

ECE390

Computer Engineering II

Lecture 3



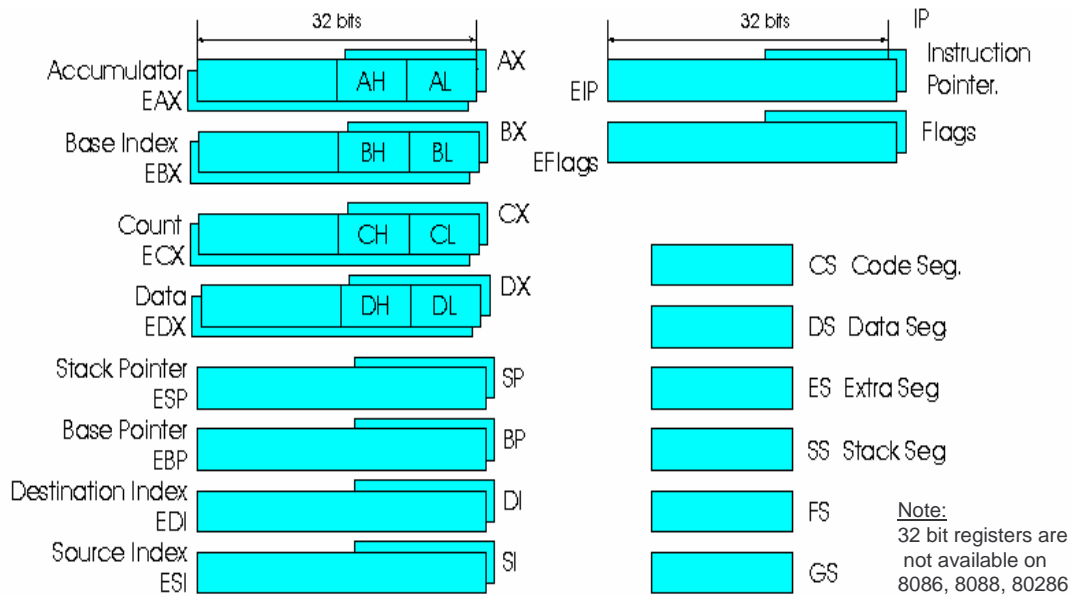
Dr. Zbigniew Kalbarczyk
University of Illinois at Urbana- Champaign

Outline



- Programming with registers
- Addressing modes
- Instruction components and format
- NASM basics

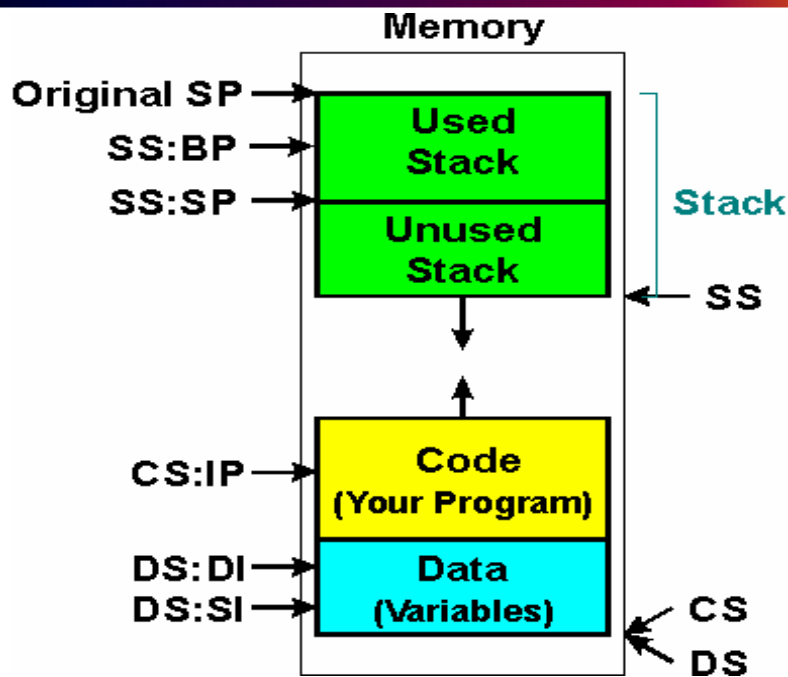
Programming Model Registers



Z. Kalbarczyk

ECE390

Use of Segments



Z. Kalbarczyk

ECE390

The MOV instruction

- MOV moves data from one place to another
 - register to register
 - register to memory
 - memory to register
 - immediate data to register
 - **NEVER** memory to memory
- Syntax: **MOV destination, source**
 - Destination and source are determined by the Addressing Mode.

Addressing Modes

- **Register** - transfers a byte or word from the source register or memory location to the destination register or memory location
MOV AX, BX
- **Immediate** - transfers an immediate byte or word of data into the destination register or memory location
MOV AX, 3456h
- **Direct** - moves a byte or word between a memory location and a register
MOV AL, [1234h] (1234h is treated as a displacement within data segment)

Addressing Modes(cont.)

- **Register Indirect** (*base relative or indexed*) - transfers a byte or word of data between a register and the memory location addressed by an index (DI or SI) or base register (BP or BX)

```
MOV AX, [BX]
```

- **Base Plus Index** (*base relative indexed*) - transfers a byte or word of data between a register and the memory location addressed by a base register (BP or BX) plus index (DI or SI) register

```
MOV DX, [BX + DI]
```

Addressing Modes(cont.)

- **Register Relative** - transfers a byte or word of data between a register and the memory location addressed by an index (DI or SI) or base register (BP or BX) plus displacement

```
MOV AX, [BX + 1000h]
```

- **Base Relative Plus Index** (*base relative indexed*) - transfers a byte or word of data between a register and the memory location addressed by a base register (BP or BX) plus an index register (DI or SI)

```
MOV AX, [BX + SI + 100h]
```

Addressing modes

- If you learn one thing about addressing modes, it is that you can reference memory with:

| | | |
|----|----|------|
| BX | SI | DISP |
| BP | DI | |

- Any above register (BX, BP, SI, DI) or an immediate displacement (DISP)
- Sum of either column 1 register with either column 2 register (BX+SI, BX+DI, BP+SI, BP+DI) with immediate displacement (DISP)
- Never with the sum of registers in the same column (not BX+BP or SI+DI)

Machine language

- **Machine language** is the native binary code that the CPU understands and uses as the instructions that control its operation.
- Interpretation of machine language allows debugging or modification at the machine language level.
- Machine language instructions are generally too complex to generate by hand. That's why we use NASM.
- 80x86 machine language instructions vary in length from 1 to as many as 13 bytes.
- There are over 20,000 variations of machine language instructions.

Machine language

- Real mode uses 16-bit instructions
 - This means that instructions use 16-bit offset addresses and 16-bit registers
 - This does *not* mean the instructions are 16-bits in length
- Protected mode can use 16 or 32 bit instructions
 - The 32-bit instruction mode assumes all offset addresses are 32 bits as well as all registers

Instruction format

| Opcode | Mode | Displacement | Data/Immediate |
|--------|------|--------------|----------------|
|--------|------|--------------|----------------|

| | | | | |
|-----------|---------------|---------------|--|--|
| OP | | | | No operands Example: NOP |
| OP | DATA8 | | | w/8-bit data Example: MOV AL, 15 |
| OP | DATA16 | | | w/16-bit data Example: MOV AX, 1234h |
| OP | DISP8 | | | w/8-bit displacement Example: JE +45 |
| OP | DISP16 | | | w/16-bit displacement Example: MOV AL, [1234h] |
| OP | MODE | | | w/mode – register to register Example: MOV AL, AH |
| OP | MODE | DISP8 | | w/mode & 8-bit displacement Example: MOV [BX + 12], AX |
| OP | MODE | DISP16 | | w/mode & 16-bit displacement Example: MOV [BX+1234], AX |

Addressing modes

- Each instruction can only access memory once
 - MOV [VAR1], [VAR2] **is invalid!**
 - MOV AX, [VAR2] followed by MOV [VAR1], AX is okay
- For 2-operand instructions, size of operands must match
 - Compare 8-bit numbers to 8-bit numbers
 - Compare 16-bit numbers to 16-bit numbers
 - CMP AH, AX **is invalid!**
- Mode byte encodes which registers the instruction uses
- When writing instructions, if data sizes aren't obvious from the context, you must explicitly state the size.
 - MOV BYTE [BX], 12h
 - MOV [BX], WORD 12h

Addressing mode examples

| Instruction | Comment | Addressing Mode | Memory Contents |
|-----------------|--|-----------------|-----------------|
| MOV AX, BX | Move to AX the 16-bit value in BX | Register | 89 D8 |
| MOV AX, DI | Move to AX the 16-bit value in DI | Register | 89 F8 |
| MOV AH, AL | Move to AL the 8-bit value in AX | Register | 88 C4 |
| MOV AH, 12h | Move to AH the 8-bit value 12H | Immediate | B4 12 |
| MOV AX, 1234h | Move to AX the value 1234h | Immediate | B8 34 |
| MOV AX, CONST | Move to AX the constant defined as CONST | Immediate | B8 1sb msb |
| MOV AX, X | Move to AX the address or offset of the variable X | Immediate | B8 1sb msb |
| MOV AX, [1234h] | Move to AX the value at memory location 1234h | Direct | A1 34 12 |
| MOV AX, [X] | Move to AX the value in memory location DS:X | Direct | A1 1sb msb |

Addressing mode examples

| Instruction | Comment | Addressing Mode | Memory Contents |
|-------------------|--|-------------------|-------------------------------------|
| MOV [X], AX | Move to the memory location pointed to by DS:X the value in AX | Direct | A3 lsb msb OP DATA16 |
| MOV AX, [DI] | Move to AX the 16-bit value pointed to by DS:DI | Indexed | 8B 05 OP MODE |
| MOV [DI], AX | Move to address DS:DI the 16-bit value in AX | Indexed | 89 05 OP MODE |
| MOV AX, [BX] | Move to AX the 16-bit value pointed to by DS:BX | Register Indirect | 8B 07 OP MODE |
| MOV [BX], AX | Move to the memory address DS:BX the 16-bit value stored in AX | Register Indirect | 89 07 OP MODE |
| MOV [BP], AX | Move to memory address SS:BP the 16-bit value in AX | Register Indirect | 89 46 OP MODE |
| MOV AX, TAB[BX] | Move to AX the value in memory at DS:BX + TAB | Register Relative | 8B 87 lsb msb OP MODE DISP16 |
| MOV TAB[BX], AX | Move value in AX to memory address DS:BX + TAB | Register Relative | 89 87 lsb msb OP MODE DISP16 |
| MOV AX, [BX + DI] | Move to AX the value in memory at DS:BX + DI | Base Plus Index | 8B 01 OP MODE |

Z. Kalbarczyk

ECE390

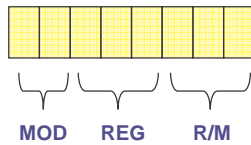
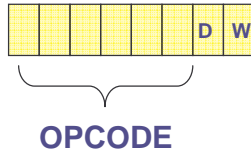
Addressing mode examples

| Instruction | Comment | Addressing Mode | Memory Contents |
|-----------------------------------|--|---------------------|-----------------------------------|
| MOV [BX + DI], AX | Move to the memory location pointed to by DS:X the value in AX | Base Plus Index | 89 01 OP MODE |
| MOV AX, [BX + DI + 1234h] | Move word in memory location DS:BX + DI + 1234h to AX register | Base Rel Plus Index | 8B 81 34 12 OP MODE DISP16 |
| MOV word [BX + DI + 1234h], 5678h | Move immediate value 5678h to memory location BX + DI + 1234h | Base Rel Plus Index | C7 81 34 12 78 56 |

Z. Kalbarczyk

ECE390

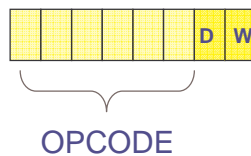
Instruction components



- The OPCODE byte contains the opcode, as well as direction (D) and data size (W) bits.
- The MODE byte only exists in instructions that use register addressing modes.
- The MODE byte encodes the target and source for 2-operand instructions.
- The target and source are specified in the REG and R/M fields.

Instruction components

- OPCODE (one or two bytes) selects the operation (e.g., addition, subtraction, move) performed by the microprocessor



- D – direction of data flow
 - D = 0 Data flow REG -> R/M
 - D = 1 Data flow R/M -> REG
- W – data size
 - W = 0 data is byte sized
 - W = 1 data is word sized or double word sized depending on Real vs. Prot. Mode

Register assignment example

- Consider the 2-byte machine language instruction 8BECh. Assume 16-bit instruction mode.

Binary representation: **1000 1011 1110 1100**, from which we get:

OPCODE: 100010 MOV
D 1 data goes from R/M to REG
W 1 data size is 16-bits or one word
MOD 11 R/M field indicates a register
REG 101 Register BP
R/M 100 Register SP

Therefore the instruction is **MOV BP, SP**

| Code | W=0 | W=1 | W=1 |
|------|-----|-----|-----|
| 000 | AL | AX | EAX |
| 001 | CL | CX | ECX |
| 010 | DL | DX | EDX |
| 011 | BL | BX | EBX |
| 100 | AH | SP | ESP |
| 101 | CH | BP | EBP |
| 110 | DH | SI | ESI |
| 111 | BH | DI | EDI |

Use of R/M Filed in Determining Addressing Mode

- If the MOD field contains 00, 01, or 10, the R/M fields takes on a new meaning

- Examples:

- If MOD = 00 and R/M = 101 the addressing mode is [DI]
- If MOD = 01 or 10 and R/M = 101 the addressing mode is [DI+33h] or [DI+2222h] where 33h and 2222h are arbitrary displacement values

| MOD | FUNCTION |
|-----|--|
| 00 | No displacement |
| 01 | 8-bit sign-extended displacement |
| 10 | 16-bit displacement |
| 11 | R/M is a register (register addressing mode) |

| R/M Code | Function |
|----------|----------|
| 000 | DS:BX+SI |
| 001 | DS:BX+DI |
| 010 | SS:BP+SI |
| 011 | SS:BP+DI |
| 100 | DS:SI |
| 101 | DS:DI |
| 110 | SS:BP |
| 111 | DS:BX |

Example

Consider machine language instruction 8A15h

Binary representation: 1000 1010 0001 0101, from which we get:

OPCODE: 100010 MOV
D 1 data goes from R/M to REG
W 0 data byte sized
MOD 00 R/M field indicates a mem addr mode
REG 010 Register DL
R/M 101 DS:DI

Therefore the instruction is **MOV DL, [DI]**

| MOD | FUNCTION |
|-----|--|
| 00 | No displacement |
| 01 | 8-bit sign-extended displacement |
| 10 | 16-bit displacement |
| 11 | R/M is a register (register addressing mode) |

| R/M Code | Function | Code | W=0 | W=1 | W=1 |
|----------|----------|------|-----|-----|-----|
| 000 | DS:BX+SI | 000 | AL | AX | EAX |
| 001 | DS:BX+DI | 001 | CL | CX | ECX |
| 010 | SS:BP+SI | 010 | DL | DX | EDX |
| 011 | SS:BP+DI | 011 | BL | BX | EBX |
| 100 | DS:SI | 100 | AH | SP | ESP |
| 101 | DS:DI | 101 | CH | BP | EBP |
| 110 | SS:BP | 110 | DH | SI | ESI |
| 111 | DS:BX | 111 | BH | DI | EDI |

Z. Kalbarczyk

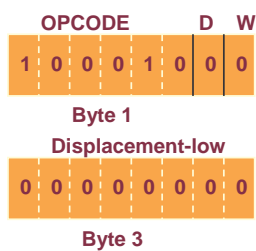
ECE390

Direct Addressing Mode

- Direct Addressing mode (for 16-bit instructions) occurs whenever memory data are referenced by only the displacement mode of addressing

MOV [1000h], DL moves the contents of DL into data segment memory location 1000h

MOV [NUMB], DL moves the contents of DL into symbolic data segment memory location NUMB



MOV [1000h], DL

Whenever the instruction has only a displacement:

MOD is always 00
R/M is always 110

Z. Kalbarczyk

ECE390

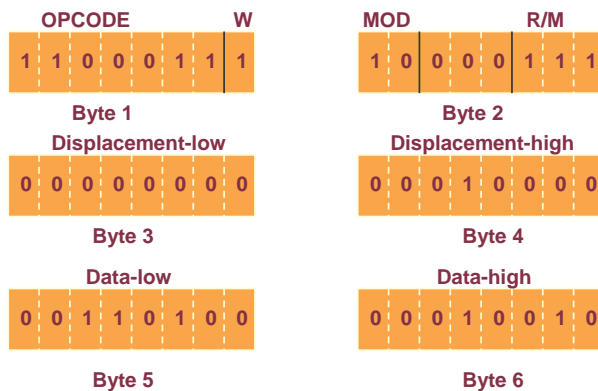
But wait...

- Doesn't this cause a problem?
 - What about **MOV DL, [BP]**?
 - No displacement, Mode 00
 - [BP] addressing mode, R/M 110
- This actually assembles as **MOV DL, [BP+0]**
 - 8-bit displacement, Mode 01
 - [BP] addressing mode, R/M 110
- Note that this makes [BP] move instructions at least three bytes long

| R/M Code | Function |
|----------|----------|
| 000 | DS:BX+SI |
| 001 | DS:BX+DI |
| 010 | SS:BP+SI |
| 011 | SS:BP+DI |
| 100 | DS:SI |
| 101 | DS:DI |
| 110 | SS:BP |
| 111 | DS:BX |

Immediate Instruction

- Consider an instruction:
MOV word [BX + 1000h], 1234h



Moves 1234h into the word-sized memory location addressed by the sum of 1000h, BX, and DS x 10h

WORD directive indicates to the assembler that the instruction uses a word-sized memory pointer (if the instruction moves a byte of immediate data, then BYTE directive is used).

The above directives are only needed when it is not clear if the operation is a byte or a word, e.g.,
MOV [BX], AL clear a byte move
MOV [BX], 1 not clear, can be byte-, word-, or double word-sized move
 should be for instance
MOV BYTE [BX], 1

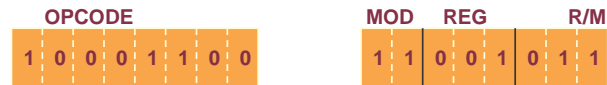
Segment MOV Instructions

- The contents of a segment register are moved by MOV, PUSH, POP
- Segment registers are selected by appropriate setting of register bits (REG field)

Code Segment Register

```
000      ES
001      CS
010      SS
011      DS
100      FS
101      GS
```

Example: MOV BX, CS



REG is 001 => selects CS
R/M is 011 => selects BX

Note: **MOV CS, ??** and **POP CS** are not allowed

Note that the opcode for this instruction is different from the prior MOV instructions

NASM Basics

```
BITS      16
;===== SECTION 1: Define constants =====
CR          EQU    0Dh            ; moves cursor to left side of screen
LF          EQU    0Ah            ; moves cursor down one line
Fmsg        DB     CR, LF, 'What did your do in class? ', CR, LF, '$'
question    DB     CR, LF, 'Input your grade', CR, LF, '$'

;===== SECTION 2: Declare external routines =====
; Declare external library routines

EXTERN kbline, dspout, dspmsg        ; imports symbols from other modules
EXTERN                                ; other routines

; Declare local routines
GLOBAL Init                            ; exports symbols to other modules

;Global variables
GLOBAL var1, var2

;===== SECTION 3: Define stack segment =====
SEGMENT stkseg STACK                 ; *** STACK SEGMENT ***
resb        64*8
stacktop:
resb        0
```

NASM Basics

```
===== SECTION 4: Define code segment =====
SEGMENT code                ; *** CODE SEGMENT ***

===== SECTION 5: Declare variables for main procedure =====a

CRLFString  DB  CR,LF,'$'
MyArray     times 26 DB 0

===== SECTION 6: Program initialization =====

..start:                    ; start of program

    mov  ax, cs              ; Initialize Default Segment register
    mov  ds, ax
    mov  ax, stkseg         ; Initialize Stack Segment register
    mov  ss, ax
    mov  sp, stacktop      ; Initialize Stack Pointer register
```

Z. Kalbarczyk

ECE390

NASM Basics

```
MAIN:
    mov  dx, question       ; puts offset of question into DX
    call dspmsg             ; LIB291 subroutine – displays message string up to '$'
    call kbdtline          ; LIB291 subroutine – get one character from keyboard input,
                          ; echo screen, put into AL

    mov  [mygrade], al     ; store character in memory mygrade
    cmp  BYTE[mygrade], 'F'
    jne  .Not_F
    mov  dx, Fmsg
    call dspmsg
    jmp  .done

.Not_F:
    mov  dl, BEL           ; ring the bell
    call dspout           ; (beep)
    ....

.done:
    call dosxit           ; end of program (exit to DOS)

=====
call  SUBR                ; call subroutine SUBR
                          ; return offset saved in Stack Segment
ret                                     ; return to caller
```

Z. Kalbarczyk

ECE390