    X dd 10                ; 32b promenna X s hodnotou 10
    Y dd 20                ; 32b promenna Y s hodnotou 20
    Z dd  0                ; 32b promenna Z s hodnotou 0
section .text              ;kodovy segment
main:
    mov  eax, [X]          ; EAX =  X
    add  eax, [Y]          ; EAX = X+Y
    mov  [Z], eax          ;  Z  = EAX
    ret
```

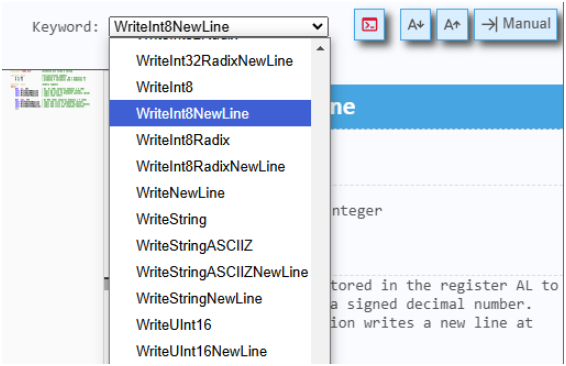


Vyzkoušejte si:

- Vytvořte si inicializovanou 8b proměnou (X = 65) a 32b proměnou (Y = -1).
- Proměnnou X vypište na výstup jako:
  číslo se znaménkem, číslo ve dvojkové soustavě, a jako znak.
- Proměnnou Y vypište na výstup jako:
  číslo se znaménkem, číslo v šestnáctkové soustavě, a jako číslo bez znaménka.

Například:

- X = 65 => 65
  => 01000001
  => A
- Y = -1 => -1
  => FFFFFFFF
  => 4294967295



Nápověda:

- Formát výpisu určujeme zavoláním správné funkce z knihovny rw32.inc.

Omezení sběrnice

Procesor a paměť počítače jsou dvě samostatná zařízení propojená sběrnicí:

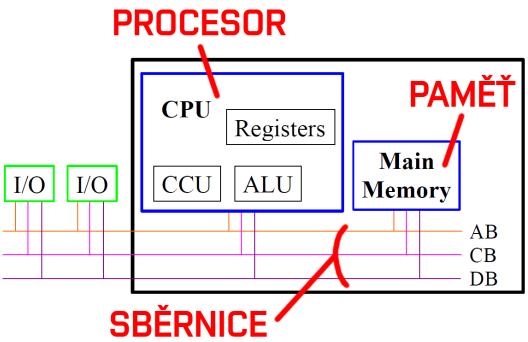
- Sběrnice **NEMŮŽE** adresovat více proměnných současně.
- Každá instrukce tak může obsahovat maximálně jedny hranaté závorky.

Například:

- Výpočet rovnice X = X + Y.

```
1 ;instrukce s 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

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

- To kterou velikost použít automaticky odvozuje překladač – podle operandů.

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

- Pokud ani jedním z operandů instrukce není registr, její velikost musíme určit ručně, jednou z pseudo-instrukcí – byte (8b), word (16b), nebo dword (32b).

```
7  section .data          ;inicializovany segment
8      X db 0             ; 8b promenna X s pocatecni hodnotou 0
9      Y dw 0             ;16b promenna Y s pocatecni hodnotou 0
10     Z dd 0             ;32b promenna Z s pocatecni hodnotou 0
11
12 section .text
13     mov byte  [X], 10 ;pouzij  8b instrukci MOV
14     mov word  [Y], 20 ;pouzij 16b instrukci MOV
15     mov dword [Z], 30 ;pouzij 32b instrukci MOV
```

Na procvičení

Vyzkoušejte si:

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

Například:

- K = 10 => K = 60
  L = 20 => L = -80
  M = 30 => M = -30

Nápověda:

- Pozor na omezení sběrnice a omezení překladače.
- Pseudo-instrukci potřebujeme pouze pokud ani jeden z operandů není registr.
- Na tom před který z operandů pseudo-instrukci umístíme nezáleží.

Ne-inicializované proměnné

V neinicializovaném segmentu rezervujeme (REServe) název, velikost a počet položek:

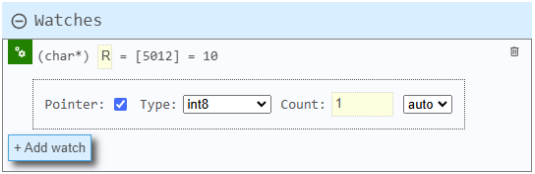
```
1  section .bss              ;ne-inicializovany datovy segment
2      R resb 1              ; (jedna) promenna jmenem R velikosti  8b
3      S resw 1              ; (jedna) promenna jmenem S velikosti 16b
4      T resd 1              ; (jedna) promenna jmenem T velikosti 32b
5      U resq 1              ; (jedna) promenna jmenem U velikosti 64b
```

Proměnné stále adresujeme pomocí jejich identifikátoru a hranatých závorek:

```
6  section .text             ;kodovy segment
7  main:
8      mov  [R], byte  10    ; do promenne R uloz  8b hodnotu 10
9      mov  [S], word  20    ; do promenne S uloz 16b hodnotu 20
10     mov  [T], dword 30    ; do promenne T uloz 32b hodnotu 30
11     ret
```

Zobrazení ne-inicializovaných proměnných v debuggeru si musíme nastavit (Watches):

- Watches -> add watch -> JMÉNO PROMĚNNÉ -> OZUBENÁ KOLEČKA -> pointer
- Nastavit musíme i velikost (Type) – int8, int16, int32, int64, a formát – auto, hex, bin.



Vyzkoušejte si:

- Vytvořte si dvě ne-inicializované 16b proměnné (X a Y).
- Ze vstupu načtěte jedno 16b číslo a jeho hodnotu uložte do proměnné X.
- Horní a spodní polovinu načteného čísla spolu vzájemně vyměňte, a jeho novou hodnotu uložte do proměnné Y.
- Hodnotu obou proměnných si zobrazte v debuggeru jako: čísla se znaménkem, v šestnáctkové soustavě a ve dvojkové soustavě.

Například:

- 23280 => X = 23280
  X = 0x5AF0
  X = 01011010 11110000
  Y = -4006
  Y = 0xF05A
  Y = 11110000 01011010

Nápověda:

- U 32b obecných registrů EAX, EBX, ECX a EDX můžeme používat celý registr, jeho spodní polovinu nebo dvě spodní čtvrtiny.

Pole

Rezervování a definice polí

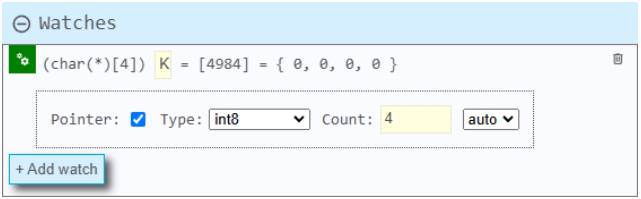
V neinicializovaném segmentu rezervujeme název, velikost a počet položek:

```
1  section .bss              ;ne-inicializovany datovy segment
2      K resb 4              ; pole ctyr  8b promennych jmenem K
3      L resw 4              ; pole ctyr 16b promennych jmenem L
4      M resd 4              ; pole ctyr 32b promennych jmenem M
5      N resq 4              ; pole ctyr 64b promennych jmenem N
```

V inicializovaném segmentu definujeme název, velikost a počáteční hodnoty:

```
6  section .data             ;inicializovany datovy segment
7      R db 10, 20, 30, 40 ; pole R s  8b hodnotami 10, 20, 30, 40
8      S dw 10, 20, 30, 40 ; pole S s 16b hodnotami 10, 20, 30, 40
9      T dd 10, 20, 30, 40 ; pole T s 32b hodnotami 10, 20, 30, 40
10     U dq 10, 20, 30, 40 ; pole U s 64b hodnotami 10, 20, 30, 40
```

Aby se pole v debuggeru zobrazilo správně musíme nastavit počet položek (Count):



Adresování polí

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

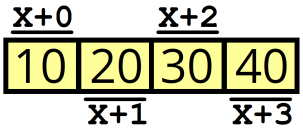
- Adresa první položky je shodná s adresou pole – její posuv je vždy nula.
- Velikost posuvu vždy udáváme v **BAJTECH** (1, 2, 4), ne v počtu položek!

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

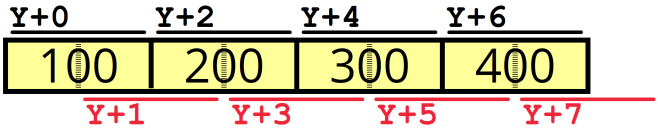
Program vypíše obsah položek pole 16b hodnot Y, s chybnými posuvy:

```
%include 'rw32.inc'           ;knihovna pro vstup a vystup
section .data                 ;inicializovany segment
    Y dw 100, 200, 300, 400 ; pole ctyr 16b hodnot 100, 200, 300, 400
section .text                 ;kodovy segment
main:
    mov  ax, [Y + 0]          ; AX = prvni polozka Y
    call WriteInt16NewLine    ; vypis AX (   100)
    mov  ax, [Y + 1]          ; AX = polovina prvni a druhe polozky Y
    call WriteInt16NewLine    ; vypis AX (-14336)
    mov  ax, [Y + 2]          ; AX = druha polozka Y
    call WriteInt16NewLine    ; vypis AX (   200)
    mov  ax, [Y + 3]          ; AX = polovina druhe a treti polozky Y
    call WriteInt16NewLine    ; vypis AX ( 11264)
    ret
```

```
X db 10, 20, 30, 40
```




```
Y dw 100, 200, 300, 400
```



Na procvičení

Vyzkoušejte si:

- Vytvořte si ne-inicializované pole tří 32b čísel (Y).
- Vytvořte si inicializované pole čtyř 32b čísel (X = 10, 20, 30, 40).
- Do položek pole Y zapište:
  Součet prvních dvou položek pole X.
  Součet posledních dvou položek pole X.
  Součet všech čtyř položek pole X.
- Obsah pole Y si zobrazte v debuggeru.

Například:

- X = 10, 20, 30, 40 => Y = 30, 70, 100

Nápověda:

- Velikost posuvu se vždy udává v počtu bajtů, ne v počtu položek.

Indexové registry

Znaky a řetězce

Obsahu pole hodnot které mají vzájemnou návaznost se říká řetězová data:

- Nejjednodušším typem řetězových dat je textový řetězec (string).
- Znaky (char) jsou 8b hodnoty, a do paměti je můžeme zadat pomocí uvozovek (`' '` nebo `" "`), nebo jako čísla interpretovaná dle tabulky ASCII.

```
1 section .data                 ;inicializovany segment
2     X db 65, 66, 67, 68       ; pole ctyr 8b hodnot 65, 66, 67, 68 jmenem X
3     Y db "ABCD"               ; pole ctyr 8b hodnot 65, 66, 67, 68 jmenem Y
```

Pro načítání a vypisování jednotlivých znaků používáme funkce ReadChar a WriteChar:

```
4     call ReadCharNewLine  ; do AL nacti znak (jeho hodnotu v ASCII)
5     call WriteCharNewLine ; obsah registru AL vypis jako znak
```

Textový řetězec je posloupnost 8b znaků zakončená znakem s hodnotu 0 (NUL):

- Speciální řídící znaky vždy zadáváme číselnou hodnotou v tabulce ASCII (zadávání pomocí zpětného lomítka `'\n'` (10) nebo `'\0'` (0) nefunguje).

```
6 section .data                     ;posloupnost znaku nasledovana znakem
7     arr db "Hello World!", 10, 0 ;konce radku (10) a konce retezce (0)
```

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

- ESI (Extended Source Index) – ze které adresy v paměti budeme číst.
- EDI (Extended Destination Index) – na kterou adresu budeme zapisovat.
- Protože do ESI a EDI dáváme adresy, tak **NEPOUŽIJEME** hranaté závorky.

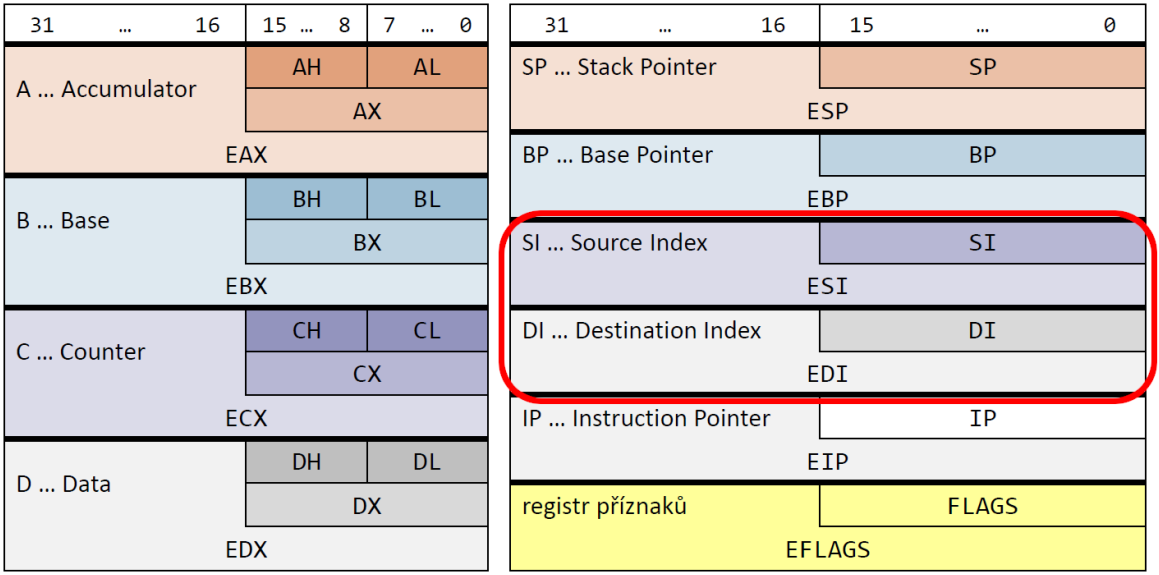
```
1     mov  esi,  arr          ; do ESI uloz adresu ARR (retezec)
2     mov  esi, [arr]         ; do ESI uloz hodnotu z adresy ARR
3                             ; (prvni ctyri znaky z retezce) - CHYBA
```

Řetězec vypíšeme funkcí WriteString, nebo WriteStringASCIIZ, z knihovny rw32.inc:

- WriteString z adresy nastavené v registru ESI vypíše ECX znaků.
- WriteStringASCIIZ z adresy v ESI vypíše řetězec ukončený znakem NUL.

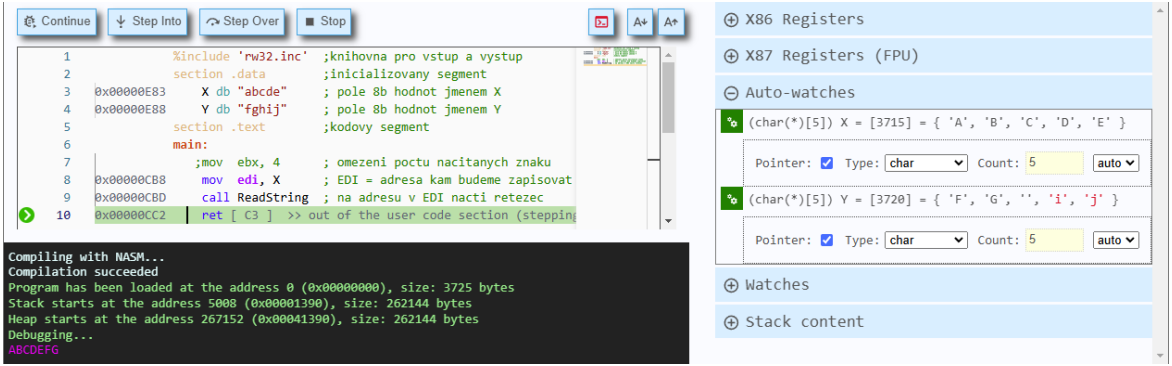
```
4  %include 'rw32.inc'          ;knihovna pro vstup a vystup
5  section .data                ;inicializovany segment
6      arr db "Hello, World!", 10, 0   ; pole 8b hodnot jmenem ARR
7  section .text                ;kodovy segment
8  main:
9      mov  esi, arr            ; ESI = adresa vypisovaneho retezce
10     call WriteStringASCIIZ   ; z ESI vypis retezec ukonceny znakem NUL
11     ret
```

Registry procesoru - indexové registry



Řetězec načteme funkcí ReadString z knihovny rw32.inc:

- Funkce ze vstupu načte EBX znaků, uloží je na adresu nastavenou v registru EDI, řetězec zakončí znakem NUL, a počet úspěšně načtených znaků nastaví do EAX.
- Pokud počet znaků neomezíme, může dojít k neplatnému přístupu do paměti.



Práci s ostatními typy řetězových dat (vektory čísel, soubory) si probereme později, společně s řetězovými instrukcemi (viz. cv07):

- Mimochodem, pro proměnnou nelze použít identifikátor str, protože by kolidoval s instrukcí STR (Store Task Register), kterou se v tomto předmětu nebudeme učit.

Program do pole arr ze vstupu načte řetězec až devíti znaků, a vypíše ho na výstup:

- Funkce ReadString, WriteString a WriteStringASCIIZ existují i ve variantě NewLine.
- Obsah pole arr si v debuggeru zobrazte jako typ string nebo char.

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

section .bss                  ;ne-inicializovany segment
    arr resb 10               ; rezervujeme misto pro 9 znaku a NUL

section .text                 ;kodovy segment
main:
    mov  edi, arr             ; EDI = adresa ciloveho pole (kam nacitame)
    mov  ebx, 9               ; EBX = maximalni pocet nacitanych znaku
    call ReadStringNewLine    ; nacti retezec a vypis konec radku

    mov  esi, arr             ; ESI = adresa zdrojoveho pole
    mov  ecx, eax             ; ECX = EAX (pocet uspesne nactenych znaku)
    call WriteStringNewLine   ; vypis ECX znaku a konec radku

   ;mov  esi, arr                  ; ESI = adresa zdrojoveho pole
   ;call WriteStringASCIIZNewLine  ; vypis retezec a konec radku
    ret
```

Vyzkoušejte si:

- Vytvořte si ne-inicializované pole, a načtěte do něj řetězec pěti znaků.
- Předpokládejte že uživatel opravdu zadal řetězec pěti znaků (podmínku jak zkontrolovat hodnotu registru EAX se budeme učit později – viz. cv05).
- Znaky řetězce přeskládejte do opačného pořadí, a vypište ho na výstup.

Například:

- ABCDE => EDCBA

Nápověda:

- Znaky jsou 8b hodnoty.
- Při alokaci nezapomeňte na místo pro ukončující znak NUL.
- Pozor na omezení sběrnice a omezení překladače.