Assembly Language for Intel-Based Computers, 4th Edition

Kip R. Irvine

Chapter 4: Data Transfers, Addressing, and Arithmetic

Lecture 14

Slides prepared by Kip R. Irvine

Revision date: 09/26/2002

Modified by Dr. Nikolay Metodiev Sirakov-3/1/2005

- Chapter corrections (Web)   Assembly language sources (Web)

Lecture 14-Data Transfer Instructions

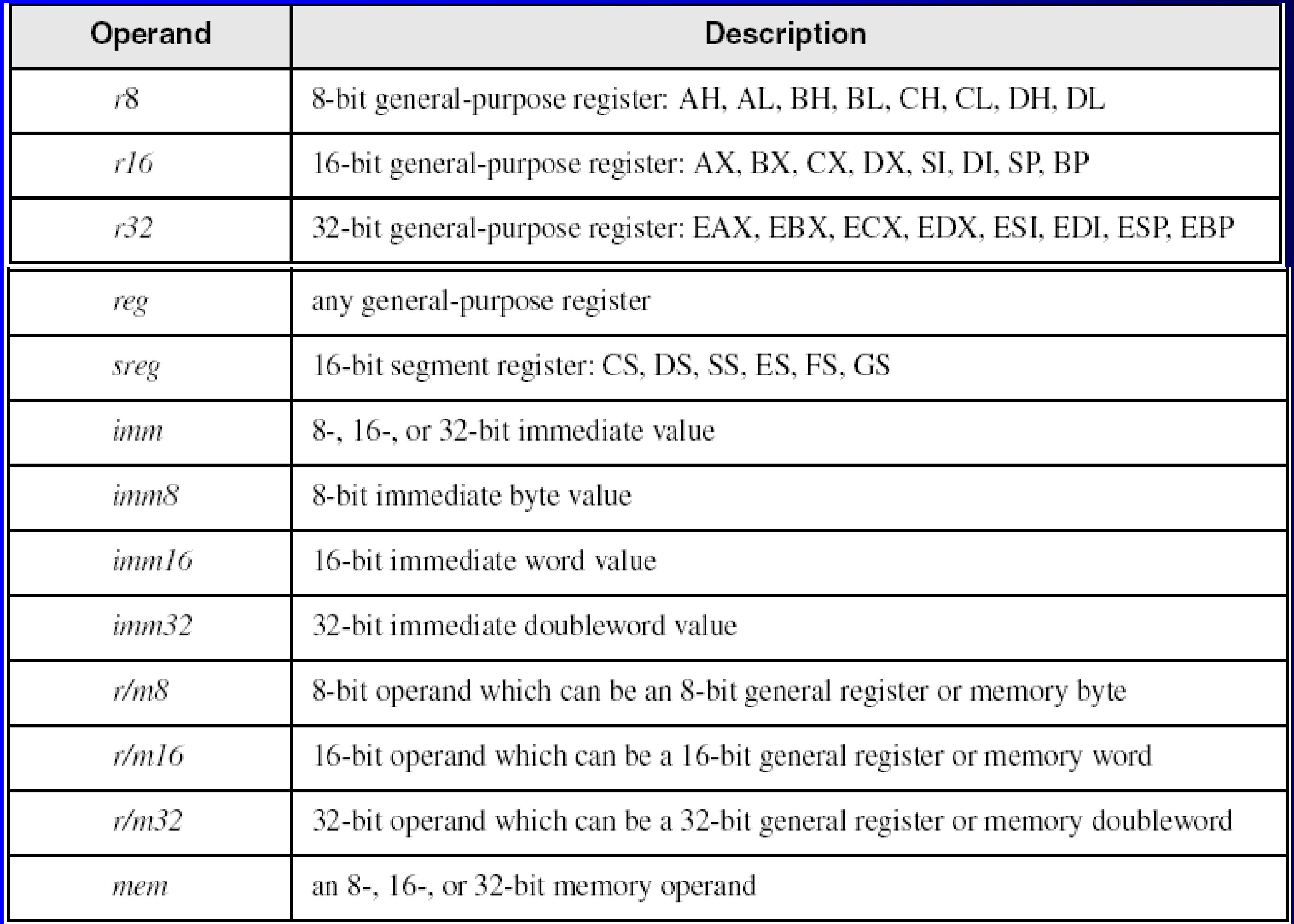
Data Transfer Instructions

- Operand Types
- Instruction Operand Notation
- Direct Memory Operands
- MOV Instruction
- Zero & Sign Extension
- LAHF and SAHF Instructions
- XCHG Instruction
- Direct-Offset Instructions

Operand Types

- Three basic types of operands:
  - Immediate – a constant integer (8, 16, or 32 bits)
    - value is encoded within the instruction
  - Register – the name of a register
    - register name is converted to a number and encoded within the instruction
  - Memory – reference to a location in memory
    - memory address is encoded within the instruction, or a register holds the address of a memory location

Instruction Operand Notation



Operand	Description
r8	8-bit general-purpose register: AH, AL, BH, BL, CH, CL, DH, DL
r16	16-bit general-purpose register: AX, BX, CX, DX, SI, DI, SP, BP
r32	32-bit general-purpose register: EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP
reg	any general-purpose register
sreg	16-bit segment register: CS, DS, SS, ES, FS, GS
imm	8-, 16-, or 32-bit immediate value
imm8	8-bit immediate byte value
imm16	16-bit immediate word value
imm32	32-bit immediate doubleword value
r/m8	8-bit operand which can be an 8-bit general register or memory byte
r/m16	16-bit operand which can be a 16-bit general register or memory word
r/m32	32-bit operand which can be a 32-bit general register or memory doubleword
mem	an 8-, 16-, or 32-bit memory operand

Direct Memory Operands

- 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
```

alternate format

MOV Instruction

- Move from source to destination. Syntax:

  MOV *destination,source*

- No more than one memory operand permitted
- CS, EIP, and IP cannot be the destination
- No immediate to segment moves

```
.data
count BYTE 100
wVal  WORD 2
.code
    mov bl,count
    mov ax,wVal
    mov count,al

    mov al,wVal                   ; error
    mov ax,count                  ; error
    mov eax,count                 ; error
```

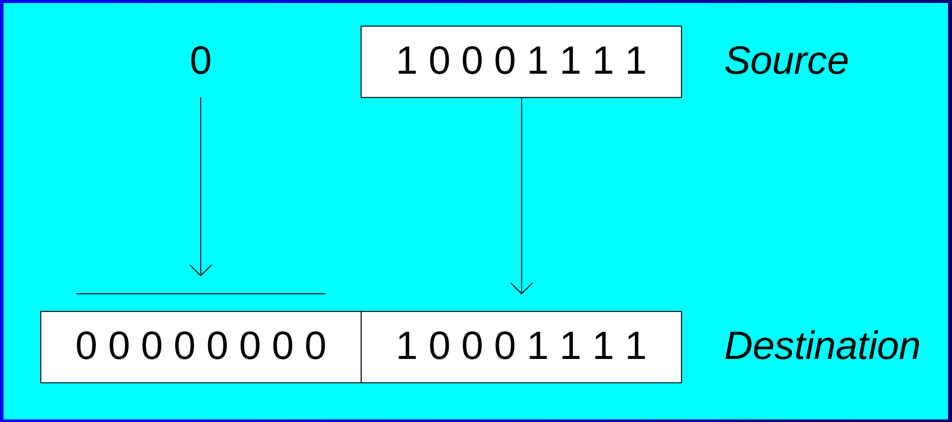
Your turn . . .

Explain why each of the following MOV statements are invalid:

```
.data
bVal   BYTE    100
bVal2  BYTE    ?
wVal   WORD    2
dVal   DWORD   5
.code
    mov ds,45                ; a.
    mov esi,wVal             ; b.
    mov eip,dVal             ; c.
    mov 25,bVal              ; d.
    mov bVal2,bVal           ; e.
```

Zero Extension

When you copy a smaller value into a larger destination, the MOVZX instruction fills (extends) the upper half of the destination with zeros.

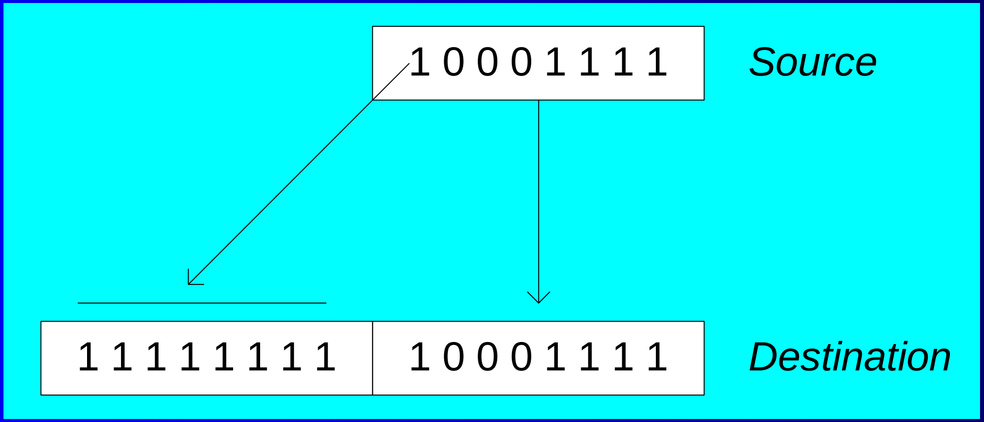


```
mov bl,10001111b
movzx ax,bl                     ; zero-extension
```

The destination must be a register.

Sign Extension

The MOVSX instruction fills the upper half of the destination with a copy of the source operand's sign bit.



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

The destination must be a register.

XCHG Instruction

XCHG exchanges the values of two operands. At least one operand must be a register. No immediate operands are permitted.

```
.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

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
```

Q: Why doesn't arrayB+1 produce 11h?

Direct-Offset Operands (cont)

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]        ; ??
```

What will happen when they run?

Your turn. . .

Write a program that rearranges the values of three doubleword values in the following array as: AF, AD, AE.

```
.data
arrayD DWORD AD, AE, AF
```

- Step1: copy the first value into EAX and exchange it with the value in the second position.

```
mov eax,arrayD
xchg eax,[arrayD+4]
```

- Step 2: Exchange EAX with the third array value and copy the value in EAX to the first array position.

```
xchg eax,[arrayD+8]
mov  arrayD,eax
```

Evaluate this . . .

- We want to write a program that adds the following three bytes:

```
.data
myBytes BYTE 80h,66h,0A5h
```

- What is your evaluation of the following code?

```
mov al,myBytes
add al,[myBytes+1]
add al,[myBytes+2]
```

- What is your evaluation of the following code?

```
mov ax,myBytes
add ax,[myBytes+1]
add ax,[myBytes+2]
```

- Any other possibilities?

Evaluate this . . . (cont)

```
.data
myBytes BYTE 80h,66h,0A5h
```

- How about the following code. Is anything missing?

```
movzx ax,myBytes
mov   bl,[myBytes+1]
add   ax,bx
mov   bl,[myBytes+2]
add   ax,bx                   ; AX = sum
```

Yes: Move zero to BX before the MOVZX instruction.

Addition and Subtraction

- INC and DEC Instructions
- ADD and SUB Instructions
- NEG Instruction
- Implementing Arithmetic Expressions
- Flags Affected by Arithmetic
  - Zero
  - Sign
  - Carry
  - Overflow

INC and DEC Instructions

- Add 1, subtract 1 from destination operand
  - operand may be register or memory
- INC *destination*
  - Logic: *destination* ← *destination* + 1
- DEC *destination*
  - Logic: *destination* ← *destination* – 1

INC and DEC Examples

```
.data
myWord  WORD 1000h
myDword DWORD 10000000h
.code
   inc myWord             ; 1001h
   dec myWord             ; 1000h
   inc myDword            ; 10000001h

   mov ax,00FFh
   inc ax                 ; AX = 0100h
   mov ax,00FFh
   inc al                 ; AX = 0000h
```

Your turn...

Show the value of the destination operand after each of the following instructions executes:

```
 .data
myByte BYTE 0FFh, 0
 .code
    mov al,myByte              ; AL = FFh
    mov ah,[myByte+1]          ; AH = 00h
    dec ah                     ; AH = FFh
    inc al                     ; AL = 00h
    dec ax                     ; AX = FEFF
```