Laborator 8

* regiştri (inclusiv indicatorii de conditii)
* introducere în Visual Studio
* instrucțiuni simple (mov, add, sub, inc, dec)
* instrucțiuni logice: not, or, and, xor, test, shl, shr, sar ror, rol, înmulțire, împărțire

Octetul (**BYTE**): bitul cu indexul 7 este cel mai semnificativ, cel cu indexul 0 este cel mai putin semnificativ. Folosit pentru reprezentarea numerelor intregi cu sau fara semn (char/unsigned char)

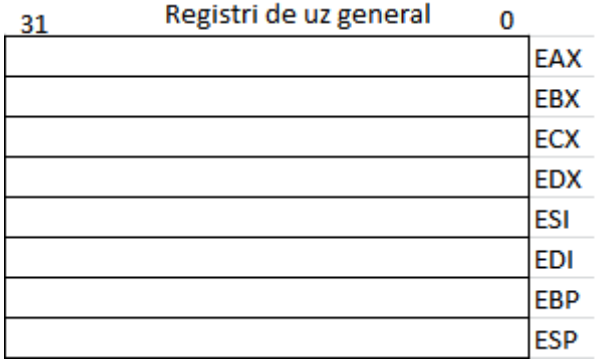
Cuvantul de dimensiune 2 (**WORD**): octetul 15-8 este cel superior; octetul 7-0 este cel inferior. Folosit pentru reprezentarea numerelor intregi cu sau fara semn (short/unsigned short)

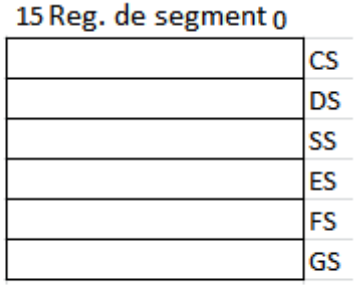
Cuvantul de dimensiune 4: (**DWORD**): octetul 31-24 este cel superior; octetul 7-0 este cel inferior. Folosit pentru reprezentarea numerelor intregi cu sau fara semn (int/unsigned int), a adreselor de memorie (pointeri)

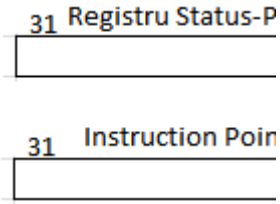
31 24	23 16	15 8	7 .....

Bit and Byte Order. Procesoarele IA-32 sunt masini de tipul little-endian: octetii dintr-un WORD/DWORD sunt pusi dinspre adresa mai mare de memorie catre adresa mai mica de memorie.

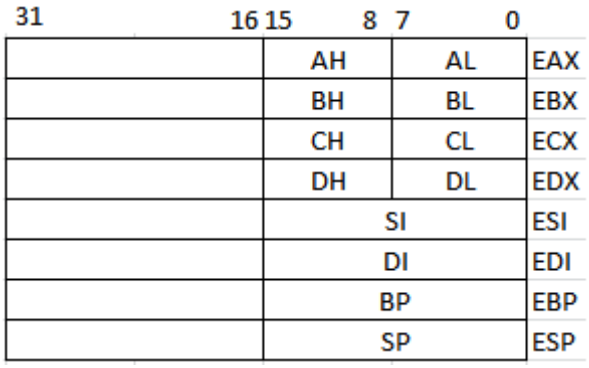
Tipuri de registri:
Registri de uz general. Acesti 8 registri sunt folositi pentru stocarea operanzilor si a pointerilor.
Registri de segment. Acesti registri stocheaza 6 selectori de segment.
EFLAGS(Status-Program si Control). Acest registru retine statusul programului in curs de executie si permite controlul limitat al procesorului.
EIP(intruction Pointer). Contine un pointer pe 32 de biti catre urmatoarea instructiune de executat.







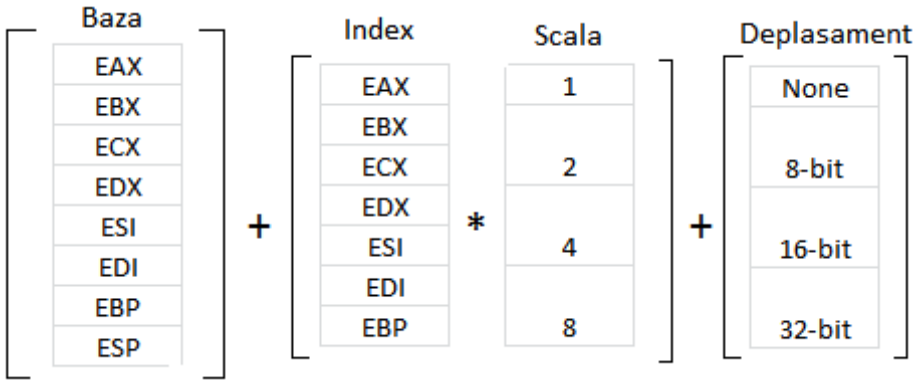
Cei mai putin semnificativi 16 biti din cadrul registrilor de uz general sunt asociati direct cu registrii generali din cadrul procesoarelor 8086 si Intel 286, si pot fi referentiati prin registrii: AX, BX, CX, DX, BP, SP, SI, DI, SP. Fiecare din cei mai putin semnificativi doi octeti din cadrul registrilor EAX, EBX,ECX,EDX pot fi accesati prin numele: AH, BH, CH, DH (cei mai semnificativi) si respectiv AL, BL, CL, DL (cei mai putin semnificativi)



Registrul pe 32 de biti EFLAGS contine mai multi marcatori de tip "Status Flag":
* **CF** (bit 0): Carry Flag este setat doar atunci cand o operatie aritmetica genereaza un transport sau se foloseste de un imprumut pe cel mai semnificativ bit. Se poate folosi la testarea depasirii in cadrul aritmeticii cu numere intregi fara semn.
* **PF** (bitul 2): Parity Flag este setat doar atunci cand cel mai putin semnificativ octet din cadrul rezultatului are un numar par de biti de 1.
* **ZF** (bitul 6): Zero flag este setat doar atunci cand rezultatul este zero.
* **SF** (bitul 7): Sign flag are valoarea celui mai semnificativ bit din cadrul rezultatului, adica este semnul al unui numar intreg cu semn (0=pozitiv,1=negativ).
* **OF** (bitul 11): Overflow flag este setat doar atunci cand apare o eroare de depasire in cadrul aritmeticii cu numere intregi cu semn.

Modul de adresare a operanzilor.
Instructiunile masina IA-32 se bazeaza pe zero sau mai multi operanzi. Unii operanzi sunt specificati in mod explicit, altii in mod implicit.
Datele pentru un operand de tip sursa pot fi localizate intr-un registru, intr-o locatie de memorie sau in cadrul instructiunii respective (un operand direct ~ immediate operand).
Datele pentru un operand de tip destinatie pot fi localizate intr-un registru sau intr-o locatie de memorie.

Adresarea memoriei.
Offsetul din cadrul unei adrese de memorie poate fi specificat direct ca o valoare statica (deplasament) sau prin intermediul unui calcul ce poate contine cel putin una din urmatoarele componente: **desplasament** (8/16/32bit)**, baza** (valoarea din cadrului unui registru general), **index** (valoarea din cadrului unui registru general), **factor de scalare** (valoarea 2, 4, sau 8). Offsetul care rezulta din aceste elemente se numeste adresa efectiva. Offset = Baza+ (Index*Factor) + Deplasament



Instructiuni:

1. Moving data: **MOV** dest, source // dest <- source.
2. Exchange: **XCHG** dest, source // temp <- dest; dest <- source; source <- temp.
3. Integer add: **ADD** dest, source// dest <- (dest+source).
4. Substract: **SUB** dest, source// dest <- (dest-source).
5. Increment: **INC** dest //dest <- (dest + 1).
6. Decrement: **DEC** dest //dest <- (dest - 1).
7. Negate: **NEG** dest //dest <- (-dest).
8. Bitwise logical not: **NOT** dest //dest <- C1(dest).
9. Bitwise logical and: **AND** dest, source //dest <- (dest & source).
10. Bitwise logical or: **OR** dest, source //dest <- (dest | source).
11. Bitwise logical xor: **XOR** dest, source //dest <- (dest ^ source).
12. Logical compare: **TEST** operand1, operand2 //Computes the Bitwise logical AND between the two operands and sets the SF, ZF and PF. The result is then discarded.

Shift Instructions: The last bit shifted beyond the destination boundary are shifted into the cary flag, then discarded.
The *count* operand can be the **CL** register or an immediate value.
There is no difference between SAL and SHL

13. Shift arithm. right: **SAR** signed_dest, count // signed_dest <- (signed_dest >> (count%32)).
14. Shift logic right: **SHR** unsigned_dest, count // unsigned_dest <- (unsigned_dest >> (count%32)).
15. Shift arithm. left: **SAL** dest, count // dest <- (dest << (count%32)).
16. Shift logic. left: **SHL** dest, count // dest <- (dest << (count%32)).
17. Rotate right: **ROR** dest, count// Rotate *dest* bits *count%32* times to the right.
17. Rotate left: **ROL** dest, count// Rotate *dest* bits *count%32* times to the left.
18. Rotate right including CF: **RCR** dest, count // The RCR instruction shifts the CF flag into the most-significant bit and shifts the least-significant bit into the CF flag

19. Rotate left including CF: **RCL** dest, count // The RCL instruction shifts the CF flag into the least-significant bit and shifts the most-significant bit into the CF flag

20. Unsigned multiply: **MUL** source2// destination <- source1 * source2;

Operand Size	Source 1	Source 2	Destination
Byte	AL	r/m8	AX
Word	AX	r/m16	DX:AX
Doubleword	EAX	r/m32	EDX:EAX

21. Unsigned divide **DIV** divizor// divident = quotient * divizor + remainder

Operand Size	Dividend	Divisor	Quotient	Remainder	Maxim Quoti
Word/byte	AX	r/m8	AL	AH	255
Doubleword/word	DX:AX	r/m16	AX	DX	65,535
Quadword/doubleword	EDX:EAX	r/m32	EAX	EDX	$2^{32} - 1$

22. Signed multiply:
* **IMUL** r/m (The same as **MUL** instruction)
* **IMUL** r, r/m (First operand = first operand*second operand)
* **IMUL** r, r/m, imm

Exercitii:

1.

```
#include <stdio.h>
#define LEAST_SEMNIF 55607
#define SEMNIF 1

//Sa se scrie codul in limbaj de asamblare care calculeaza suma: 1+2+3+...+n, unde n =
92682
//Atentie, aceasta suma nu se poate reprezenta folosind doar 32 de biti.

void main(){
 int n;
 n = 92682;
 int least_semnif,semnif;
 _asm{
```

```
 /*In acest bloc scrieti codul ASM*/
 }
 if( least_semnif == LEAST_SEMNIF && SEMNIF == semnif){
 printf("Ok!\n");
 }else{
 printf("Failed! Your result is: %d*pow(2,32)+%d\n",semnif, least_semnif);
 }
}
```

2

```
#include <stdio.h>

//Sa se scrie codul in limbaj de asamblare care oglindeste bitii  unui numar

void main(){
 char number;
 number = 140;

 _asm{
 /* Completati */
 }
 if( number != 49)
 printf("Failed! Your result is %d\n",number);
 else
 printf("OK!");
}
```

l8p1r.cpp (1k) Alexandru Baetu, 4 dec. 2014, 00:29 v.2