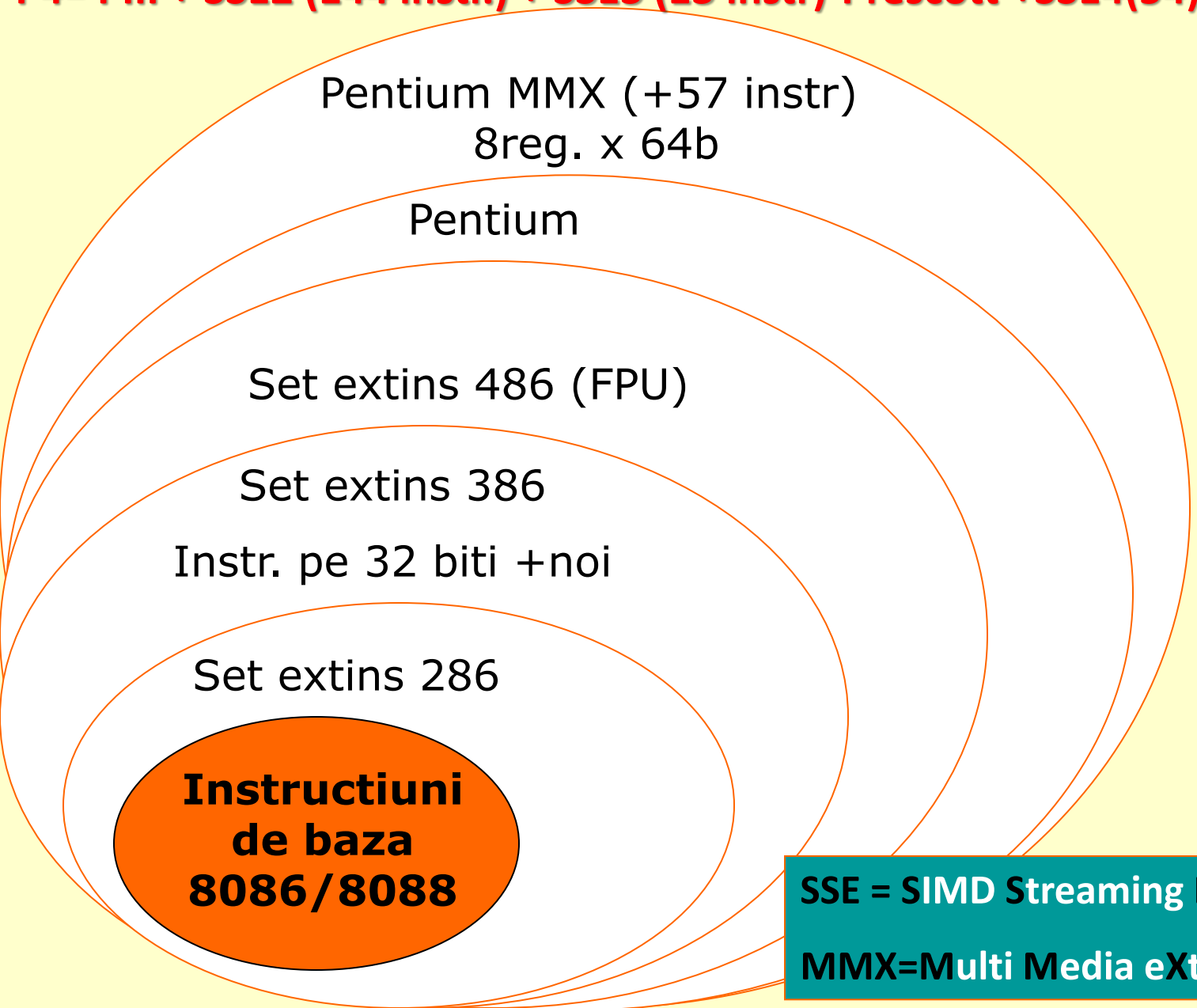
“Rabdarea este insotitoarea intelepciunii” Sf. Augustin

Curs 11

Setul extins de instructiuni x86

PIII = PII + SSE (70 instr.) – 8 reg x 128b – FP-DP

P4= PIII + SSE2 (144 instr.) + SSE3 (13 instr)-Prescott +SSE4(54) +SSE5 (170)…..



SSE = SIMD Streaming Extension

MMX=Multi Media eXtention

Setul de instructiuni extins x86 (32 biti)

Regiştri generali:

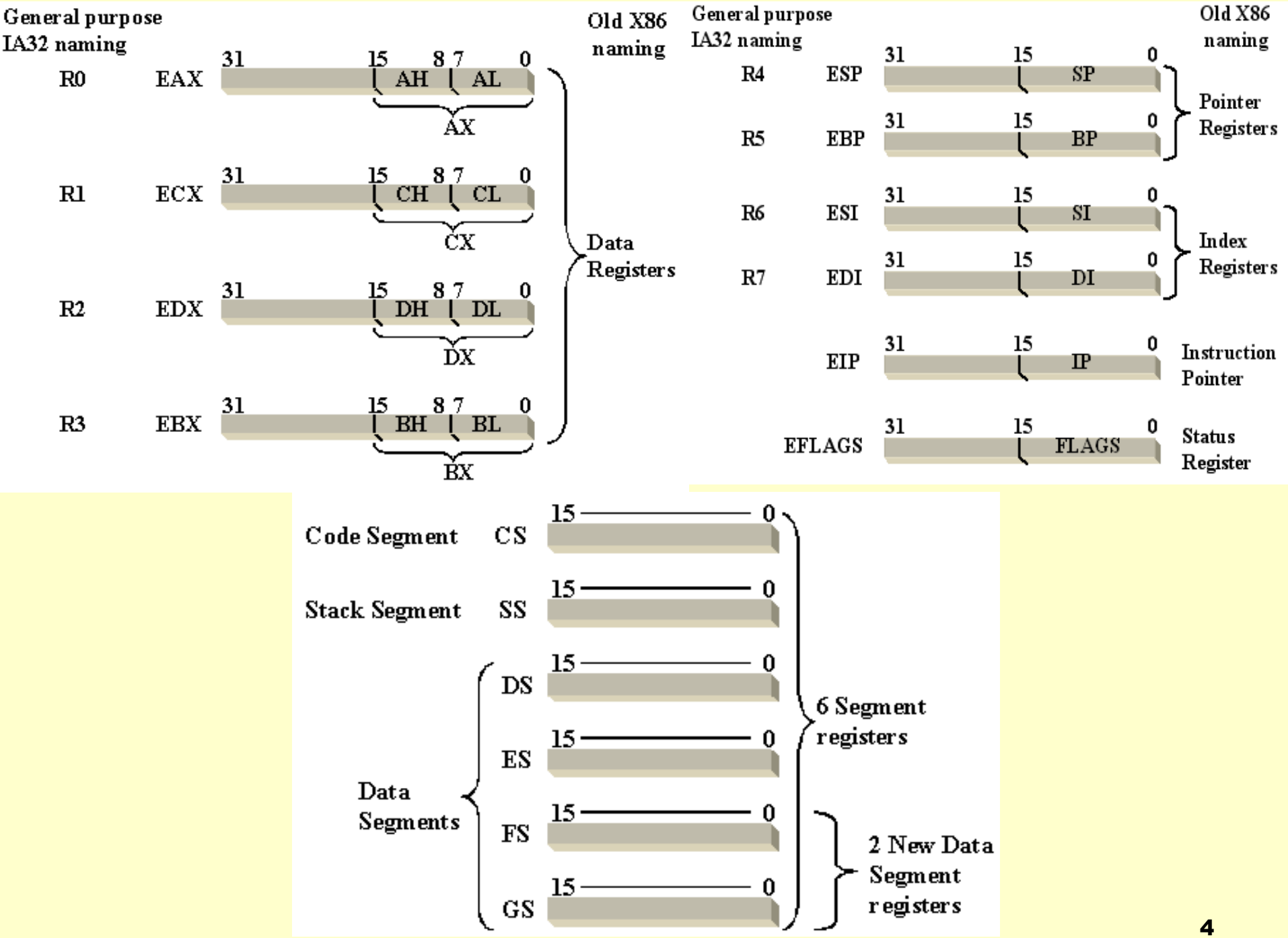
EAX — acumulatorul implicit
EBX — conţine implicit o adresă de bază ptr. anumite moduri de adresare
ECX — contor implicit
EDX — acumulator extins implicit sau registru de date
ESI — index pentru sursă
EDI — index pentru destinaţie
EBP — indicatorul bazei în stivă
ESP — indicatorul curent în stivă

Regiştri de stare şi control:

EIP — indicator (numărător) de instrucţiuni
EF — registrul de flaguri
CR0…CR4 — regiştri de control procesor
DR0…DR7 — regiştri pentru depanare (debug)

Regiştri segment:

DS, ES
FS — registru segment suplimentar (de date)
GS — registru segment suplimentar (de date)


General purpose
IA32 naming
Old X86
naming
R0
EAX
31
15
8
7
0
AH
AL
AX
R1
ECX
CH
CL
CX
R2
EDX
DH
DL
DX
R3
EBX
BH
BL
BX
Data
Registers
General purpose
IA32 naming
Old X86
naming
R4
ESP
SP
R5
EBP
BP
Pointer
Registers
R6
ESI
SI
R7
EDI
DI
Index
Registers
EIP
IP
Instruction
Pointer
EFLAGS
FLAGS
Status
Register
Code Segment
CS
Stack Segment
SS
Data
Segments
DS
ES
FS
GS
6 Segment
registers
2 New Data
Segment
registers

Setul de instructiuni extins x86

- Utilizarea setului extins de instrucţiuni în aplicaţii trebuie anunţată asamblorului folosind directivele:
- .8086 .8087 .186 .286 .287 ;
- .286P .386 .387 .386P .486;
- .486P .586 .586P .MMX .SSE

vezi Lucrarea 12

Setul de instructiuni extins

1. Instrucţiuni de transfer

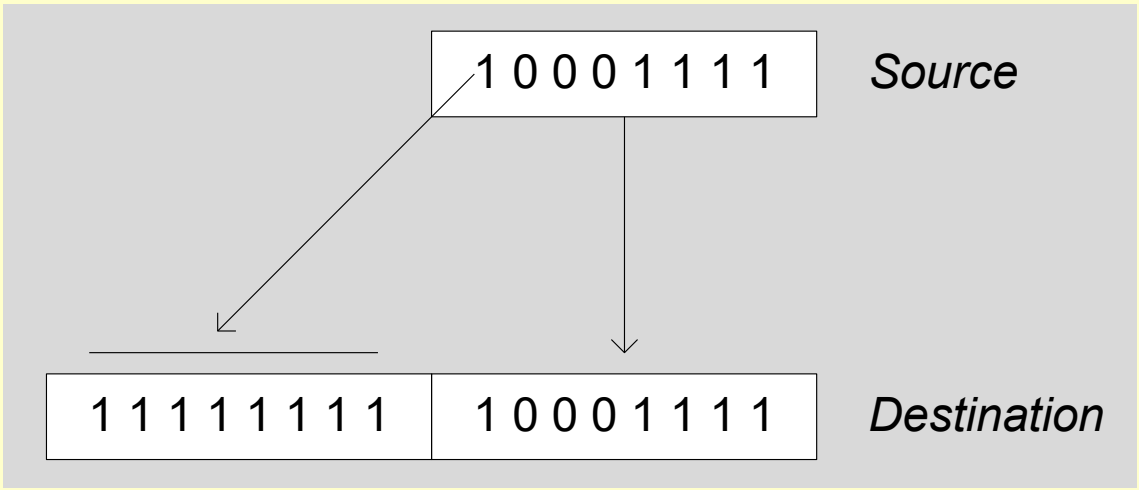
MOVSX (Move with Sign Extension) –

MOVSX destinaţie, sursă *; destinaţie ← sign_extend(sursă)*

Operanzi: reg,reg
reg,mem

Exemplu:

```
MOVSX   EAX, AL                 ;octet → dublucuvânt
MOVSX   EDI, WORD PTR [ESI]     ;cuvânt → dublucuvânt
```



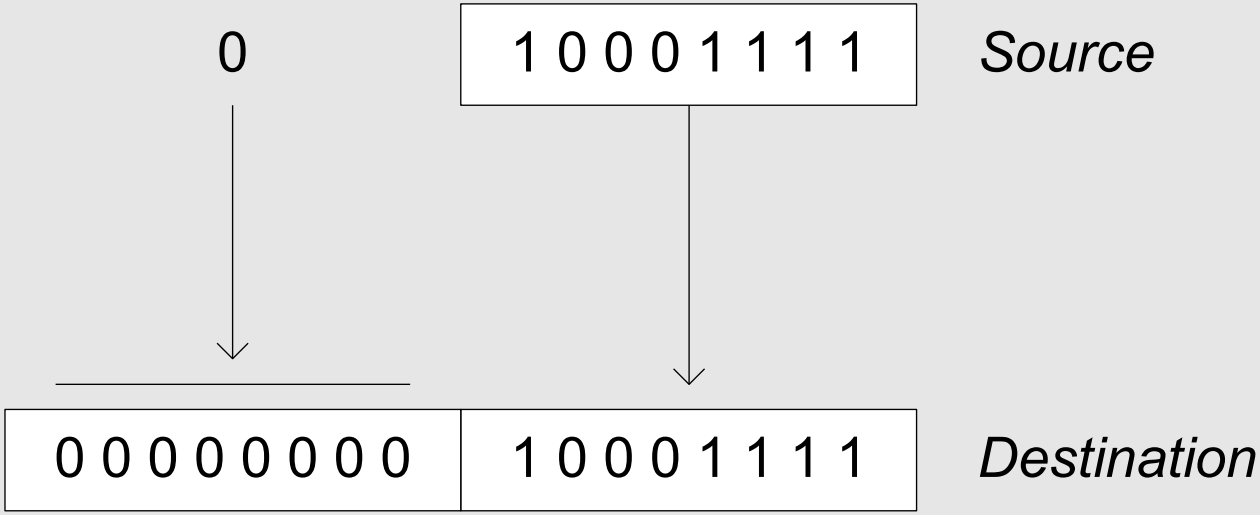
MOVZX (Move with Zero Extension) –

MOVZX destinaţie, sursă ;destinaţie ← sursă

Operanzi: reg,reg
reg,mem

Exemplu:

```
MOVZX   EAX, AL         ;octet → dublucuvânt
                        ; EAX=00 00 00|AL
```



PUSHFD (Push EFLAGS Registers)

PUSHFD *ESP ← ESP – 4*

[SS:ESP] ← EF

POPFD (Pop Stack into EFLAGS)

POPFD *EF ← [SS:ESP]*

ESP ← ESP + 4

PUSHF (Push 16-bit EFLAGS Registers)

PUSHF *ESP ← ESP – 2*

[SS:ESP] ← EFlow

POPF (Pop Stack into FLAGS) – aduce din stivă cei mai puţin semnificativi 16 biţi ai registrului **EF.** Instrucţiunea nu are operanzi, iar flagurile se schimbă în mod corespunzător.

POPFD *EFlow ← [SS:ESP]*

ESP ← ESP + 2

PUSHAD (Push 32-bit General Registers) –

;temp ← *ESP*

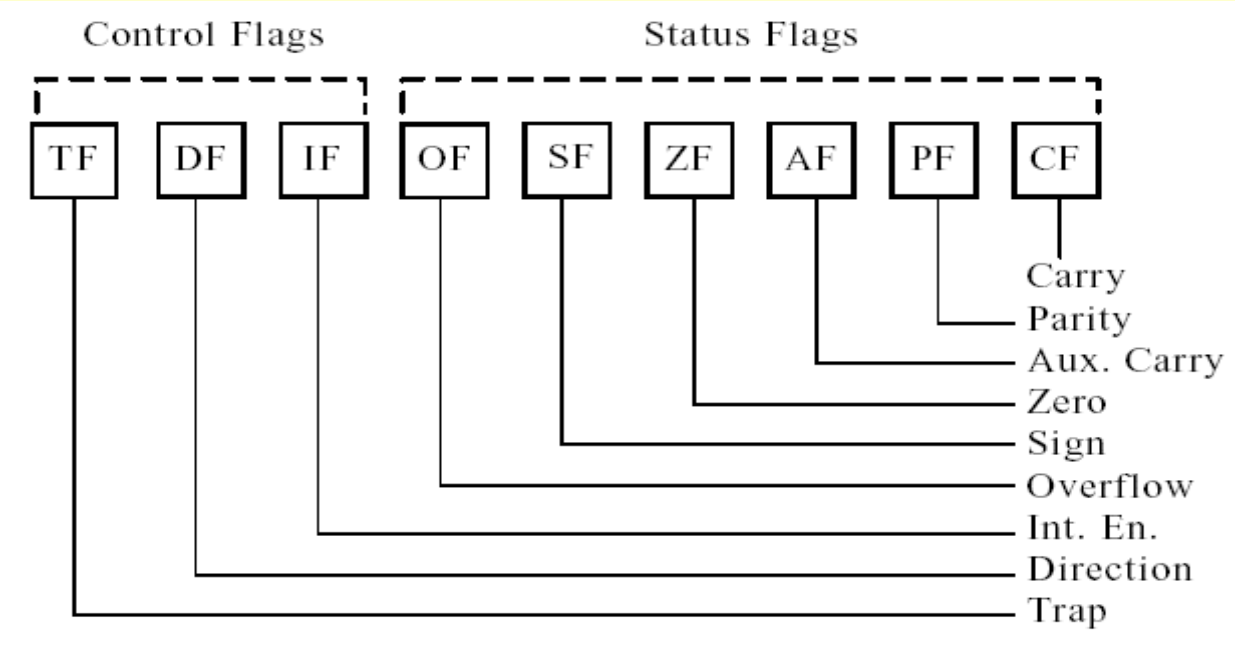
;PUSH EAX PUSH ECX PUSH EDX PUSH EBX

;PUSH temp PUSH EBP PUSH ESI PUSH EDI

POPAD (Pop All General Registers) –

SAHF (Store AH în EFLAGS) – încarcă conţinutul registrului AH în LSO al registrului EFlags, cu mascarea biţilor rezervaţi (7,6,4,2 şi 0)

;EF ← EF or (AH and 0D5H)



Lseg (Load Segment Register)

LDS reg, mem ;*destinaţie* ← *[sursă]*

LES reg, mem ;seg ← *[sursă+4]*

LFS reg, mem

LGS reg, mem

LSS reg, mem

BSWAP (Byte Swap) – Se realizează conversia între formatele "big-endian" şi "little-endian" schimbând ordinea celor 4 octeţi care compun data conţinută de registrul precizat.

BSWAP reg32 ;temp ← *reg*

reg[0…7] ← temp[*24…31]*

reg[8…15] ← temp[*16…24]*

reg[16…23] ← temp[*8…15]*

reg[24…31] ← temp[*0…7]*

Ex. 12345678h >> 78563412h

INSB, INSW, INSD (Input String from I/O Port)

– destinaţia dată de (ES:EDI) primeşte date de la portul de intrare, cu adresa specificata de către DX.

INSB, INSW, INSD (ES:EDI) ← port(DX)

(EDI) = (EDI)±(d) ;d=1 / 2 / 4

OUTSB, OUTSW, OUTSD (Output String to I/O Port)

– *OUTSB, OUTSW, OUTSD port(DX) ← (ES:ESI)*

(ESI) = (ESI)±(d);d=1 / 2 / 4

2. Instructiuni aritmetice

CMPXCHG (Compare and Exchange) –

CMPXCHG *operand1, operand2*

;ACC = operand1 ?? , dacă sunt egale atunci

;ZF=1 şi operand1:= operand2

;altfel ZF=0 ACC = operand2

Operanzi: reg,reg

mem,reg

XADD (Exchange and ADD)

XADD destinaţie, sursă ; dest. originală >> temp

; destinaţie = destinaţie + sursă şi
; dest. originală=temp >> sursă

Operanzi: reg,reg

mem,reg

IMUL (*Integer* (*Signed*) Multiplication) –

IMUL *op1*
IMUL *op1,op2*
IMUL *op1,op2,op3*

Operanzi:	**op1**	**op2**	**op3**	
	Reg			acc ← *acc * reg*
	Mem			acc ← *acc * mem*
	Reg,	reg		op1 ← *op1*op2*
	Reg,	mem		op1 ← *op1*op2*
	Reg,	imed		op1 ← *op1*op2*
	Reg,	reg,	imed	op1 ← *op2*op3*
	Reg,	mem,	imed	op1 ← *op2*op3*

BSF (Bit Scan Forward) –

BSF destinaţie, sursă; dacă (sursa = 0) atunci ZF = 1 şi destinaţia ←*???*
altfel ZF ← *0, temp ← 0*
while (bit(sursa, temp) =0)
temp ← temp+1
destinaţie ← *temp*
Operanzi: *reg,reg*
reg,mem

Exemplu:

```
MOV     AX, 85C0H       ; AX = 1000 0101 1100 0000 b
BSF     BX, AX          ;(BX) ← 6
```

BSR (Bit Scan Reverse) –

BSR destinaţie, sursă

; dacă destinaţia este (AX, BX,CX, DX, SI,DI, BP, SP)
atunci startbit ← 15; altfel startbit ← 31
dacă (sursa = 0) atunci ZF = 1 şi destinaţia ← ???
altfel ZF ← 0
temp ← startbit
while (bit (sursa, temp) = 0)
temp ← temp-1
destinaţie ← temp
Operanzi: *reg,reg*
reg,mem

Exemplu:

```
MOV     EAX, 34A00000H  ; EAX=00110100100101000 0....0h
BSR     BX, EAX         ;BX ← 29,  BX ← [log2 (EAX)],
                          partea întreagă a valorii  log2(AX)
```

BT (Bit Test) –

BT destinaţie, index *; CF ← destinaţie [index]*

Operanzi:
- reg, data
- mem, data
- reg, reg
- mem, reg

Exemplu:

```
MOV     EAX, 18          ;poziţia bitului care va fi testat
MOV     EBX, 0F749Eh     ;data pentru test
BT      EBX, EAX         ;CF ← 1   (1111 0111 0100 1001 1110)
JC      eticheta         ;salt conform bitului de test
```

BTC (Bit Test and Complement) –

BTC destinaţie, index *CF ← destinaţie [index]*

*destinaţie [index] ← **not** destinaţie [index]*

Operanzi: reg, data
mem, data
reg,reg
mem,reg

BTR (Bit Test and Reset)

BTR destinaţie, index *CF ← destinaţie [index]*

destinaţie [index] ← 0

Operanzi: reg, data
mem, data
reg,reg
mem,reg

BTS (Bit Test and Set)

BTR destinaţie, index *CF ← destinaţie [index]*

destinaţie [index] ← 1

Operanzi: reg, data
mem,data
reg,reg
mem,reg

SETcc dst (Set Byte on Condition) –

Condiţia *‚cc'* poate fi: A/ AE/ B/ BE/ C/ E/ G/ GE/ L/ LE/ NA/ NAE/ NB/ NBE/ NC/ NE/ NG/ NGE/ NL/ NLE/ NO/ NP/ NS/ NZ/ O/ P/ PE/ PO/ S/ Z

Exemplu:

```
SETC    dest    ;Set if Carry/ CF=1=> dest=1
SETLE   dest    ;Set if less or equal/ S ≠ O&Z=1 => dest=1
```

CMOVcc dest,src *(CMOVA, CMOVAE, CMOVB, CMOVBE, CMOVC, CMOVE, CMOVG, CMOVGE, CMOVL, CMOVLE, CMOVNA, CMOVNAE, CMOVNB, MOVNBE, CMOVNC, CMOVNE, CMOVNG, CMOVNGE, CMOVNL, CMOVNLE, CMOVNO, CMOVNP, CMOVNS, CMOVNZ, CMOVO, CMOVP, CMOVPE, CMOVPO, CMOVS, CMOVZ)*

Exemplu:

```
XOR EBX,EBX             ; EBX=0
ADD ECX,[val]           ; ECX=ECX+val
CMOVC ECX,EBX           ; If C=1, ECX=EBX  =0 else ECX=[val]
```

SHLD (Shift Left Double)

SHLD *destinaţie, sursă, count*

temp ← max (*count*, 31)

value ← sursă >>destinaţie

value ← value * 2temp

dest ← value

Operanzi:

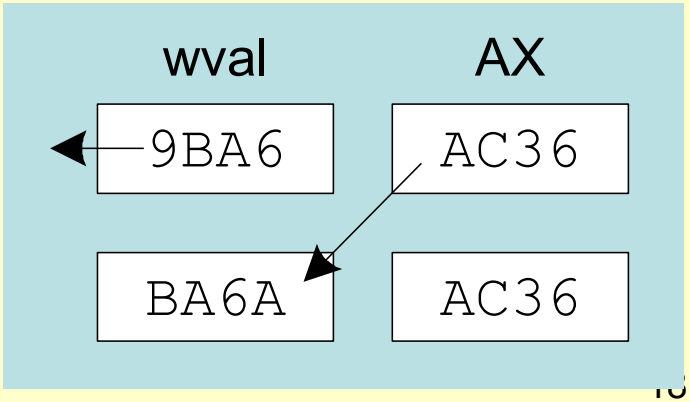
reg, reg, imed

mem, reg, imed

reg, reg, CL

mem, reg, CL

```
.data
wval WORD 9BA6h
.code
mov  ax,0AC36h
shld wval,ax,4
```



SHRD (Shift Right Double) –

SHRD *destinaţie, sursă, count*

temp ← max (*count*, 31)

value ← sursă >>destinaţie

value ← valoare div 2temp

dest ← valoare

Operanzi:

reg, reg, imed

mem, reg, imed

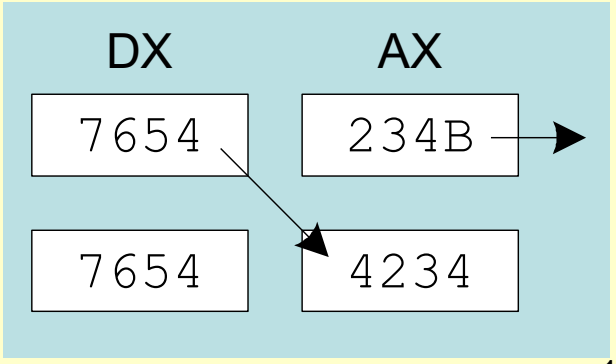
reg, reg, CL

mem, reg, CL

```
mov  ax,234Bh
mov  dx,7654h
shrd ax,dx,4
```

inainte:

dupa:



3. Instructiuni speciale

CPUID (CPU IDentification) – returnează informaţii de identificare a procesorului

INVD (Invalidate Cache) – invalidează memoria cache internă a procesorului;

MOV (Move Special) – copiază sau încarcă un registru special al CPU în /dintr-un registru general. Regiştrii speciali sunt CR0, CR2, CR3 (Control Register), respectiv DR0, DR1, DR2, DR3, DR6, DR7 (Debug Register).

MOV destinaţie, sursă; *(destinaţie) ← sursă*

Operanzi: reg,reg

Exemplu:

```
MOV   EAX, CR0      ;se salvează CR0 în EAX
MOV   DR7, ECX      ;se încarcă DR7 cu conţinutul ECX
```

RDMSR (Read from Model Specific Register) – conţinutul unui registru MSR, precizat de ECX, este copiat în **EDX | EAX**. Cu ajutorul MSR pot fi observate şi controlate detaliile specifice procesorului (configuration of OS-relevant things) – instr. privilegiate executate de OS.

RDMSR ; EDX | EAX ← MSR[ECX]

WRMSR (Write to Model Specific Register) – un operand din EDX | EAX este copiat într-un registru MSR

+ INSTRUCTIUNI MMX, SSE

```
// Exemplu.  MMX_CPUID_TEST.cpp : Defines the entry point for the //console application

#include “stdafx.h”

int _tmain(int argc, _TCHAR* argv[])
        {
        bool supMMX = true;
                asm {
                        mov supMMX, 1
                        mov EAX, 1
                        cpuid
                        test EDX, 0x800000          ;MMXsup =bit 23 ?
                        jnz sup
                        mov supMMX, 0
                sup:
                                                    };
                        if (supMMX) printf("MMX is supported!\n");
                    else
                        printf("MMX not supported!\n");
                        getchar();
                        return 0;
        };
```