Microprocessors (0630371)

Fall 2010/2011 – Lecture Notes # 10

Data Transfers, Addressing, and Arithmetic

Outline of the Lecture

- **Operand Types.**
- **MOV Instruction and overlapping values.**
- **Zero and Sign Extension and instructions.**
- **LAHF and SAHF Instructions.**
- **XCHG Instruction.**
- **Direct-Offset Operands and Instructions.**
- **Programming Example.**

Operand Types

Three basic types of operands:

- **Immediate operand**– a constant integer (8, 16, or 32 bits) value is encoded within the instruction.
- **Register operand** – the name of a register, register name is converted to a number and encoded within the instruction.
- **Memory operand** (**Direct operand**)– reference to a location in memory, memory address is encoded within the instruction, or a register holds the address of a memory location
  - A direct memory operand is a named reference to storage in memory.
  - The named reference (label) is automatically dereferenced by the assembler.

```
.data
var1 BYTE 10h
.code
mov al,var1 ; AL = 10h
mov al,[var1] ; AL = 10h, this is an alternate format
```

MOV Instruction and overlapping values

- The given register can be modified using differently sized data.
- When one word is moved to AX, it overwrites the existing value of AL.
- When one double word is moved to EAX, it overwrites the existing value of AX.

Example

```
.data
OneByte BYTE 78h
oneWord WORD 1234h
oneDword DWORD 12345678h
.code
mov  eax, 0         ; EAX = 00000000h
mov  al, OneByte    ; EAX = 00000078h
mov  ax, oneWord    ; EAX = 00001234h
mov  eax, oneDword  ; EAX = 12345678h
mov  ax,0           ; EAX = 12340000h
```

Zero and Sign Extension

Copying Smaller Values into Larger Ones

- What happens if we write:

```
.data
count WORD 1
.code
mov ecx, 0
mov cx, count; ECX is 0
```

Copying Smaller Values into Larger Ones

- What happens if we write:

```
.data
signedVal SWORD -16       ; FFF0h (-16)
.code
mov ecx, 0
mov cx, SignedVal; ECX is 0000FFF0h (+65520)
```

- The value in ECX is completely different from -16, we could solve the problem by writing:

```
mov ecx, FFFFFFFFx
mov cx, signedVal ; ECX=FFFFFFF0h (-16)
```

MOVZX (Zero Extension instruction)

- When you copy a smaller value into a larger destination, the **MOVZX** instruction fills (extends) the upper half of the destination with **zeros**.
- There are 3 formats:

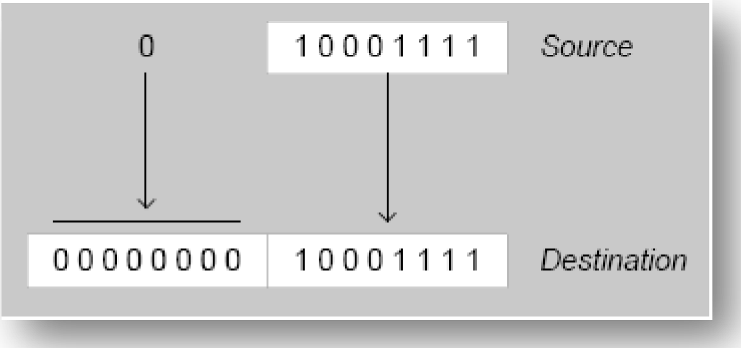
```
movzx r32, r/m8
movzx r32, r/m16
movzx r16, r/m8
```

- The destination must be a register.

Examples of MOVZX

Example 1: Register to register

```
mov bl,10001111b
movzx ax,bl ; zero-extension
```



Example 2: Register to register

```
mov bx, 0A69Bh
movzx eax, bx ; EAX = 0000A69Bh
movzx edx, bl ; EDX = 0000009Bh
movzx cx, bl ; CX = 009Bh
```

Example 3: Memory to register

```
.data
byte1 BYTE 9Bh
word1 WORD 0A69Bh
.code
movzx eax, word1 ; EAX = 0000A69Bh
movzx edx, byte1 ; EDX = 0000009Bh
movzz cx, byte1 ; CX = 009Bh
```

MOVSX (Sign Extension instruction)

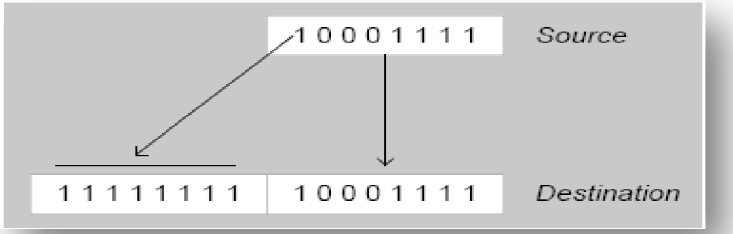
- The MOVSX instruction fills the upper half of the destination with a copy of the source operand's sign bit.
- There are 3 formats:

```
movsx r32, r/m8
movsx r32, r/m16
movsx r16, r/m8
```

- MOVSX is only used with signed integers.
- The destination must be a register.

Example 1: Register to register

```
mov bl,10001111b
movsx ax,bl ; sign extension
```



Example 2: Register to register

```
mov bx, 0A69Bh
movzx eax, bx ; EAX = FFFFA69Bh
movzx edx, bl ; EDX = FFFFFF9Bh
movzx cx, bl ; CX = FF9Bh
```

LAHF and SAHF Instructions

- Loads/Stores flag values from/to EFLAGS register into/from AH
- **SAHF** copies the value of bits 7, 6, 4, 2, 0 of the AH register into the SF, ZF, AF, PF, and CF respectively. This instruction was provided to make easier conversion of assembly language program written for 8080 and 8085 to 8086.

```
.data
saveflags BYTE ?
.code
lahf ; load flags into AH
mov saveflags,ah ; save them in a variable
.
.
.
mov ah,saveflags ; load saved flags into AH
sahf ; copy into Flags register
```

XCHG Instruction

```
XCHG destination, source
```

- The syntax is:

```
xchg reg, reg
xchg reg, mem
xchg mem, reg
```

- This does not require the use of a third location to swap values, making it very useful.
- The Exchange instruction exchanges the contents of the register with the contents of another register (or) the contents of the register with the contents of the memory location. Direct memory to memory exchanges are not supported.
- The both operands must be the same size and one of the operand must always be a register.
- No immediate operands are permitted.

Example 1:

```
    XCHG AX, DX         ; Exchange word in AX with word
in DX
    XCHG BL, CH         ; Exchange byte in BL with byte
in CH
    XCHG AL, Money [BX] ; Exchange byte in AL with byte
                        ; in memory at EA.
```

Example 2:

```
.Data
var1 WORD 1000h
var2 WORD 2000h
.code
xchg ax,bx     ; exchange 16-bit regs
xchg ah,al     ; exchange 8-bit regs
xchg var1,bx   ; exchange mem, reg
xchg eax,ebx   ; exchange 32-bit regs
xchg var1,var2 ; error: two memory operands
```

Direct-Offset Operands and Instructions

- A **constant offset** is added to a data label to produce an **effective address (EA)**. The address is dereferenced to get the value inside its memory location.

```
.data
arrayB BYTE 10h,20h,30h,40h
.code
mov al,arrayB+1 ; AL = 20h
mov al,[arrayB+1] ; alternative notation
```

- A constant offset is added to a data label to produce an effective address (EA). The address is dereferenced to get the value inside its memory location.

```
.data
arrayW WORD 1000h,2000h,3000h
arrayD DWORD 1,2,3,4
.code
mov ax,[arrayW+2] ; AX = 2000h
mov ax,[arrayW+4] ; AX = 3000h
mov eax,[arrayD+4] ; EAX = 00000002h
; Will the following statements assemble?
mov ax,[arrayW-2] ; ??
mov eax,[arrayD+16] ; ??
```

Programming Example

```
TITLE Data Transfer Examples       (Moves.asm)
; Chapter 4 example. Demonstration of MOV and
; XCHG with direct and direct-offset operands.
INCLUDE Irvine32.inc
.data
val1  WORD 1000h
val2  WORD 2000h
arrayB BYTE  10h,20h,30h,40h,50h
arrayW WORD  100h,200h,300h
arrayD DWORD 10000h,20000h
.code
main PROC
;  MOVZX
 mov    bx,0A69Bh
 movzx  eax,bx       ; EAX = 0000A69Bh
 movzx  edx,bl       ; EDX = 0000009Bh
 movzx  cx,bl        ; CX  = 009Bh
;  MOVSX
 mov   bx,0A69Bh
 movsx eax,bx        ; EAX = FFFFA69Bh
 movsx edx,bl        ; EDX = FFFFFF9Bh
 mov  bl,7Bh
 movsx cx,bl         ; CX  = 007Bh
;  Memory-to-memory exchange:
 mov  ax,val1        ; AX = 1000h
 xchg ax,val2        ; AX = 2000h, val2 = 1000h
 mov  val1,ax        ; val1 = 2000h
;  Direct-Offset Addressing (byte array):
 mov al,arrayB       ; AL = 10h
 mov al,[arrayB+1]   ; AL = 20h
 mov al,[arrayB+2]   ; AL = 30h
;  Direct-Offset Addressing (word array):
 mov ax,arrayW       ; AX = 100h
 mov ax,[arrayW+2]   ; AX = 200h
;  Direct-Offset Addressing (doubleword array):
 mov eax,arrayD                ; EAX = 10000h
 mov eax,[arrayD+4]            ; EAX = 20000h
 mov eax,[arrayD+TYPE arrayD]  ; EAX = 20000h
 exit
main ENDP
END main
```