Registre de segmente

(https://www.tutorialspoint.com/compile_assembly_online.php)

Segmentele sunt zone specifice definite într-un program pentru a conține date, cod și stivă. Există trei segmente principale −

- **Segment de cod** − Conține toate instrucțiunile de executat. Un registru de segment de cod de 16 biți sau un registru CS stochează adresa de pornire a segmentului de cod.
- **Segment de date** - Conține date, constante și zone de lucru. Un registru de segment de date pe 16 biți sau un registru DS stochează adresa de pornire a segmentului de date.
- **Segment de stivă** - Conține date și adrese de returnare ale procedurilor sau subrutinelor. Este implementat ca o structură de date „stivă”. Registrul Segmentului de stivă sau registrul SS stochează adresa de pornire a stivei.

```
section  .text
   global _start   ;must be declared for linker (gcc)

_start:            ;tell linker entry point
   mov   edx,len   ;message length
   mov   ecx,msg   ;message to write
   mov   ebx,1     ;file descriptor (stdout)
   mov   eax,4     ;system call number (sys_write)
   int   0x80      ;call kernel

   mov   edx,9     ;message length
   mov   ecx,s2    ;message to write
   mov   ebx,1     ;file descriptor (stdout)
   mov   eax,4     ;system call number (sys_write)
   int   0x80      ;call kernel
```

```
    mov   eax,1     ;system call number (sys_exit)
    int   0x80      ;call kernel

section  .data
msg db 'Buna dimineata la ora 9:',0xa ;a message
len equ $ - msg  ;length of message
s2 times 9 db '*'
```

Exercitiu: Sa se citeasca un numar de la tastatura si sa se afiseze pe ecran:

```
section .data                ;Data segment
  userMsg db 'Please enter a number: ' ;Ask the user to enter a number
  lenUserMsg equ $-userMsg          ;The length of the message
  dispMsg db 'You have entered: '
  lenDispMsg equ $-dispMsg

section .bss        ;Uninitialized data
  num resb 5

section .text       ;Code Segment
  global _start

_start:             ;User prompt
  mov eax, 4
  mov ebx, 1
  mov ecx, userMsg
  mov edx, lenUserMsg
  int 80h
```

```
;Read and store the user input
mov eax, 3
mov ebx, 2
mov ecx, num
mov edx, 5       ;5 bytes (numeric, 1 for sign) of that information
int 80h

;Output the message 'The entered number is: '
mov eax, 4
mov ebx, 1
mov ecx, dispMsg
mov edx, lenDispMsg
int 80h

;Output the number entered
mov eax, 4
mov ebx, 1
mov ecx, num
mov edx, 5
int 80h

; Exit code
mov eax, 1
mov ebx, 0
int 80h
```

int 0x80 este o instrucțiune în limbajul de asamblare utilizată pe arhitectura x86 pentru a efectua un apel de sistem (syscall) către nucleul (kernel) sistemului de operare Linux. Această instrucțiune declanșează o întrerupere software care comută executarea de la modul utilizator la modul kernel, permițând astfel programelor să solicite servicii de la sistemul de operare, cum ar fi citirea sau scrierea într-un fișier, alocarea memoriei sau terminarea unui proces.

Directive definite

DB	Definiți octet	alocă 1 octet
DW	Definiți cuvântul	alocă 2 octeți
DD	Definiți Doubleword	alocă 4 octeți
DQ	Definiți Quadword	alocă 8 octeți
DT	Definiți zece octeți	alocă 10 octeți

Exemple de utilizare a directivelor:

```
choice          DB      'y'
number          DW      12345
neg_number      DW      -12345
big_number      DQ      123456789
real_number1    DD      1.234
real_number2    DQ      123.456
```

Exemplu de program folosind directive DB:

```
section .text
   global _start         ;must be declared for linker (gcc)


_start:                  ;tell linker entry point
   mov  edx,1               ;message length
   mov  ecx,choice          ;message to write
   mov  ebx,1               ;file descriptor (stdout)
   mov  eax,4               ;system call number (sys_write)
   int     0x80             ;call kernel


   mov  eax,1               ;system call number (sys_exit)
   int     0x80             ;call kernel
```

```
section .data
choice DB 'y'
```

Directiva TIMES

Directiva TIMES permite mai multe inițializari la aceeași valoare. Este utilă în definirea matricelor și tabelelor. Următorul program afișează 9 asteriscuri pe ecran.

```
section .text
   global _start       ;must be declared for linker (ld)

_start:                ;tell linker entry point
   mov  edx,9          ;message length
   mov  ecx, stars     ;message to write
   mov  ebx,1          ;file descriptor (stdout)
   mov  eax,4          ;system call number (sys_write)
   int     0x80        ;call kernel

   mov  eax,1          ;system call number (sys_exit)
   int     0x80        ;call kernel

section .data
stars   times 9 db '*'
```

Directiva EQU

Directiva **EQU** este utilizată pentru definirea constantelor. Sintaxa directivei EQU este următoarea −

```
CONSTANT_NAME EQU expression
```

De exemplu,

```
TOTAL_STUDENTS equ 50
```

Următorul exemplu ilustrează utilizarea directivei EQU

```
SYS_EXIT  equ 1
SYS_WRITE equ 4
STDIN     equ 0
STDOUT    equ 1
section  .text
   global _start    ;must be declared for using gcc

_start:            ;tell linker entry point
   mov eax, SYS_WRITE
   mov ebx, STDOUT
   mov ecx, msg1
   mov edx, len1
   int 0x80

   mov eax, SYS_WRITE
   mov ebx, STDOUT
   mov ecx, msg2
   mov edx, len2
   int 0x80

   mov eax, SYS_WRITE
   mov ebx, STDOUT
   mov ecx, msg3
   mov edx, len3
   int 0x80

   mov eax,SYS_EXIT   ;system call number (sys_exit)
   int 0x80           ;call kernel

section  .data
```

```
msg1 db         'Python!',0xA,0xD
len1 equ $ - msg1

msg2 db 'Instrumentatie,', 0xA,0xD
len2 equ $ - msg2

msg3 db 'Arhitectura! '
len3 equ $- msg3
```

Exercitii:

Corectati codul de mai jos astfel incat sa se execute cu success, iar la output sa se afiseze "Exercitii Assembler"

```
section .text
   global _start     ;must be declared for linker (ld)

_main:            ;tells linker entry point
   mov  edx,len     ;message length
   mov  ecx,msg     ;message to write
   mov  ebx,1       ;file descriptor (stdout)
   mov  eax,4       ;system call number (sys_write)
   int     0x40     ;call kernel

   mov  eax,1       ;system call number (sys_exit)
   int     0x40     ;call kernel

section .data
msg db '', 0xa  ;string to be printed
len equ $ - msg     ;length of the string
```

Scrieți un program în limbaj de asamblare care afișează mesajul "Hello, World!" pe ecran folosind syscall-ul **sys_write** într-un sistem Linux pe 32 de biți.

Implementați un program în limbaj de asamblare care calculează suma a două numere întregi citite de la tastatură și afișează rezultatul. Utilizați syscall-urile **sys_read** și **sys_write** pentru interacțiunea cu utilizatorul.

Scrieți un program în limbaj de asamblare care primește un șir de caractere de la utilizator, inversează șirul și îl afișează înapoi utilizatorului. Considerați șirul terminat cu un caracter newline (**\n**).

Dezvoltați un program în limbaj de asamblare care calculează factorialul unui număr întreg nenegativ dat și afișează rezultatul. Numărul de intrare trebuie să fie citit de la tastatură și rezultatul afișat pe ecran.

Implementați un program în limbaj de asamblare care convertește un număr din sistemul binar în sistemul decimal și îl afișează. Numărul binar va fi citit de la tastatură ca un șir de caractere **0** și **1**, iar rezultatul va fi afișat ca un număr întreg decimal.