Instructiuni

Partea I

Clasificare

- Importanta (grad de utilizare): uzuale, ocazionale, folosite foarte rar
- Tipul de procesor: 8086 (modul real), '386 (modul protejat), x87 (instructiuni flotante), Pentium II (MMX)
- Tipul operanzilor: 8/16 biti, 32 biti, 64 biti
- Complexitate: simple (o singura operatie, mod simplu de adresare, lungime scurta, timp redus de executie); complexe
- Gradul de cunostere: cunoscute, de care am auzit, necunoscute

Alta clasificare - dupa tipul operatiei

- **Transfer**: mov, lea, push, pop
- **Conversie**: cbw, xlat
- **Aritmetice**: add, sub, mul, div
- **Logice, de rotatie, deplasare si pe bit**: and, shl, rcr, bts
- **Pe siruri**: cmpsb, movsb
- **De control a fluxului de program**: jmp, call, jae
- **I/O**: in, out
- **Ale coprocesorului matematic**: fld, fdiv
- **MMX**: pmadd
- **Alte instructiuni**: clc, stc, rdtsc, cpuid

Instructiuni de transfer

Instructiunea MOV

MOV Dest, Src ;Dest <- Src

```
mov r/m8, r8
mov r/m16, r16
mov r/m32, r32
mov r8, r/m8
mov r16, r/m16
mov r32, r/m32
mov r/m8, imm8
mov r/m16, imm16
mov r/m32, imm32
mov Sreg, r/m16
mov r16/r32/m16, Sreg
```

```
1:      mov ax, bx
2:      mov [eax], bx
3:      mov al, [esi]
4:      mov ecx, 123
5:      mov al, eax
6:      mov [ebx], 123
7:      mov cs, 1234h
8:      mov [esi], var2
9:      mov cl, 278
```

Instructiunea MOV

MOV Dest, Src ;Dest <- Src

```
mov r/m8, r8
mov r/m16, r16
mov r/m32, r32
mov r8, r/m8
mov r16, r/m16
mov r32, r/m32
mov r/m8, imm8
mov r/m16, imm16
mov r/m32, imm32
mov Sreg, r/m16
mov r16/r32/m16, Sreg
```

```
1:      mov ax, bx
2:      mov [eax], bx
3:      mov al, [esi]
4:      mov ecx, 123
5:      mov al, eax
6:      mov [ebx], 123
7:      mov cs, 1234h
8:      mov [esi], var2
9:      mov cl, 278
```

Instructiunea LEA - Load Effective Address

LEA Dest, Src

;Dest <- OffsetAddr (Src)

```
lea r16, m
lea r32, m
```

Incarcare pointer near intr-un registru

```
1:   lea eax, [ebx]
2:   lea esi, var1
3:   lea ax, [bx+4]
4:   lea ecx, [ebx+esi]
5:   lea esi, eax
6:   lea [esi], eax
7:   lea esi, 1234h
```

Instructiunea LEA - Load Effective Address

LEA Dest, Src

;Dest <- OffsetAddr (Src)

```
lea r16, m
lea r32, m
```

Incarcare pointer near intr-un registru

```
1:   lea eax, [ebx]
2:   lea esi, var1
3:   lea ax, [bx+4]
4:   lea ecx, [ebx+esi]
5:   lea esi, eax
6:   lea [esi], eax
7:   lea esi, 1234h
```

LDS/LES/LFS/LGS/LSS - Load Far Pointer

```
LDS r16,m16:16
LDS r32,m16:32
;DS<-selector r<-offset (m)

LSS r16,m16:16
LSS r32,m16:32
;SS<-selector r<-offset (m)

LES r16,m16:16
LES r32,m16:32
;ES<-selector r<-offset (m)
```

*la fel LGS, LFS

```
1:      lds eax, [ebx]
2:      les esi, var1
3:      lss ebp, p
2:      les si, v1
```

Transfer cu stiva

;pune pe stiva valoarea din Src
PUSH Src
`push r/m16`
`push r/m32`

;scoate de pe stiva si pune in Dest
POP Dest
`pop r/m16`
`pop r/m32`

PUSHA ; push toti r16 pe stiva
PUSHAD ; push toti r32 pe stiva
; EAX, ECX, EDX, EBX, ESP, EBP, ESI, si EDI

POPA ; pop de pe stiva in toti r16
POPAD ; pop de pe stiva in toti r32
; EDI, ESI, EBP, EBX, EDX, ECX, si EAX

PUSHF ; push low 16b din eFlags
PUSHFD ; push eFlags
POPF ;pop de pe stiva in low 16 eFlags
POPFD ;pop de pe stiva in eFlags

Alte instructiuni de transfer

XCHG r/m, r

interschimba continutul operanzilor

LAHF

AH ← EFLAGS(SF:ZF:0:AF:0:PF:1:CF)

SAHF

EFLAGS(SF:ZF:0:AF:0:PF:1:CF) ← AH

BSWAP r32

(la procesoarele 486 si mai noi)

inverseaza ordinea octetilor

Instructiuni de conversie

MOVZX dest, src
extinde *src* cu 0 si copiaza in *dest*

MOVSX dest, src
extinde *src* cu semnul si copiaza in *dest*

CBW
AX ← sign-extend AL

CWD
DX:AX ← sign-extend AX

CWDE
EAX ← sign-extend AX

CDQ
EDX:EAX ← sign-extend of EAX.

XLAT m8
AL ← (DS:BX + ZeroExtend(AL))

Exercitii

Sa se scrie un program care pune pe stiva o valoare pe byte, o valoare pe word si una pe doubleword. Indicati cum se schimba valoarea ESP dupa fiecare operatie.

Rezolvare

```
…
MOV EAX, 0
MOV AL, 8
CBW
PUSH AX ;ESP = ESP-2
MOV AX, 16
PUSH AX ;ESP = ESP-2
MOV EAX, 333
PUSH EAX ;ESP = ESP-4
…
```

Moduri de adresare

Operanzi

- Date imediate
  - Procesorul citeste operandul din codul instructiunii
  - Timp de acces minim
- Registru
  - Procesorul citeste operandul din registru
  - Timp de acces minim
- Locatie de memorie interna
  - Access pe baza de adresa
  - Ciclu de transfer pe magistrala
  - Timp de access mediu
- Porturi I/O
  - Registre continute in interfetele de intrare/iesire HW
  - Access pe baza de adresa
  - Instructiuni speciale sau prin functii expuse de SO

Locatia de memorie: specificarea adresei

SELECTOR: OFFSET(sau Adresa liniara)

(selector-16b;offset sau adr. Liniara 32b)

MOV ES:[EBX], EAX

Specificarea offsetului/adresei liniare:

Offset = Baza+Index*Scala+Deplasament

MOV EAX, [EBX+ESI*2+12h]

MOV EAX, [EBX][EDI]

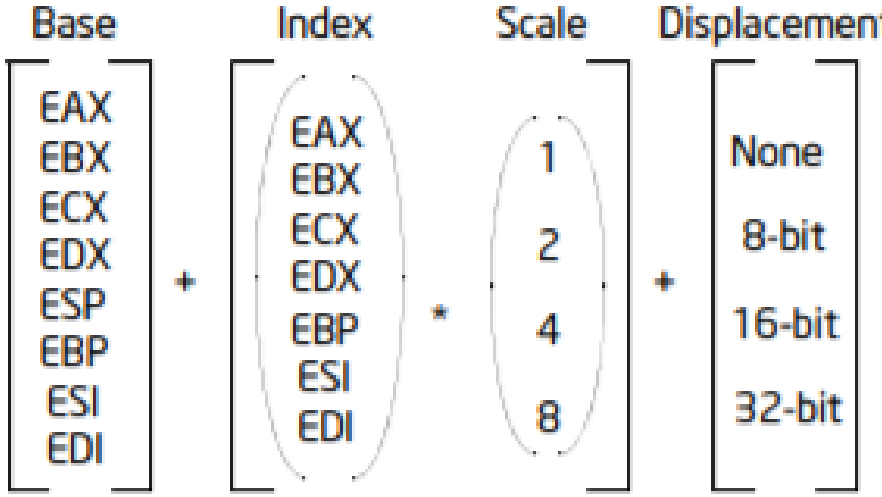
Baza - valoare intr-un registru general

Index - valoare intr-un registru general

Scala - valoare egala cu 2, 4 sau 8

Deplasament - valoare pe 8, 16 sau 32 de biti

Din manualul Intel IA-32



Deplasament

MOV EAX, [012345h]

Baza

MOV EAX, [ESI]

Baza+Deplasament

MOV EAX, [EBX+8]

Index*Scala+Deplasament

MOV EAX, [ESI*2+4]

Baza+Index+Deplasament

MOV EAX, [EBX+EDI+16]

Adresare imediata

- Operandul este o constanta
- Operandul este continut in codul instructiunii

- Avantaje
  - Operandul este citit o data cu instructiunea
  - Timp de executie foarte mic
- Dezavantaje
  - Instructiunea poate lucra cu o singura valoare
  - Lungimea constantei este in acord cu celalalt operand
  - Flexibilitate limitata

Adresare tip registru

- Operandul este continut intr-un registru al UCP

- Avantaje
  - Timp de acces foarte mic; nu necesita ciclu de transfer pe magistrala
  - Instructiune scurta (nr. mic de biti pt. specificare operand)
  - Timp de executie foarte mic
- Dezavantaje
  - Numar limitat de registre interne => nu toate variabilele pot fi pastrate in registre
  - Exista limitari in privinta registrelor speciale (ex: registrele segment)

Adresare directa

- Operandul este specificat printr-o adresa de memorie (adresa relativa fata de inceputul unui segment - **Deplasament**)
- Adresarea directa se foloseste pt. variabile simple (date nestructurate)

- Avantaje
  - Adresa operandului este continuta in codul instructiunii
- Dezavantaje
  - Instructiunea poate lucra cu o singura locatie de memorie (octet, cuvant sau dublu-cuvand)
  - Necesita ciclu suplimentar de transfer cu memoria =>timp de executie mai mare

Adresare indirecta

- Adresare **bazata** sau **indexata**
  - Offset=Baza
  - Offset=Baza+Deplasament
  - Offset=Index
  - Offset=Index+Deplasament
- Adresare **indexat-scalata**
  - Offset=Index*Scala+Deplasament
- Adresare **mixta**
  - Offset=Baza+Index*Scala+Deplasament

- Adresa de baza si index-ul se afla in registri
- Scala si deplasamentul sunt constante
- Moduri de adresare folosite pentru structuri de date de diferite feluri
- Necesita executia de operatii pentru calculul adresei de offset
- D: Timp de executie mare
- A: Flexibilitate maxima

Exercitii

```
1:    mov ax, bx
2:    mov [eax], bx
3:    mov al, [esi+4]
4:    mov ecx, 123
5:    mov al, [ebx+esi*2+4]
6:    mov [ebx+edi], 123
7:    mov bx, [1234h]
8:    mov al, var2
9:    mov cl, var[ebx][eax]
```

- Precizati, pentru fiecare linie de cod, ce moduri de adresare sunt folosite

Rezolvare

1: mov ax, bx ;tip registru ambii operanzi
2: mov [eax], bx ;tip indirect (indexat sau bazat) op1, tip registru op2
3: mov al, [esi+4] ;tip reg op1, tip indirect (baza/index+deplasament) op2
4: mov ecx, 123 ;tip reg op1, tip imediat op2
5: mov al, [ebx+esi*2+4] ;tip reg op1, tip indirect (mixt) op2
6: mov [ebx+edi], 123 ;tip indirect (mixt) op1, tip imediat op2
7: mov bx, [1234h] ;tip registru op1, adresare directa op2
8: mov al, var2 ;tip registru op1, adresare directa op2
9: mov cl, var[ebx][eax] ;tip registru op1, indirect (mixt) op2

Instructiuni aritmetice

Adunare, scadere

ADD dest, src

dest=dest + src

ADD r/m, imm

ADD r/m, r

ADD r, r/m

Flags: OF, SF, ZF, AF, CF si PF

ADC dest, src

dest=dest + src + CF

la fel cu ADD

SUB dest, src

dest=dest - src

SUB r/m, imm

SUB r/m, r

SUB r, r/m

Flags: OF, SF, ZF, AF, CF si PF

SBB dest, src

dest=dest - (src + CF)

la fel cu SUB

CMP dest, src

Scadere fara pastrarea rezultatului.

Se seteaza **doar flag-urile**.

INC dest

dest=dest+1

INC r/m

Flags: OF, SF, ZF, AF si PF

DEC dest

dest=dest-1

DEC r/m

NEG dest

NEG r/m

dest = -dest

CF=0 daca dest=0, altfel CF=1

Inmultire, impartire

MUL r/m

Dest: AL, AX, EAX

Src: Operand 1

AX = AL*Src8

DX:AX = AX*Src16

EDX:EAX = EAX*Src32

Flags: CF si OF = 0 daca jumatatea superioara a rezultatului e 0, altfel =1;

Restul flag-urilor sunt nedefinite.

IMUL r/m

identic cu MUL

IMUL r, r/m

Dest = Truncate(Dest*Src)

IMUL r, r/m, imm

Dest = Truncate(Src1*Src2)

Flags: la variantele cu 2 si 3 operanzi CF si OF = 1 daca rezultatele trebuie trunchiate.

DIV r/m

Dest: AX, DX:AX, EDX:EAX

Src: Operand 1

AL=AX/Src8; AH=AX MOD Src8;

AX=DX:AX/Src16;

DX=DX:AX MOD Src16;

EAX=EDX:EAX/Src32;

EDX=EDX:EAX MOD Src32;

Flags: nedefinite

IDIV r/m

Impartire cu semn.

Aceleasi reguli ca si la DIV.

Instructiuni logice, de deplasare, de rotire si pe bit

Instructiuni logice

AND Dest, Src
Dest = Dest AND Src
AND r/m, r/imm
AND r, r/m
Flags: CF, OF = 0; SF, ZF, PF conform rezultatului.

OR Dest, Src
Dest = Dest OR Src
OR r/m, r/imm
OR r, r/m
Flags: CF, OF = 0; SF, ZF, PF conform rezultatului.

XOR Dest, Src
Dest = Dest XOR Src
XOR r/m, r/imm
XOR r, r/m
Flags: CF, OF = 0; SF, ZF, PF conform rezultatului.

NOT Dest
Dest = NOT Dest
NOT r/m
Flags: nu sunt afectate

Instructiuni de deplasare

SHL/SAL shift left logic/aritmetic **sunt echivalente**

SHR, **SAR** shift right logic, arithmetic

Shift logic: se completeaza cu 0

Shift aritmetic la dreapta: se completeaza cu valoarea bitului de semn

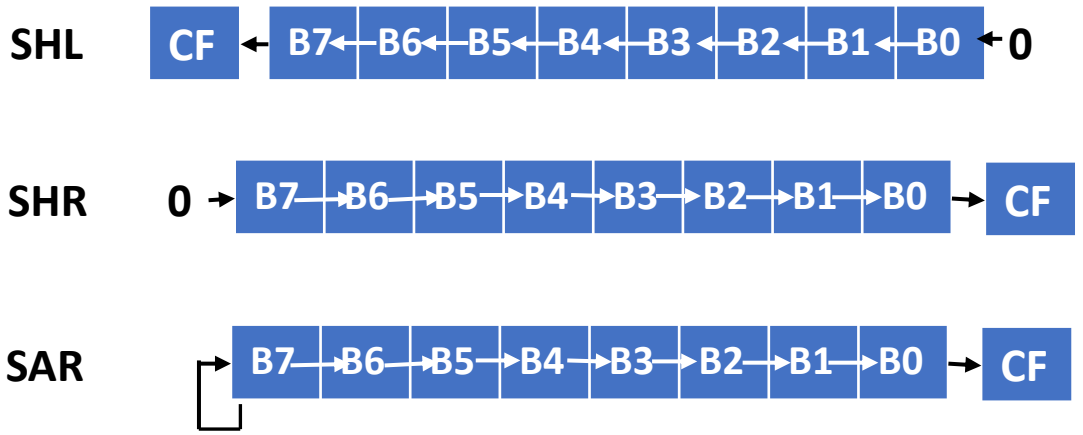
Instructiunile au aceeasi sintaxa:

SHL r/m, 1/CL/imm8

Operand 1 - sursa

Operand 2 - numar de pozitii

Flags: CF, OF; SF, ZF, PF -conform rezultatului



Instructiuni de rotire

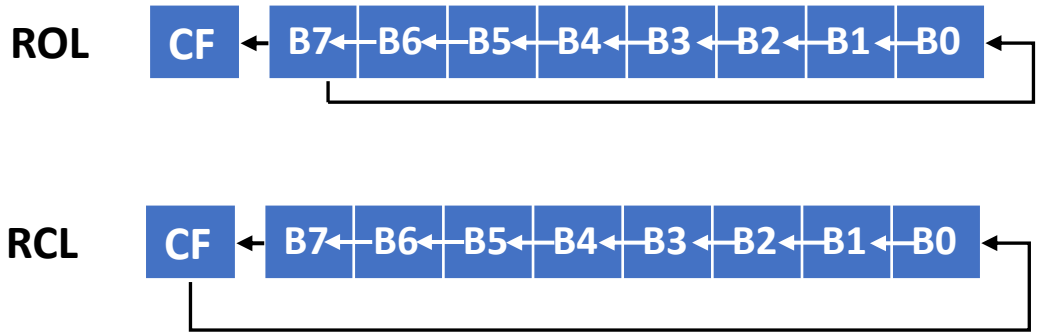
ROL, **ROR**, **RCL**, **RCR** - Rotate

ROL r/m, 1/CL/imm8

Operand 1 - sursa

Operand 2 - numar de pozitii

Flags: CF, OF; SF, ZF, PF -conform rezultatului



Instructiuni pe bit

TEST r/m, imm/r

Operatie: operand 1 AND operand 2

Rezultatul: nu se pastreaza

Flag-uri afectate: OF=0, CF=0, SF, ZF, PF conform rezultatului

BT r/m, r/imm

CF ← Bit(BitBase, BitOffset)

Flag-uri afectate: CF

BTC r/m, r/imm

CF ← Bit(BitBase, BitOffset);

Bit(BitBase, BitOffset) ← NOT Bit(BitBase, BitOffset);

BTS r/m, r/imm

CF ← Bit(BitBase, BitOffset);

Bit(BitBase, BitOffset) ← 1;

BTR r/m, r/imm

CF ← Bit(BitBase, BitOffset);

Bit(BitBase, BitOffset) ← 0;

Instructiuni de intrare/iesire

IN AL/AX/EAX, imm8

IN AL/AX/EAX, DX

copiaza in AL/AX/EAX date de la portul cu adresa imm8 sau specificata in DX

OUT imm8, AL/AX/EAX

OUT DX, AL/AX/EAX

trimite la portul cu adresa imm8 sau specificata in DX datele din AL/AX/EAX

Test