LAB 3 – Registre & instructiuni

Nume	Rol
EAX	acumulator; apeluri de sistem, I/O, aritmetică
EBX	registru de bază; folosit pentru adresarea bazată a memoriei
ECX	contor în cadrul instrucțiunilor de buclare
EDX	registru de date; I/O, aritmetică, valori de întrerupere; poate extinde EAX la 64 de biți
ESI	sursă în cadrul operațiilor pe stringuri
EDI	destinație în cadrul operațiilor pe stringuri
EBP	base sau frame pointer; indică spre cadrul curent al stivei
ESP	stack pointer; indică spre vârful stivei

Nume	Nume extins	Descriere
CF	Carry Flag	Setat dacă rezultatul depășește valoarea întreagă maximă (sau minimă) reprezentabilă pe numere **unsigned**
PF	Parity Flag	Setat dacă byte-ul low al rezultatului conține un număr par de biți de 1
AF	Auxiliary Carry Flag	Folosit în aritmetică BCD; setat dacă bitul 3 generează carry sau borrow
ZF	Zero Flag	Setat dacă rezultatul instrucțiunii precedente este 0
SF	Sign Flag	Are aceeași valoare cu a bitului de semn din cadrul rezultatului (1 negativ, 0 pozitiv)
OF	Overflow Flag	Setat dacă rezultatul depășește valoarea întreagă maximă (sau minimă) reprezentabilă pe numere **signed**

Nume	Operanzi	Descriere
add	dst, src	Adună sursa cu destinația; rezultatul se scrie la destinație
sub	dst, src	Se scade din destinație sursa și se reține în destinație rezultatul
and	dst, src	Se efectuează operația de ȘI logic între sursă și destinație și se reține rezultatul în destinație
test	dst, src	Se efectuează operația de ȘI logic între sursă și destinație fără a se reține rezultatul undeva
shl	dst, <const>	Se face shiftare logică la stânga a destinației cu un număr constant de poziții

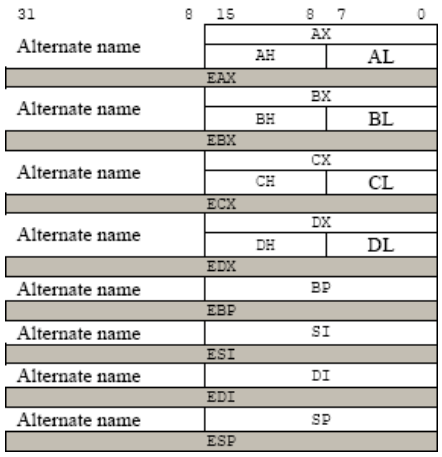
Nume	Operanzi	Descriere
jmp	<adresă>	Efectuează salt necondiționat la adresa indicată (direct, prin registru, prin etichete)
cmp	dst, src	Compară sursa cu destinația (detalii mai jos)
j*condiție*	<adresă>	Efectuează salt condiționat, în funcție de valoarea flagului/variabilei de condiție
call	<adresă>	Face apel la subrutina care se găsește la adresa indicată

Nume	Operanzi	Descriere
mov	dst, src	Mută valoarea din sursă peste destinație
push	src	Mută valoarea din sursă în vârful stivei
pop	dst	Mută valoarea din vârful stivei în destinație
lea	dst, src	Încarcă adresa efectivă a sursei în destinație
xchg	dst, src	Interschimbă valorile din sursă și destinație

Ex.: - hello_world.asm -> printeaza mesaj cu macro-uri
- grumpy-jumps.asm
- sets.asm -> multimi de numere tinute pe biti, operatii logice
- min.asm -> instructiunea xchg
- fibonacci.asm -> implementat cu xchg

LAB 4 – Adresare directa si bazata

Mnemonică	Descriere
LOOPE/LOOPZ label	Decrementează ECX; sari la eticheta label dacă ECX != 0 și ZF == 1
LOOPNE/LOOPNZ label	Decrementează ECX; sari la eticheta label dacă ECX != 0 și ZF != 1




```
.DATA
    var     DB 64    ; Declară un octet conținând valoarea 64. Etichetează
                     ; locația de memorie cu "var".
    var2    DB ?     ; Declară un octet neinițializat etichetat cu "var2".
            DB 10    ; Declară un octet neetichetat, inițializat cu 10. Acest
                     ; octet va fi plasat la adresa (var2 + 1).
    X       DW ?     ; Declară un cuvânt(2 octeți) neinițializat, etichetat cu "X".
    Y       DD 3000  ; Declară un cuvânt dublu (4 octeți) cu eticheta "Y",
                     ; inițializat cu valoarea 3000.
    Z       DD 1,2,3 ; Declară 3 cuvinte duble (a câte 4 octeți fiecare)
                     ; incepând cu adresa "Z" și inițializate cu 1, 2, respectiv 3.
                     ; De exemplu, 3 va fi plasat la adresa (Z + 8).
```

```
mov eax, [0xcafebab3]         ; directă (deplasament)
mov eax, [esi]                ; indirectă (bază)
mov eax, [ebp-8]              ; bazată (bază + deplasament)
mov eax, [ebx*4 + 0xdeadbeef] ; indexată (index * scală + deplasament)
mov eax, [edx + ebx + 12]     ; bazată și indexată fără scală (bază + index + deplasament)
mov eax, [edx + ebx*4 + 42]   ; bazată și indexată cu scală (bază + index * scală + deplasame
```

Ex.: - 0-recap/power-2.asm -> descompunere in puteri ale lui 2
- **1-2-multiply/multiply.asm -> inmultirea a doua numere pe 1/2/4 octeti**
- 3-4-sum/sum_n.asm si 3-4-sum/sum_n_square.asm
- 5-6-7-sum-array/ -> suma elementelor si a patratelor elementelor unui vector
- **8-divide/divide.asm -> impartirea a doua numere pe 1/2/4 octeti**
- bonus/ -> numarul de elemente pare/pozitive dintr-un vector

LAB 5 – Structuri, vectori, siruri

- `rep` - repeat while (ecx != 0)
- `repe/repz` - repeat while (»equal« and ecx !=0) (ex: repetă până găsim un element diferit în vector)
- `repne/repnz` - repeat while (»not equal« and ecx !=0) (ex: repetă până găsim un element comun în vector)

Declarare structura:

```
struc mystruct
    a:    resw 1    ; a va referi un singur element de dimensiune un cuvânt
    b:    resd 1    ; b va referi un singur element de dimensiune un dublu cuvânt
    c:    resb 1    ; c va referi un singur element de dimensiune un octet
    d:    resd 1    ; d va referi un singur element de dimensiune un dublu cuvânt
    e:    resb 6    ; e va referi 6 elemente de dimensiune un octet
endstruc
```

Initializare campuri structura:

```
struct_var:
    istruc mystruct
        at a, dw        -1
        at b, dd        0x12345678
        at c, db        ' '
        at d, dd        23
        at e, db        'Gary', 0
    iend
```

MOVS
CMPS
SCAS
LODS
STOS

De exemplu, pentru a declara un vector de 100 de cuvinte inițializate cu valoarea 42, vom folosi construcția:

```
section .data
    myVect:    times 100    dw 42
```

Pe de altă parte, dacă dorim declararea unui vector de 20 de elemente dublu cuvinte neinițializate, folosim instrucțiuni din familia `res` astfel:

Declararea unui vector:

```
section .bss
    myVect:    resd 20
```

Ex.: **- memset.asm -> stosb**
- 2-3-strings**/**strings.asm **-> strlen (**folosind repne scasb) si numarul de aparitii ale unui caracter
- 4-5-print-structure/ -> modificarea si afisarea unei structuri
- 6-process-structure/ -> prelucrarea unei structuri
- bonus/find2.asm -> cautarea unui subsir intr-un sir

LAB 6 – Stiva

Ex.: - reverse-array.asm -> inverseaza vector folosind push si pop
- 4-local-var/merge-arrays.asm -> merge sort pe 2 vectori sortati crescator
- 5-gcd/gcd.asm -> cel mai mare divizor comun a doua numere

LAB 7 – Functii

Ex.: - print-string.asm -> apel functia puts
- print-string-len.asm -> apel functia printf

LAB 8 – C + Assembly

Ex.:` - 1-inline-for/ & 2-inline-rotate/ -> **inline assembly**
- 3-inline-cpuid/ -> **inline assembly & informatii despre procesor**
- 4-5-max-c-calls/ -> program C cu apel de functie assembly
- 6-7-max-assembly-calls/ -> program assembly cu apel de functie C
- bonus/ -> assembly pe 64 biti

LAB 9 – Analiza programelor

- IDA
- GDB
- objdump: objdump -M intel -d hello-world.o
- nm

LAB 10 – Buffer overflow

- fgets, puts, gets, ...

LAB 11 - Optimizari

LAB 12 – Virgula mobila

Instrucțiuni de tip pop

Instrucțiune	Descriere
fstp DWORD [registru]	Citește o valoare de tip float (4 octeți) din vârful stivei și o salvează la adresa indicată de registru. Valoarea este eliminată de pe stivă.
fst DWORD [registru]	Similar cu instrucțiunea anterioară, dar valoarea rămâne în vârful stivei.
fstp QWORD [registru]	Citește o valoare de tip double (8 octeți) din vârful stivei și o salvează la adresa indicată de registru, eliminând valoarea de pe stivă
fst QWORD [registru]	Similar cu instrucțiunea anterioară, dar valoare rămâne în vârful stivei.

Instrucțiuni de tip push

Instrucțiune	Descriere
fld1	Stochează constanta 1 în vârful stivei
fldz	Stochează constanta 0 în vârful stivei
fldpi	Stochează constanta π în vârful stivei
fld DWORD [registru]	Stochează valoarea de tip float (4 octeți) de la adresa indicată de registru
fild DWORD [registru]	Stochează valoarea de tip integer (4 octeți) de la adresa indicată de registru
fld QWORD [registru]	Stochează valoarea de tip double (8 octeți) de la adresa indicată de registru
fld st0	Duplică valoarea din vârful stivei
fxch	Interschimbă primele două valori de pe stivă .

Instrucțiuni matematice

Instrucțiune	Descriere
fabs	Înlocuiește prima valoare de pe stivă cu valoarea ei absolută.
fchs	Change sign - schimbă semnul valorii din vârful stivei.
frndint	Round to integer - rotunjește prima valoare de pe stivă la întreg.
fadd dword/qword [registru]	Adună prima valoare de pe stivă cu cea indicată de adresa din registru (pe 4/8 octeți).
fdiv dword/qword [registru]	Împarte prima valoare de pe stivă cu cea indicată de adresa din registru (pe 4/8 octeți)
fdivr dword/qword [registru]	Similar cu instrucțiunea precedentă, dar împărțirea este inversă.
fmul dword/qword [registru]	Înmulțește prima valoare de pe stivă cu cea indicată de adresa din registru (pe 4/8 octeți).
fsub dword/qword [registru]	Scade din prima valoare de pe stivă valoarea indicată de adresa din registru (pe 4/8 octeți).
fsubr dword/qword [registru]	Similar cu instrucțiunea anterioară, dar ordinea operanzilor se schimbă
fsqrt	Înlocuiește prima valoare de pe stivă cu rădăcina ei pătrată
fsin	Înlocuiește prima valoare de pe stivă cu rezultatul funcției sin.
fcos	Înlocuiește prima valoare de pe stivă cu rezultatul funcției cos.

Instrucțiunile `fiadd`, `fisub`, `fidiv`, `fimul` funcționează exact ca cele din tabelul de mai sus, dar primesc ca argument o valoare întreagă de 2/4 octeți (word/dword).

Instrucțiuni de comparație

Instrucțiune	Descriere
fcom	Compară primele două valori de pe stivă și setează flag-urile interne FPU.
fcomi	Similar cu prima instrucțiune, dar setează flag-urile ZF, PF și CF din registrul EFLAGS.
fcomip	Similar cu prima instrucțiune, dar setează flag-urile și elimină prima valoare din vârful stivei.
ficom word [registru]	Compara prima valoare de pe stivă cu un număr întreg pe 2 octeți de la adresa indicată de registru.
ficom dword [registru]	Compară prima valoare de pe stivă cu un număr întreg pe 4 octeți de la adresa indicată de registru.
ficom qword [registru]	Compară prima valoare de pe stivă cu un număr întreg pe 8 octeți de la adresa indicată de registru.
ficomp word/dword/qword [registru]	Similar cu instrucțiunile anterioare, dar elimină și prima valoare de pe stivă.
ftst	Compară prima valoare de pe stivă cu 0.0.

Ex.:
- suma.asm -> suma unui vector de numere fractionare
- media.asm -> media unui vector de numere fractionare
- integer-div.asm -> impartirea a 2 nr intregi cu rezultat fractionar
- extract.asm -> extragere parte intreaga si fractionara
- media-int.asm -> media unui vector de numere intregi
- max.asm -> maximul dintr-un vector de numere fractionare
- arcsin.asm