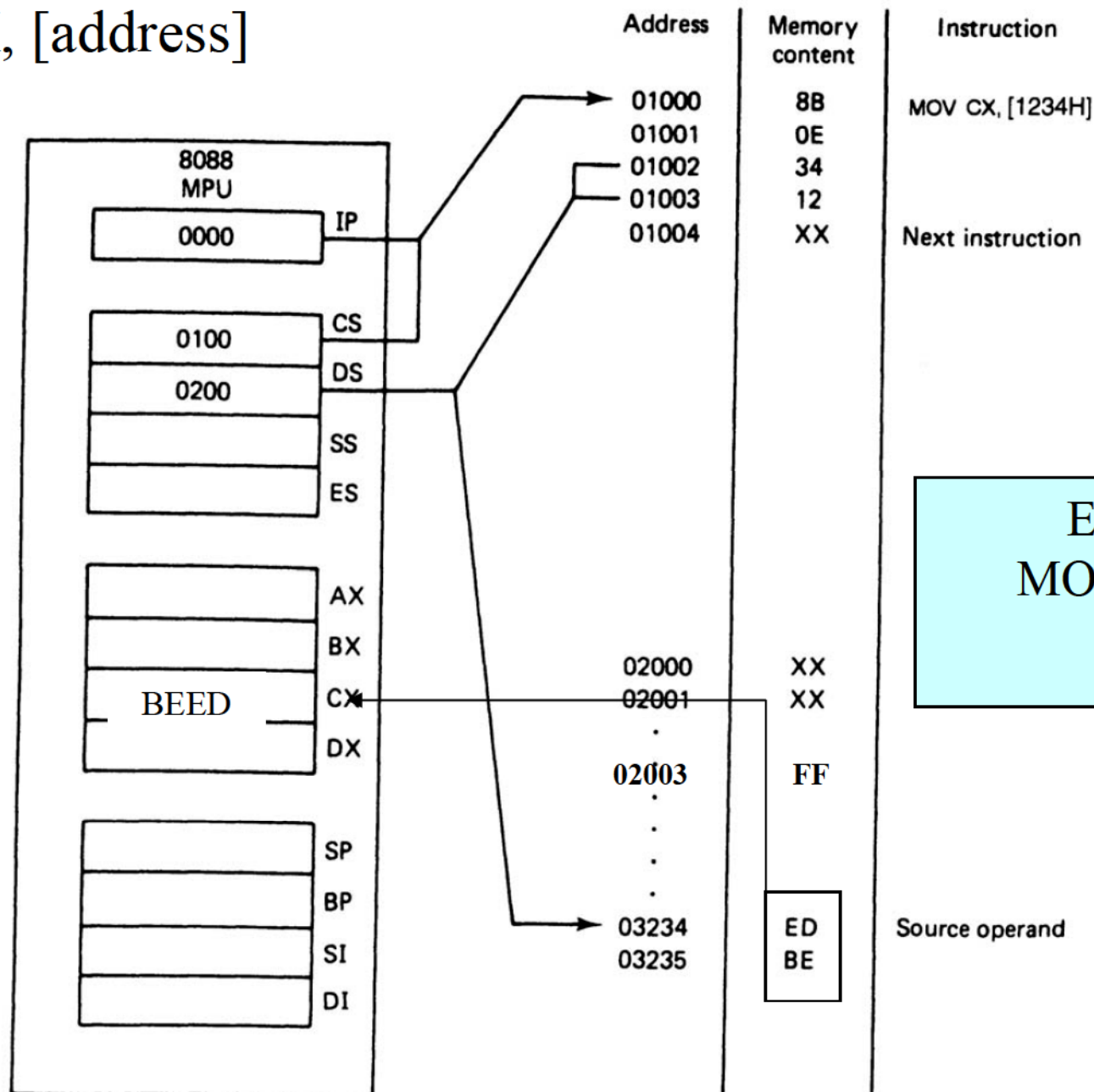


Addressing Modes

- When the 8086 executes an instruction, it performs the specified function on data
- These data, called operands,
 - May be a part of the instruction
 - May reside in one of the internal registers of the microprocessor
 - May be stored at an address in memory
- **Register Addressing Mode**
 - MOV AX, BX
 - MOV ES, AX
 - MOV AL, BH
- **Immediate Addressing Mode**
 - MOV AL, 15h
 - MOV AX, 2550h
 - MOV CX, 625

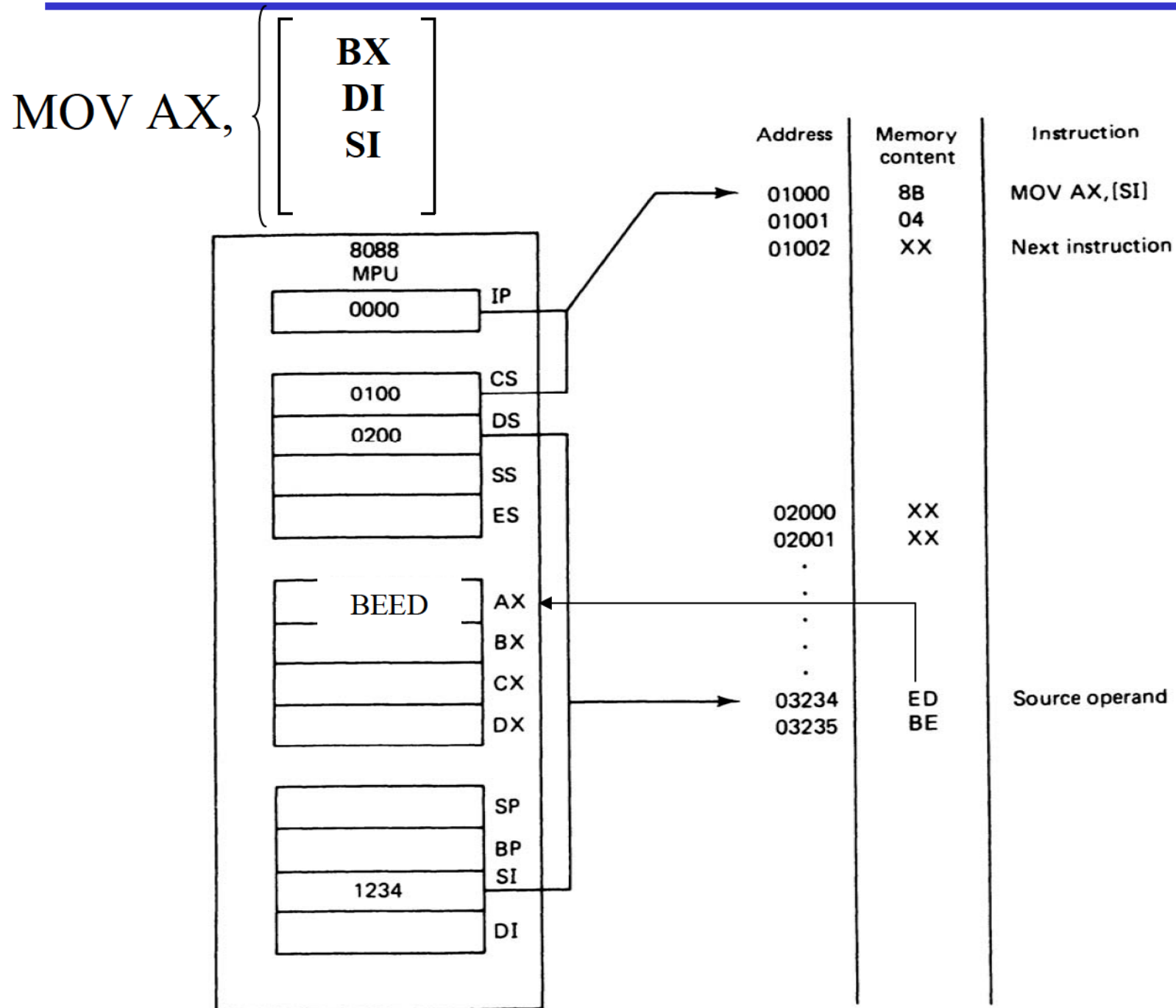
Direct Addressing Mode

MOV CX, [address]



Example:
MOV AL,[03]
AL=?

Register Indirect Addressing Mode



Example for Register Indirect Addressing

- Assume that DS=1120, SI=2498 and AX=17FE show the memory locations after the execution of:

MOV [SI],AX

DS (Shifted Left) + SI = 13698.

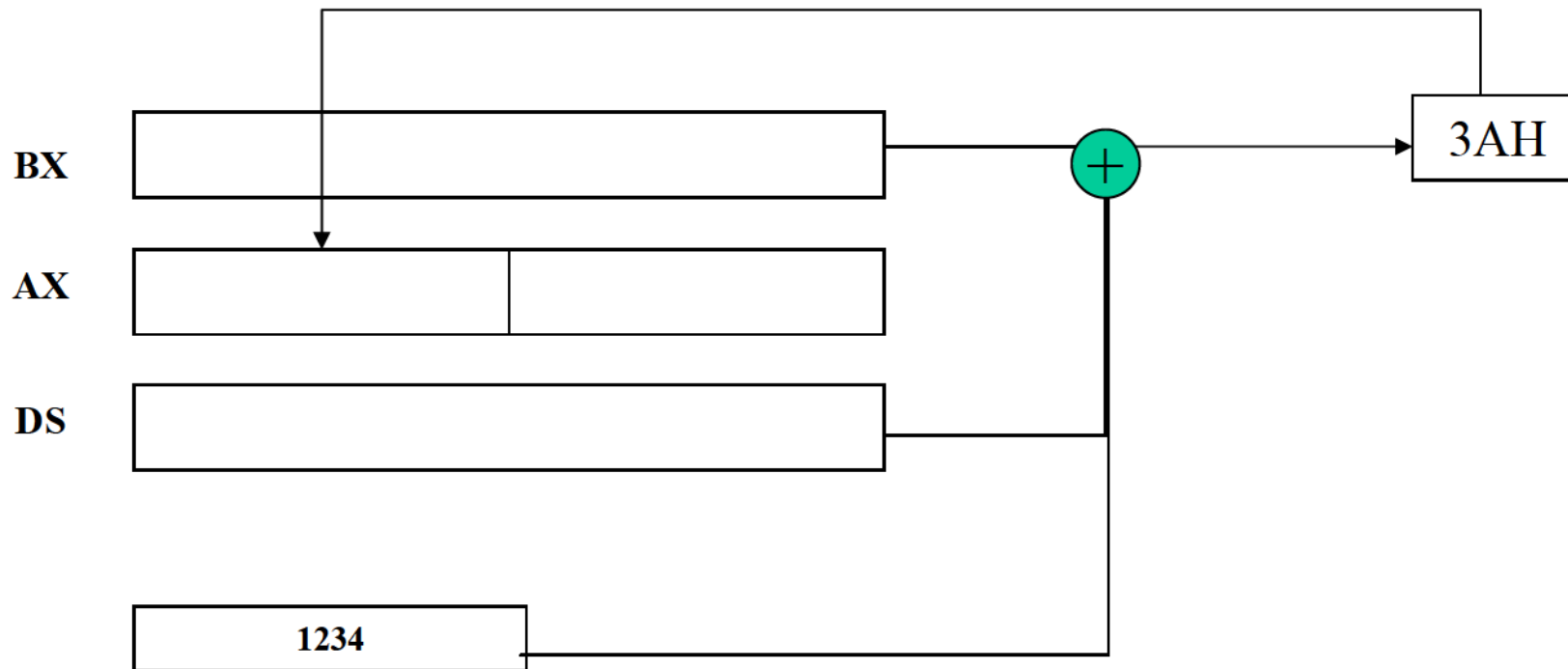
With little endian convention:

Low address 13698 → FE

High Address 13699 → 17

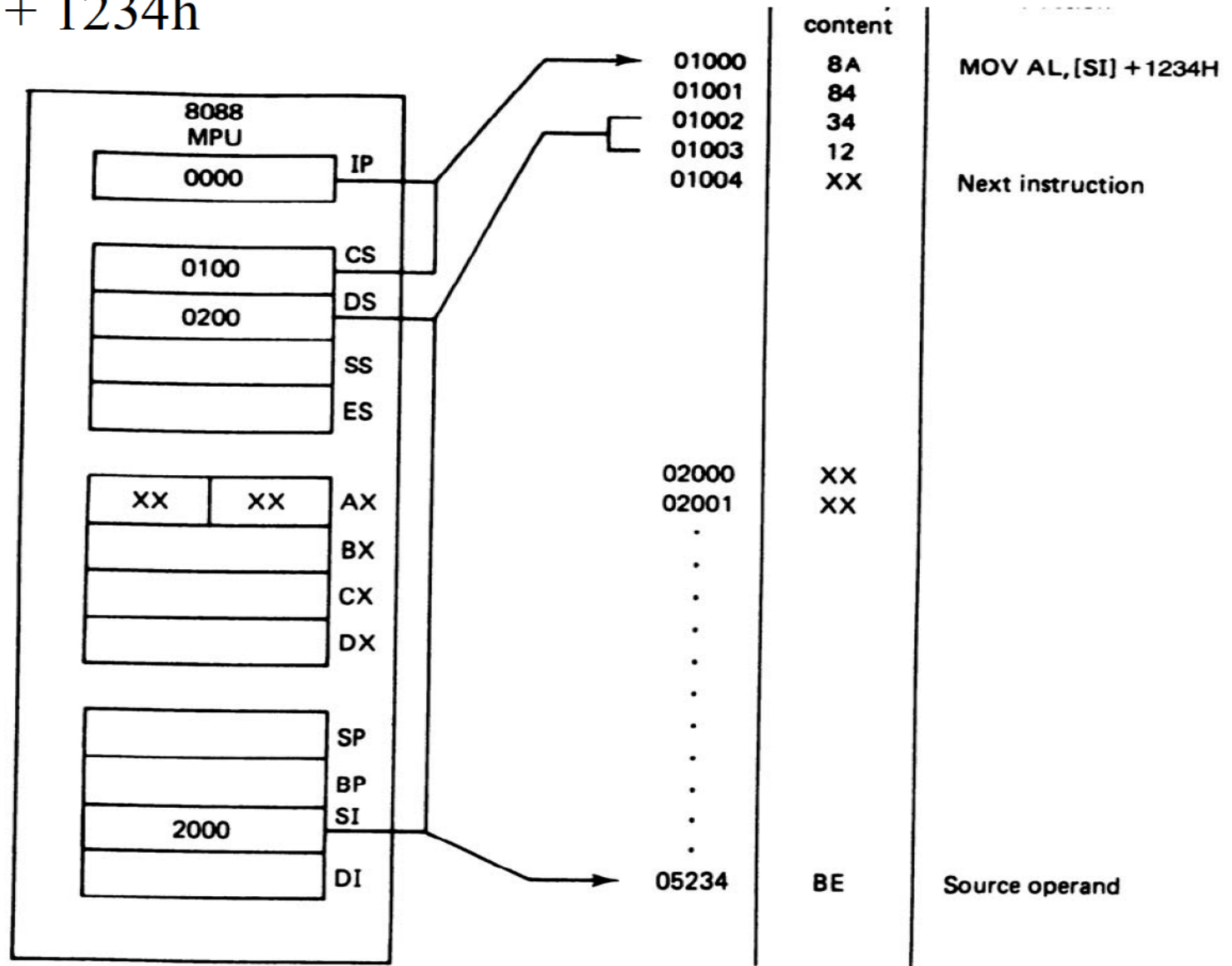
Based-Relative Addressing Mode

MOV AH, [$\begin{smallmatrix} \text{DS:BX} \\ \text{SS:BP} \end{smallmatrix}$] + 1234h



Indexed Relative Addressing Mode

MOV AH, [$\begin{smallmatrix} SI \\ DI \end{smallmatrix}$] + 1234h



Example: What is the physical address MOV [DI-8],BL if DS=200 & DI=30h ?
 DS:200 shift left once 2000 + DI + -8 = 2028

Based-Indexed Relative Addressing Mode

- Based Relative + Indexed Relative
- We must calculate the PA (physical address)

$$PA = \begin{array}{|c|} \hline CS \\ \hline SS \\ \hline DS \\ \hline ES \\ \hline \end{array} : \begin{array}{|c|} \hline BX \\ \hline BP \\ \hline \end{array} + \begin{array}{|c|} \hline SI \\ \hline DI \\ \hline \end{array} + \begin{array}{|c|} \hline 8 \text{ bit displacement} \\ \hline 16 \text{ bit displacement} \\ \hline \end{array}$$

MOV AH,[BP+SI+29]

or

MOV AH,[SI+29+BP]

or

MOV AH,[SI][BP]+29

The
register
order does
not matter

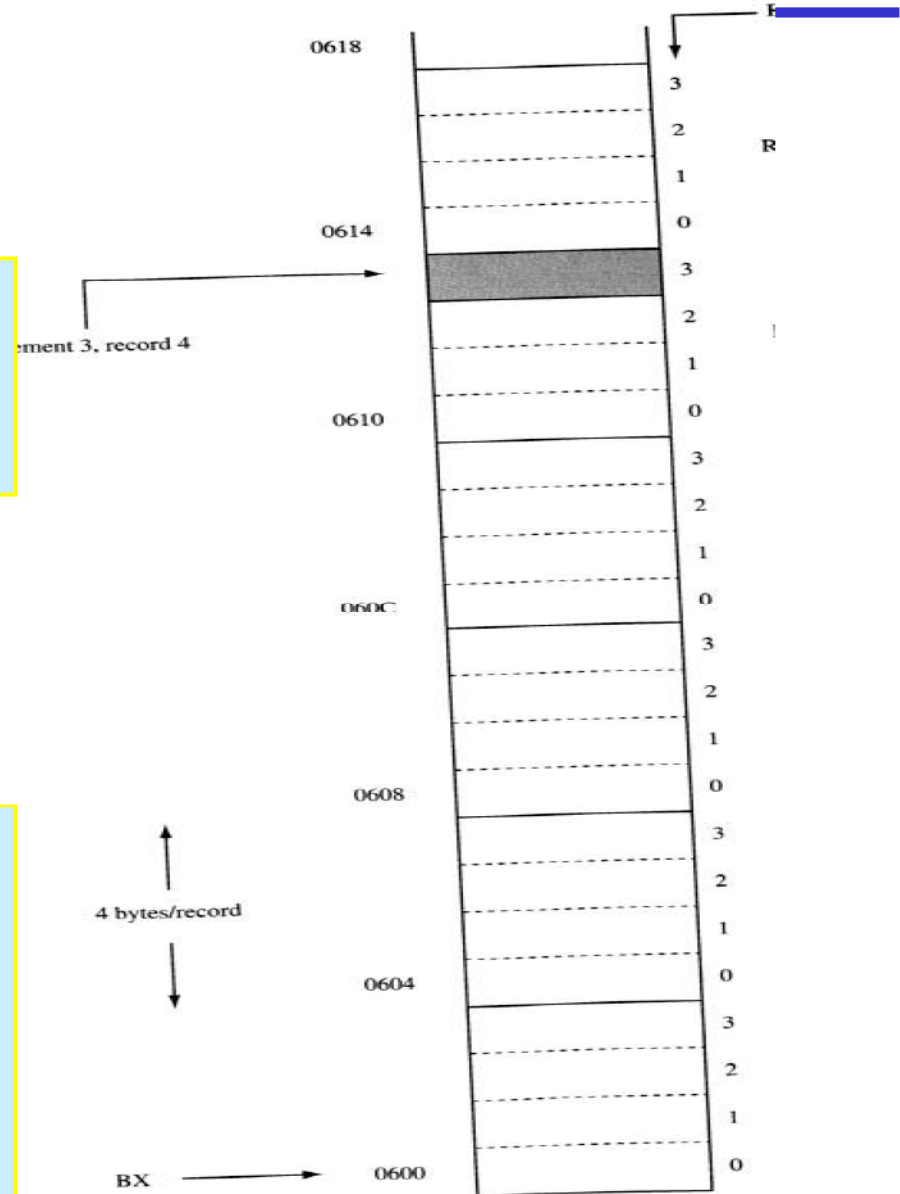
Based-Indexed Addressing Mode

Figure 4-4.
The base + index +
placement
addressing mode can
be used to access a
particular element in a
particular record of an
array.

```
MOV BX, 0600h  
MOV SI, 0010h ; 4 records, 4 elements each.  
MOV AL, [BX + SI + 3]
```

OR

```
MOV BX, 0600h  
MOV AX, 004h ;  
MOV CX, 04;  
MUL CX  
MOV SI, AX  
MOV AL, [BX + SI + 3]
```



Summary of the addressing modes

| Addressing Mode | Operand | Default Segment |
|------------------------|--|-----------------|
| Register | Reg | None |
| Immediate | Data | None |
| Direct | [offset] | DS |
| Register Indirect | [BX] [SI] [DI] | DS DS DS |
| Based Relative | [BX]+disp [BP]+disp | DS SS |
| Indexed Relative | [DI]+disp [SI]+disp | DS DS |
| Based Indexed Relative | [BX][SI or DI]+disp [BP][SI or DI]+disp | DS SS |

16 bit Segment Register Assignments

| Type of Memory Reference | Default Segment | Alternate Segment | Offset |
|--------------------------|-----------------|-------------------|-----------------|
| Instruction Fetch | CS | none | IP |
| Stack Operations | SS | none | SP,BP |
| General Data | DS | CS,ES,SS | BX, address |
| String Source | DS | CS,ES,SS | SI, DI, address |
| String Destination | ES | None | DI |

Segment override

| Segment Registers | CS | DS | ES | SS |
|-------------------|----|----------|----------|-------|
| Offset Register | IP | SI,DI,BX | SI,DI,BX | SP,BP |

| Instruction Examples | Override Segment Used | Default Segment |
|-----------------------|-----------------------|-----------------|
| MOV AX,CS:[BP] | CS:BP | SS:BP |
| MOV DX,SS:[SI] | SS:SI | DS:SI |
| MOV AX,DS:[BP] | DS:BP | SS:BP |
| MOV CX,ES:[BX]+12 | ES:BX+12 | DS:BX+12 |
| MOV SS:[BX][DI]+32,AX | SS:BX+DI+32 | DS:BX+DI+32 |

Example for default segments

- The following registers are used as offsets. Assuming that the default segment used to get the logical address, give the segment register associated?

a) BP b)DI c)IP d)SI, e)SP, f) BX

- Show the contents of the related memory locations after the execution of this instruction

MOV [BP][SI]+10,DX

if DS=2000, SS=3000,CS=1000,SI=4000,BP=7000,DX=1299 (all hex)

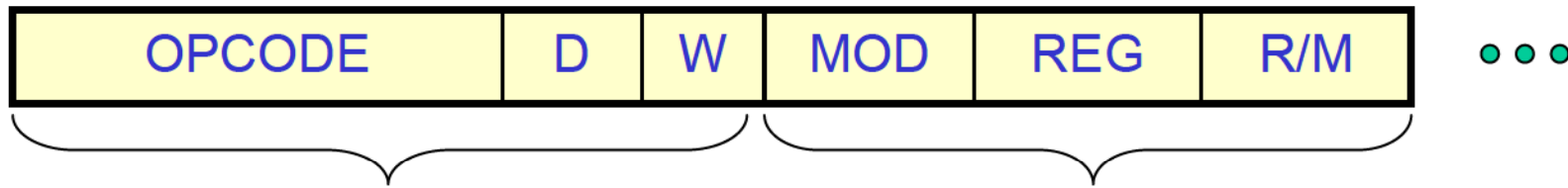
Assembly Language

- There is a one-to-one relationship between assembly and machine language instructions
- What is found is that a compiled machine code implementation of a program written in a high-level language results in inefficient code
 - More machine language instructions than an assembled version of an equivalent handwritten assembly language program
- Two key benefits of assembly language programming
 - It takes up less memory
 - It executes much faster

Languages in terms of applications

- One of the most beneficial uses of assembly language programming is **real-time applications**.
- Real time means the task required by the application must be completed before any other input to the program that will alter its operation can occur
- For example the device service routine which controls the operation of the floppy disk drive is a good example that is usually written in assembly language
- Assembly language not only good for controlling hardware devices but also **performing pure software operations**
 - searching through a large table of data for a special string of characters
 - Code translation from ASCII to EBCDIC
 - Table sort routines
 - Mathematical routines
- Assembly language: perform real-time operations
- High-level languages: Those operations mostly not critical in time.

Converting Assembly Language Instructions to Machine Code



- An instruction can be coded with 1 to 6 bytes
- **Byte 1 contains three kinds of information:**
 - Opcode field (6 bits) specifies the operation such as add, subtract, or move
 - Register Direction Bit (D bit)
 - Tells the register operand in REG field in byte 2 is source or destination operand
 - 1: Data flow to the REG field from R/M
 - 0: Data flow from the REG field to the R/M
 - Data Size Bit (W bit)
 - Specifies whether the operation will be performed on 8-bit or 16-bit data
 - 0: 8 bits
 - 1: 16 bits
- **Byte 2 has two fields:**
 - Mode field (MOD) – 2 bits
 - Register field (REG) - 3 bits
 - Register/memory field (R/M field) – 2 bits

Continued

- REG field is used to identify the register for the first operand

| REG | W = 0 | W = 1 |
|-----|-------|-------|
| 000 | AL | AX |
| 001 | CL | CX |
| 010 | DL | DX |
| 011 | BL | BX |
| 100 | AH | SP |
| 101 | CH | BP |
| 110 | DH | SI |
| 111 | BH | DI |

Continued

- 2-bit MOD field and 3-bit R/M field together specify the second operand

| CODE | EXPLANATION |
|------|--|
| 00 | Memory Mode, no displacement follows* |
| 01 | Memory Mode, 8-bit displacement follows |
| 10 | Memory Mode, 16-bit displacement follows |
| 11 | Register Mode (no displacement) |

*Except when R/M = 110, then 16-bit displacement follows

(a)

| MOD = 11 | | | EFFECTIVE ADDRESS CALCULATION | | | |
|----------|-------|-------|-------------------------------|----------------|------------------|-------------------|
| R/M | W = 0 | W = 1 | R/M | MOD = 00 | MOD = 01 | MOD = 10 |
| 000 | AL | AX | 000 | (BX) + (SI) | (BX) + (SI) + D8 | (BX) + (SI) + D16 |
| 001 | CL | CX | 001 | (BX) + (DI) | (BX) + (DI) + D8 | (BX) + (DI) + D16 |
| 010 | DL | DX | 010 | (BP) + (SI) | (BP) + (SI) + D8 | (BP) + (SI) + D16 |
| 011 | BL | BX | 011 | (BP) + (DI) | (BP) + (DI) + D8 | (BP) + (DI) + D16 |
| 100 | AH | SP | 100 | (SI) | (SI) + D8 | (SI) + D16 |
| 101 | CH | BP | 101 | (DI) | (DI) + D8 | (DI) + D16 |
| 110 | DH | SI | 110 | DIRECT ADDRESS | (BP) + D8 | (BP) + D16 |
| 111 | BH | DI | 111 | (BX) | (BX) + D8 | (BX) + D16 |

(b)

Examples

- MOV BL,AL
- Opcode for MOV = 100010
- We'll encode AL so
 - D = 0 (AL source operand)
- W bit = 0 (8-bits)
- MOD = 11 (register mode)
- REG = 000 (code for AL)
- R/M = 011

| OPCODE | D | W | MOD | REG | R/M |
|--------|---|---|-----|-----|-----|
| 100010 | 0 | 0 | 11 | 000 | 011 |

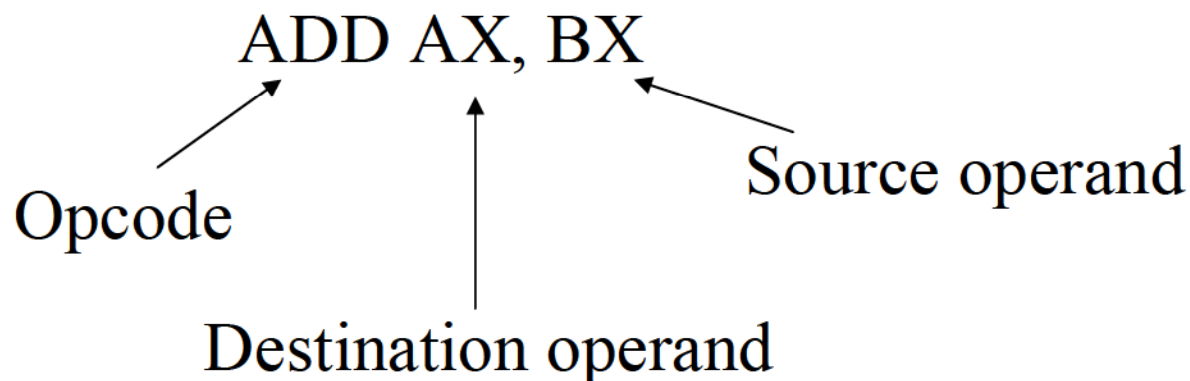
MOV BL,AL => 10001000 11000011 = 88 C3h

ADD AX,[SI] => 00000011 00000100 = 03 04 h

ADD [BX][DI] + 1234h, AX => 00000001 10000001 ____ h
=> 01 81 34 12 h

Software

- The sequence of commands used to tell a microcomputer what to do is called a **program**
- Each command in a program is called an **instruction**
- 8088 understands and performs operations for **117 basic instructions**
- The native language of the **IBM PC** is the machine language of the 8088
- A program written in machine code is referred to as **machine code**
- In 8088 assembly language, each of the operations is described by alphanumeric symbols instead of just 0s or 1s.



Instructions

[LABEL:] MNEMONIC [OPERANDS] [; COMMENT]

↑
Address identifier
Max 31 characters
: indicates it opcode
generating instruction

↑
Instruction

↑
Does not generate any machine code

Ex. START: MOV AX,BX ; copy BX into AX

DEBUG program instruction set (page 825 mzd)

- Debug instructions
- List of commands
 - a Assemble [address] you can type in code this way
 - c range address ; compare c 100 105 200
 - d [range] ; Dump d 150 15A
 - e address [list] ; Enter e 100
 - f Fill range list F 100 500 ‘ ‘
 - g Go [=address] addresses runs the program
 - h Value1 Value2 ; addition and subtraction H 1A 10
 - i Input port I 3F8
 - r Show & change registers Appears to show the same thing as t, but doesn't cause any code to be executed.
 - t Trace either from the starting address or current location.
 - u UnAssemble

Some examples with debug

0100 mov ax,24b6

0103 mov di, 85c2

0106 mov dx,5f93

0109 mov sp,1236

010c push ax

010d push di

010e int 3

Display the stack contents after execution.

-D 1230 123F

Some examples with DEBUG

- 0100 mov al,9c
- 0102 mov dh,64
- 0104 add al,dh
- 0109 int 3

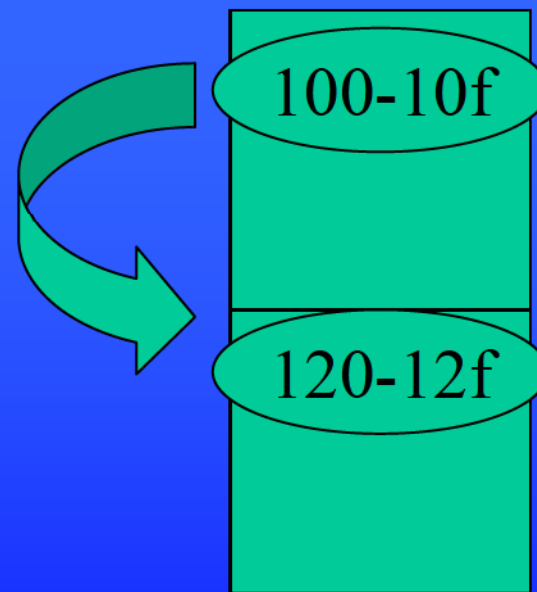
trace these three commands and observe the flags

- After the code has been entered with the A command
- Use CX to store data indicating number of bytes to save. BX is the high word.
- Use N filename.com
- Then W command to write to file.
- L loads this file.

Example

Copy the contents of a block of memory (16 bytes) starting at location 20100h to another block of memory starting at 20120h

```
MOV AX,2000
MOV DS,AX
MOV SI, 100
MOV DI, 120
MOV CX, 10
NXTPT: MOV AH, [SI]
        MOV [DI], AH
        INC SI
        INC DI
        DEC CX
        JNZ NXTPT
```



Assembler Directives

.MODEL SMALL ; selects the size of the memory model usually sufficient
max 64K code 64K data

.STACK ; beginning of the stack segment

.DATA ; beginning of the data segment

.CODE ; beginning of the code segment

Ex: .DATA

DATAW DW 213FH

DATA1 DB 52H

SUM DB ? ; nothing stored but a storage is assigned

Ex: .CODE

PROGRAMNAME PROC; Every program needs a name

.... ; program statements

PROGRAMNAME ENDP

END PROGRAMNAME

Sample Program

```
title Hello World Program                (hello.asm)
; This program displays "Hello, world!"
.model small
.stack 100h
.data
message db "Hello, world!",0dh,0ah,'$' ;newline+eoc
.code
main proc
    mov ax,@data ; address of data
    mov ds,ax
    mov ah,9
    mov dx,offset message ;disp.msg.starting at 0
    int 21h ; or LEA dx,message will do!
    mov ax,4C00h ; halt the program and return
    int 21h
main endp
end main
```

The PTR Operator

- `INC [20h]` ; is this byte/word/dword? or
- `MOV [SI],5`
 - Is this byte 05?
 - Is this word 0005?
 - Or is it double word 00000005?
- Byte or word or doubleword?
- To clarify we use the PTR operator
 - `INC BYTE PTR [20h]`
 - `INC WORD PTR [20h]`
 - `INC DWORD PTR [20h]`
- or for the mov example:
 - `MOV byte ptr [SI],5`
 - `MOV word ptr[SI],5`

The PTR Operator

- Would we need to use the PTR operator in each of the following?

```
MOV AL,BVAL  
MOV DL,[BX]  
SUB [BX],2  
MOV CL,WVAL  
ADD AL,BVAL+1
```

```
.data  
BVAL DB 10H,20H  
WVAL DW 1000H
```

```
MOV AL,BVAL  
MOV DL,byte ptr [BX]  
SUB [BX],byte ptr 2  
MOV CL,byte ptr WVAL  
ADD AL,BVAL+1
```