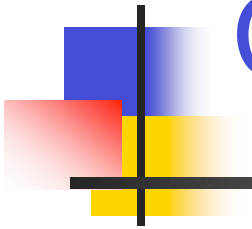
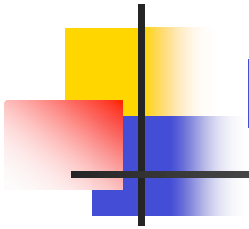


Chapter 3

Assembly Language Programming



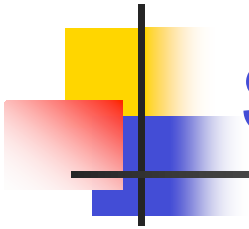
Introduction

3.2 Software: The Microcomputer Program

3.3 Assembly Language Program Development on the IBM-Compatible PC/AT

3.4 The 80386DX Microprocessor Instruction Set

3.5 Addressing Modes of the 80386DX Microprocessor

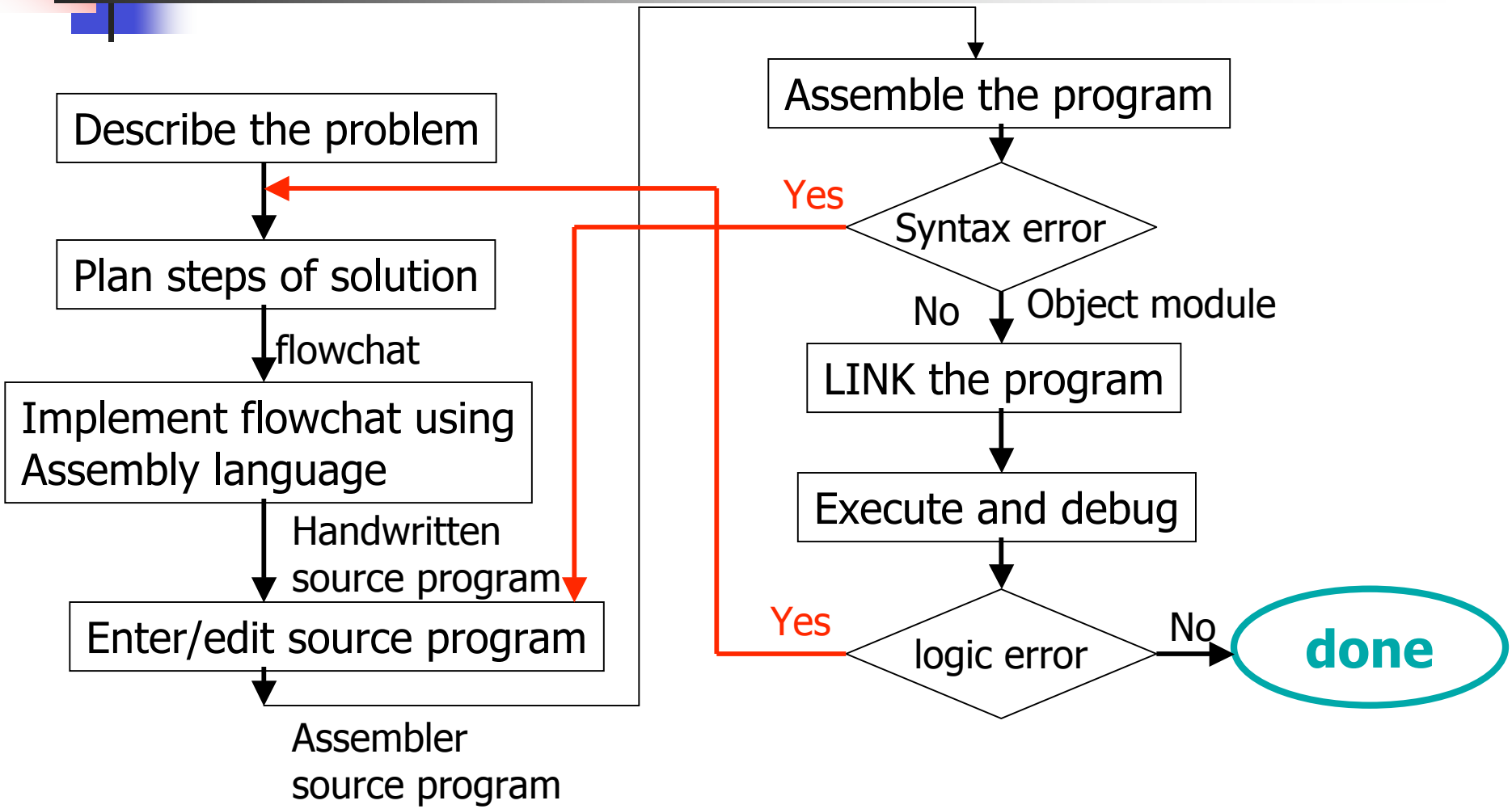


Software

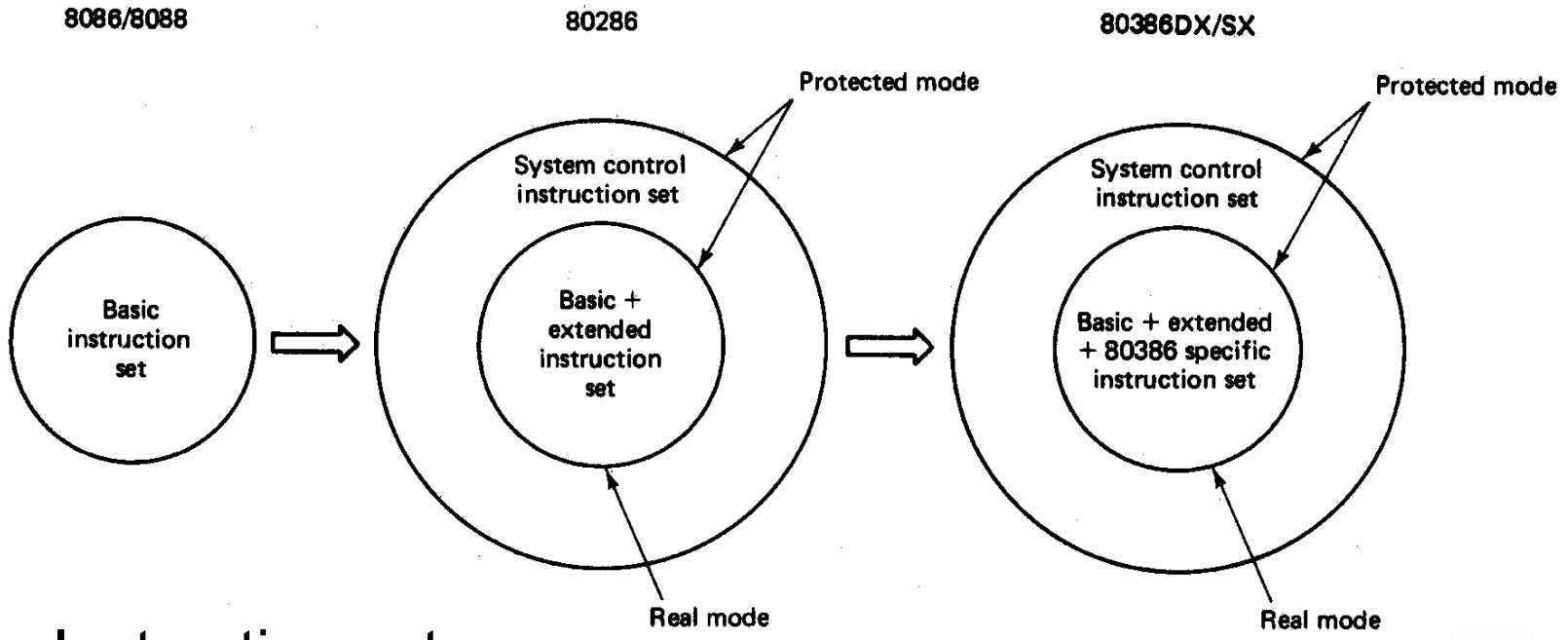
- Instruction -> Program -> Software
- Machine language -> Assembly Language -> High Level Language (C/C++, Java)
- Source code -> Object code -> Executable
- Assembly language
  - Instruction: `Label: Instruction ; comment`

    Example: `START: MOV EAX, EBX; COPY EBX INTO EAX`
  - List file: line number, offset, machine language (code), instruction, comments

Assembly Language Program Development



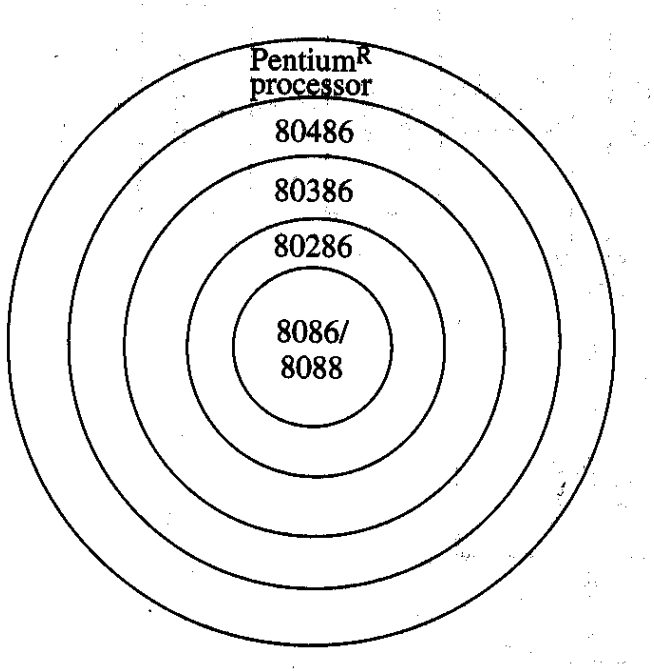
The 80386DX Base Instruction Set



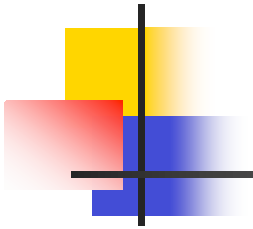
Instruction set

- Defines the basic operations a programmer can make the microprocessor perform
- 8088/8086 instruction set contains 117 basic instructions

Instruction Set Compatibility

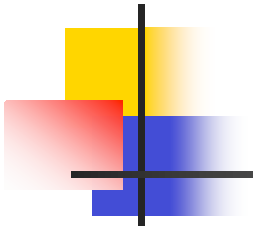


- The 80x86 instruction set has evolved in an upward compatible manner
  - Base instruction set → 8088/8086 processor
  - Extended instruction set → 80286 processor
  - System control instruction set → 80286 processor
  - 80386 specific instruction set → 80386DX/SX
  - 80486 specific instruction set → 80486DX/SX
  - Pentium specific instruction set → Original Pentium processor



Instruction Groups

- **Instruction groups**
  - **Instructions are organized into groups of functionally related instructions**
    - **Data Transfer instructions**
    - **Input/output instructions**
    - **Arithmetic instructions**
    - **Logic instructions**
    - **String Instructions**
    - **Control transfer instructions**
    - **Processor control**



Instruction Assembly Notation

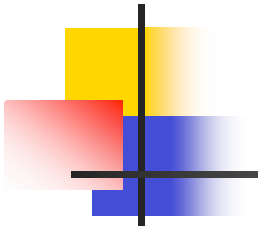
- **Each instruction is represented by a mnemonic that describes its operation—called its operation code (opcode)**
  - **MOV = move → data transfer**
  - **ADD = add → arithmetic**
  - **AND = logical AND → logic**
  - **JMP = unconditional jump → control transfer**
- **Operands are the other parts of an assembly language Instructions**
  - **Identify whether the elements of data to be processed are in registers or memory**
    - **Source operand– location of one operand to be process**
    - **Destination operand—location of the other operand to be processed and the location of the result**

Machine Language

DATA TRANSFER

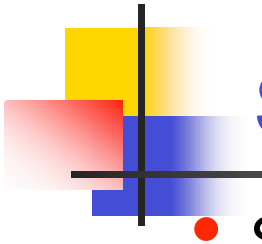
MOV = Move:	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Register/memory to/from register	1 0 0 0 1 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate to register/memory	1 1 0 0 0 1 1 w	mod 0 0 0 r/m	(DISP-LO)	(DISP-HI)	data	data if w = 1
Immediate to register	1 0 1 1 w reg	data	data if w = 1			
Memory to accumulator	1 0 1 0 0 0 0 w	addr-lo	addr-hi			
Accumulator to memory	1 0 1 0 0 0 1 w	addr-lo	addr-hi			
Register/memory to segment register	1 0 0 0 1 1 1 0	mod 0 SR r/m	(DISP-LO)	(DISP-HI)		
Segment register to register/memory	1 0 0 0 1 1 0 0	mod 0 SR r/m	(DISP-LO)	(DISP-HI)		

- Native language of the 8088/8086 (PC) is machine language (code)
  - One to one correspondence to assembly language statements
  - Instructions encoded with 0's and 1's
  - Machine instructions can take up from 1 to 6 bytes
  - Example: Move=MOV
    - The wide choice of register operands, memory operands, and addressing mode available to access operands in memory expands the move instruction to 28 different forms
    - Ranges in size from 2 to 6 bytes



Software and the Program

- Microcomputer is a general computation resource
  - Has the ability to process data, but does not know how the data is to be processed
  - Must be told:
    - Where to get information (data)?
    - What to do with the information?
    - Where to put the results?
  - This is the job of the program
- Program: sequence of instructions that tells the computer what to do
  - Simple program—few instructions
  - Complex program—100Ks to millions of instructions
  - Microcomputer fetches and executes one instruction after the other
  - Instructions guide the uC step by step through the task that is to be performed



Software and the Program

- System software—a group of programs that enable the microcomputer to operate
  - The operating system (OS)
    - Windows98
    - Windows2000
- Application programs—a collection of programs installed on the microcomputer for use by the operator.
  - Word
  - Excel
  - PowerPoint

Structure of an Assembly Language Statement

- General structure of an assembly language statement

```
LABEL:      INSTRUCTION        ;COMMENT
```

  - Label—address identifier for the statement
  - Instruction—the operation to be performed
  - Comment—documents the purpose of the statement
  - Example:

```
START:    MOV   AX, BX     ; Copy BX into AX
```

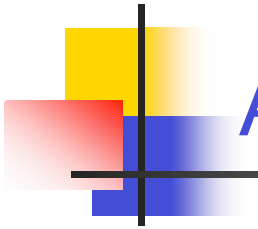
- Other examples:

```
INC SI      ;Update pointer
ADD  AX, BX
```

  - Few instructions have a label—usually marks a jump to point
  - Not all instructions need a comment

Source Program

```
TITLE   BLOCK-MOVE PROGRAM

        PAGE    ,132

COMMENT *This program moves a block of specified number of bytes
        from one place to another place*


;Define constants used in this program

        N       =       16              ;Bytes to be moved
        BLK1ADDR=       100H            ;Source block offset address
        BLK2ADDR=       120H            ;Destination block offset addr
        DATASEGADDR=    1020H           ;Data segment start address


STACK_SEG       SEGMENT         STACK 'STACK'
                DB              64 DUP(?)
STACK_SEG       ENDS


CODE_SEG        SEGMENT         'CODE'
BLOCK           PROC            FAR
        ASSUME  CS:CODE_SEG,SS:STACK_SEG

;To return to DEBUG program put return address on the stack

        PUSH    DS
        MOV     AX, 0
        PUSH    AX

;Set up the data segment address

        MOV     AX, DATASEGADDR
        MOV     DS, AX

;Set up the source and destination offset addresses

        MOV     SI, BLK1ADDR
        MOV     DI, BLK2ADDR

;Set up the count of bytes to be moved

        MOV     CX, N

;Copy source block to destination block

NXTPT:  MOV     AH, [SI]                ;Move a byte
        MOV     [DI], AH
        INC     SI                      ;Update pointers
        INC     DI
        DEC     CX                      ;Update byte counter
        JNZ     NXTPT                   ;Repeat for next byte
        RET                             ;Return to DEBUG program
BLOCK           ENDP
CODE_SEG        ENDS
        END     BLOCK                   ;End of program
```

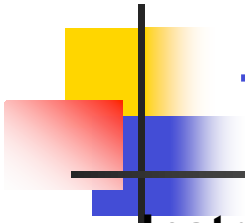


Assembler and the source program

- Assembly language program
  - Assembly language program (.asm) file—known as source code
  - Converted to machine code by a process called assembling
  - Assembling performed by a software program—an 80x86 assembler
  - Machine (object) code that can be run is output in the executable (.exe) file
  - Source listing output in (.lst) file—printed and used during execution and debugging of program
- DEBUG—part of disk operating system (DOS) of the PC
  - Permits programs to be assembled and disassembled
  - Line-by-line assembler
  - Also permits program to be run and tested

The Listing File

```
Microsoft (R) Macro Assembler Version 5.10                  5/17/92 18:10:04
BLOCK-MOVE PROGRAM                                          Page     1-1


                        TITLE BLOCK-MOVE PROGRAM

                              PAGE        ,132
                        COMMENT *This program moves a block of specified number of bytes
                                 from one place to another place*


                        ;Define constants used in this program

                             N=          16          ;Bytes to be moved
                             BLK1ADDR=   100H        ;Source block offset address
                             BLK2ADDR=   120H        ;Destination block offset addr
                             DATASEGADDR=1020H       ;Data segment start address


                             STACK_SEG   SEGMENT     STACK 'STACK'
0000  0040[                              DB          64 DUP(?)
      ??
                        ]

0040                         STACK_SEG   ENDS


0000                         CODE_SEG    SEGMENT     'CODE'
0000                         BLOCK         PROC      FAR
                             ASSUME      CS:CODE_SEG,SS:STACK_SEG

                        ;To return to DEBUG program put return address on the stack

0000  1E                     PUSH  DS
0001  B8 0000                MOV   AX, 0
0004  50                     PUSH  AX

                        ;Setup the data segment address

0005  B8 1020                MOV   AX, DATASEGADDR
0008  8E D8                  MOV   DS, AX

                        ;Setup the source and destination offset adresses

000A  BE 0100                MOV   SI, BLK1ADDR
000D  BF 0120                MOV   DI, BLK2ADDR

                        ;Setup the count of bytes to be moved

0010  B9 0010                MOV   CX, N

                        ;Copy source block to destination block

0013  8A 24             NXTPT:MOV   AH, [SI]                ;Move a byte
0015  88 25                   MOV   [DI], AH
0017  46                      INC   SI                      ;Update pointers
0018  47                      INC   DI
0019  49                      DEC   CX                      ;Update byte counter
001A  75 F7                   JNZ   NXTPT                   ;Repeat for next byte
001C  CB                      RET                           ;Return to DEBUG program
001D                          BLOCK       ENDP
001D                          CODE_SEG    ENDS
                              END         BLOCK             ;End of program
```



The Listing File

Instruction statements—operations to be performed by the program

- **Example—line 53**

```
0013 8A 24 NXTPT: MOV AH, [SI] ;Move a byte
```

Where:

0013 = offset address of first byte of code in the current CS

8A24 = machine code of the instruction

NXTPT: = Label

MOV = instruction mnemonic

AH = destination operand—a register

[SI] = source operand—in memory

;Move xxxxx = comment

- **Directives—provides directions to the assembler program**
  - **Example—line 20**

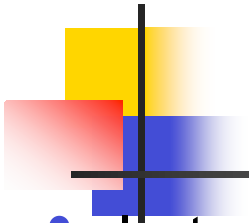
0000 0040 DB 64 DUP(?)

Defines and leaves un-initialized a block of 64 bytes in memory for the stack

More Information in the Listing

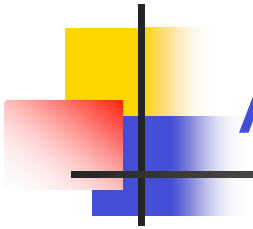
```
Segments and Groups:

                N a m e                 Length   Align   Combine Class

CODE_SEG  . . . . . . . . . . . .  001D  PARA  NONE  'CODE'
STACK_SEG . . . . . . . . . . . .  0040  PARA  STACK 'STACK'

Symbols:

                N a m e                 Type   Value   Attr

BLK1ADDR  . . . . . . . . . . . .  NUMBER  0100
BLK2ADDR  . . . . . . . . . . . .  NUMBER  0120
BLOCK . . . . . . . . . . . . . .  F PROC  0000  CODE_SEG          Length = 001D

DATASEGADDR . . . . . . . . . . .  NUMBER  1020

N . . . . . . . . . . . . . . . .  NUMBER  0010
NXTPT . . . . . . . . . . . . . .  L NEAR  0013  CODE_SEG

@CPU  . . . . . . . . . . . . . .  TEXT  0101h
@FILENAME . . . . . . . . . . . .  TEXT  block
@VERSION  . . . . . . . . . . . .  TEXT  510


     59 Source  Lines
     59 Total   Lines
     15 Symbols

47222 + 347542 Bytes symbol space free

     0 Warning Errors
     0 Severe Errors

               (b)
```

- Other information provided in the listing
  - Size of code segment and stack
  - Names, types, and values of constants and variables
  - # lines and symbols used in the program
  - # errors that occurred during assembly



Addressing Modes

- Instructions perform the operation they specify on elements of data that are called its operand
- Types of operands
  - Source operand
  - Destination operand
  - Content of source operand combined with content of destination operand → Result saved in destination operand location
- Operands may be
  - Part of the instruction—source operand only
  - Held in one of the internal registers—both source and destination operands
  - Stored at an address in memory—either the source or destination operand
  - Held in an input/output port—either the source or destination operand



Addressing Modes

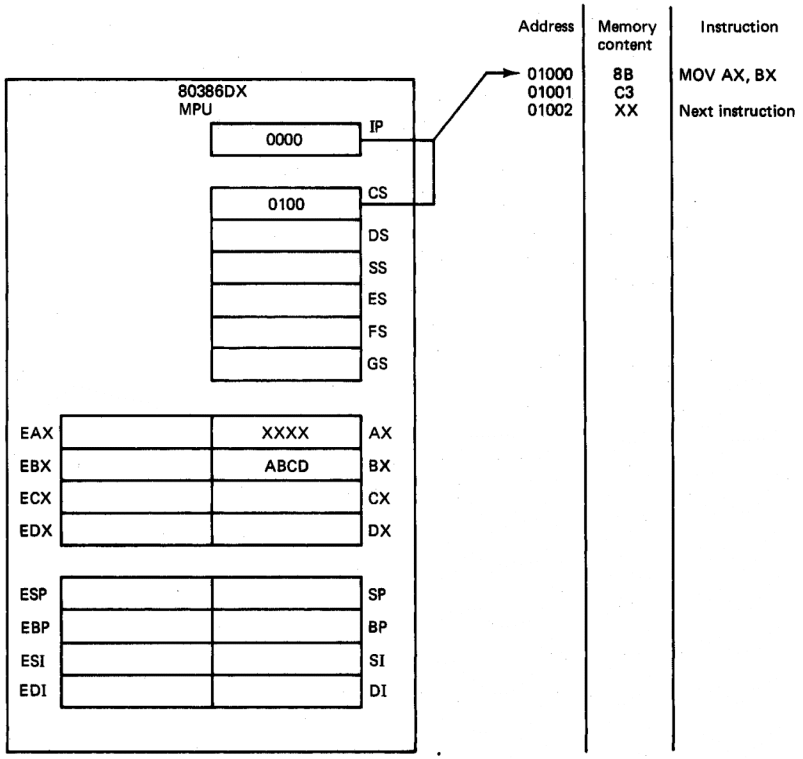
- Types of addressing modes
  - Register addressing modes
  - Immediate operand addressing
  - Memory operand addressing
- Each operand can use a different addressing mode

Register Operand Addressing Mode

Register	Operand size		
	Byte (Reg8)	Word (Reg16)	Double word (Reg32)
Accumulator	AL, AH	AX	EAX
Base	BL, BH	BX	EBX
Count	CL, CH	CX	ECX
Data	DL, DH	DX	EDX
Stack pointer	–	SP	ESP
Base pointer	–	BP	EBP
Source index	–	SI	ESI
Destination index	–	DI	EDI
Code segment	–	CS	–
Data segment	–	DS	–
Stack segment	–	SS	–
E data segment	–	ES	–
F data segment	–	FS	–
G data segment	–	GS	–

- Register addressing mode operands
  - Source operand and destination operands are both held in internal registers of the 80386DX/SX
  - Only the data registers can be accessed as bytes, words, or double words
    - Ex. AL,AH → bytes
    - AX → word
    - EAX → double word
  - Index and pointer registers as words or double words
    - Ex. SI → word pointer
    - ESI → double word pointer
  - Segment registers only as words
    - Ex. DS → word pointer

Register Operand Addressing Mode



- Example

  MOV AX,BX

  Source = BX → word data

  Destination = AX → word data

  Operation: (BX) → (AX)

- State before fetch and execution

  CS:IP = 0100:0000 = 01000H

  Move instruction code = 8BC3H

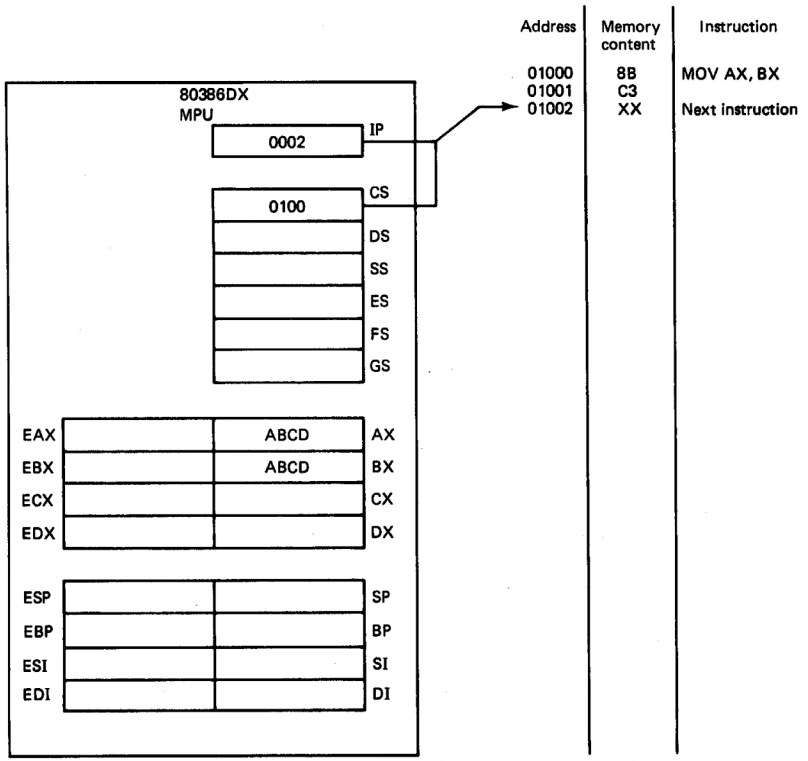
  (01000H) = 8BH

  (01001H) = C3H

  (BX) = ABCDH

  (AX) = XXXX → don't care state

Register Operand Addressing Mode



- Example (continued)
- State after execution

  CS:IP = 0100:0002 = 01002H

  01002H → points to next sequential instruction

  (BX) = ABCDH

  (AX) = ABCDH → Value in BX copied into AX

  Rest of the bits in EAX unaffected

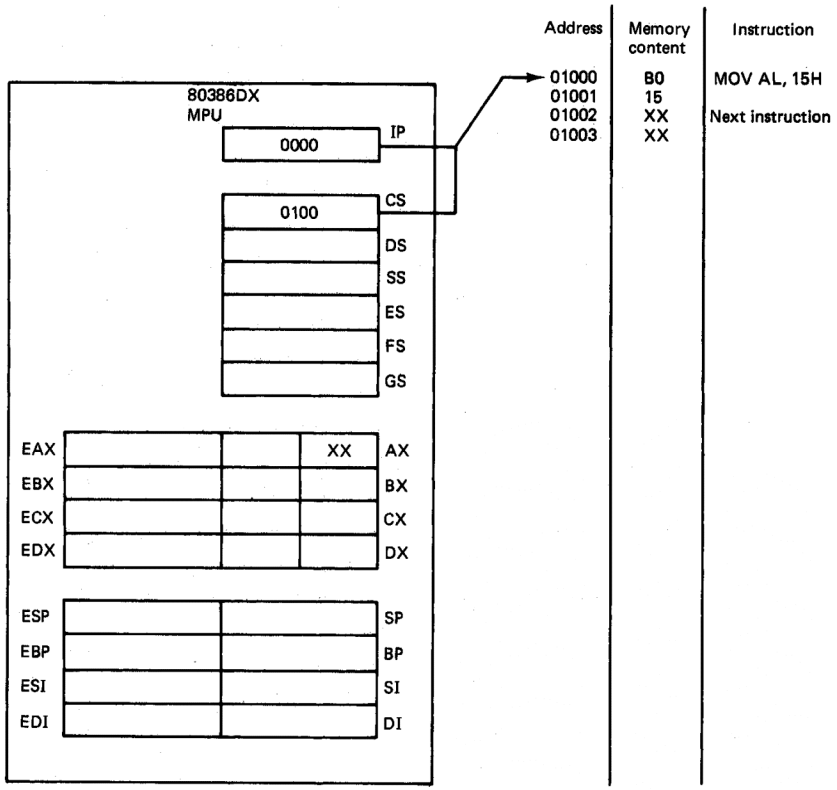
Immediate Operand Addressing Mode

- Immediate operand
  - Operand is coded as part of the instruction
  - Applies only to the source operand
  - Destination operand uses register addressing mode or a memory addressing mode
- Types
  - Imm8 = 8-bit immediate operand
  - Imm16 = 16-bit immediate operand
  - Imm32 = 32-bit immediate operand
- General instruction structure and operation

  MOV Rx,ImmX

  ImmX → (Rx)

Immediate Operand Addressing Mode



- Example

  MOV AL,15H

  Source = Imm8 → immediate byte data

  Destination = AL → Byte of data

  Operation: (Imm8) → (AL)

- State before fetch and execution

  CS:IP = 0100:0000 = 01000H

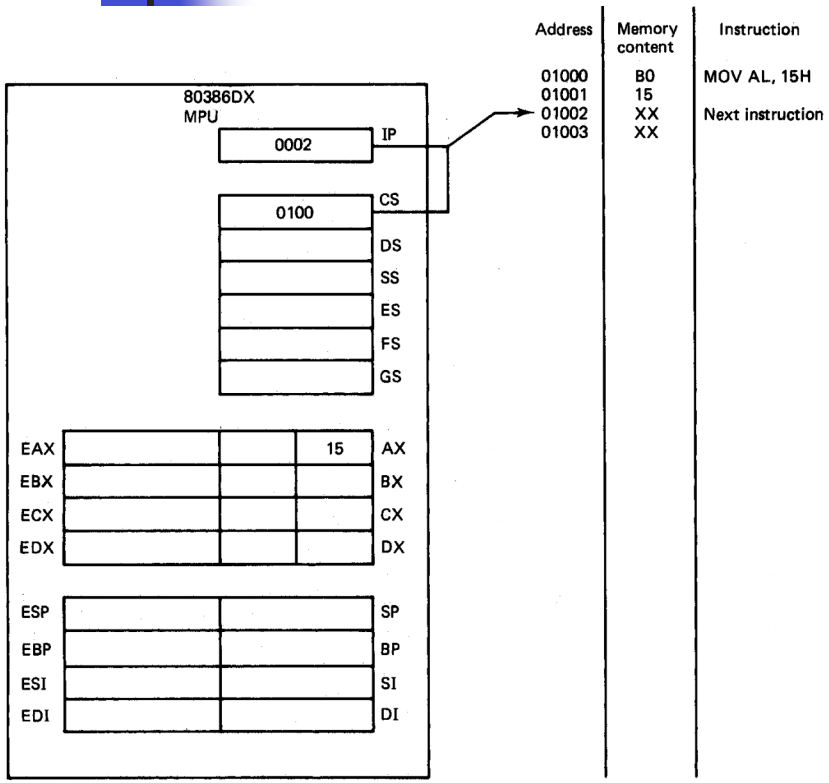
  Move instruction code = B015H

  (01000H) = B0H

  (01001H) = 15H → Immediate data

  (AL) = XX → don't care state

Immediate Operand Addressing Mode



- Example (continued)
- State after execution

  CS:IP = 0100:0002 = 01002H

  01002H → points to next sequential instruction

  (AL) = 15H → Immediate value in copied into AX

  Rest of bits in EAX unaffected

16-bit Memory Operand Addressing Modes

PA = SBA : EA

PA = Segment base : Base + Index + Displacement

$$PA = \left\{\begin{matrix}CS\\SS\\DS\\ES\end{matrix}\right\} : \left\{\begin{matrix}BX\\BP\end{matrix}\right\} + \left\{\begin{matrix}SI\\DI\end{matrix}\right\} + \left\{\begin{matrix}\text{8-bit displacement}\\\text{16-bit displacement}\end{matrix}\right\}$$

- Accessing operands in memory
  - Only one operand can reside in memory—either the source or destination
  - Calculate the 20-bit physical address (PA) at which the operand in stored in memory
  - Perform a read or write to this memory location
  - 16-bit memory addressing modes produce 8088/8086/80286 compatible code

16-bit Memory Operand Addressing Modes

PA = SBA : EA

PA = Segment base : Base + Index + Displacement

$$PA = \left\{\begin{matrix}CS\\SS\\DS\\ES\end{matrix}\right\} : \left\{\begin{matrix}BX\\BP\end{matrix}\right\} + \left\{\begin{matrix}SI\\DI\end{matrix}\right\} + \left\{\begin{matrix}8\text{-bit displacement}\\16\text{-bit displacement}\end{matrix}\right\}$$

- Physical address computation
  - Given in general as
    - PA = SBA:EA
    - SBA = Segment base address
    - EA = Effective address
  - Components of a effective address
    - Base → base registers BX or BP
    - Index → index register SI or DI
    - Displacement → 8 or 16-bit displacement
    - Not all elements are used in all computations—results in a variety of addressing modes

Direct Addressing Mode

PA = Segment base:Direct address

$$PA = \left\{ \begin{array}{l} CS \\ DS \\ SS \\ ES \\ FS \\ GS \end{array} \right\} : \left\{ \text{Direct address} \right\}$$

- Direct addressing mode
  - Similar to immediate addressing in that information coded directly into the instruction
  - Immediate information is the effective address—called the direct address
- Physical address computation

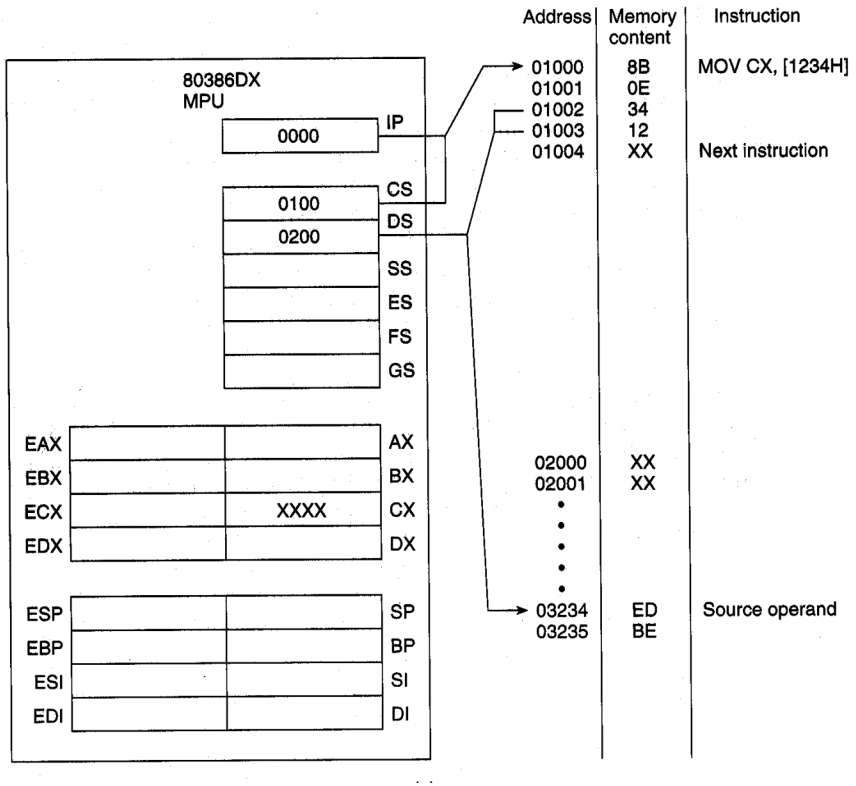
  PA = SBA:EA → 20-bit address

  PA = SBA:[DA] → immediate 8-bit or 16 bit displacement
  - Segment base address is DS by default

    PA = DS:[DA]
  - Segment override prefix (SEG) is required to enable use of another segment register

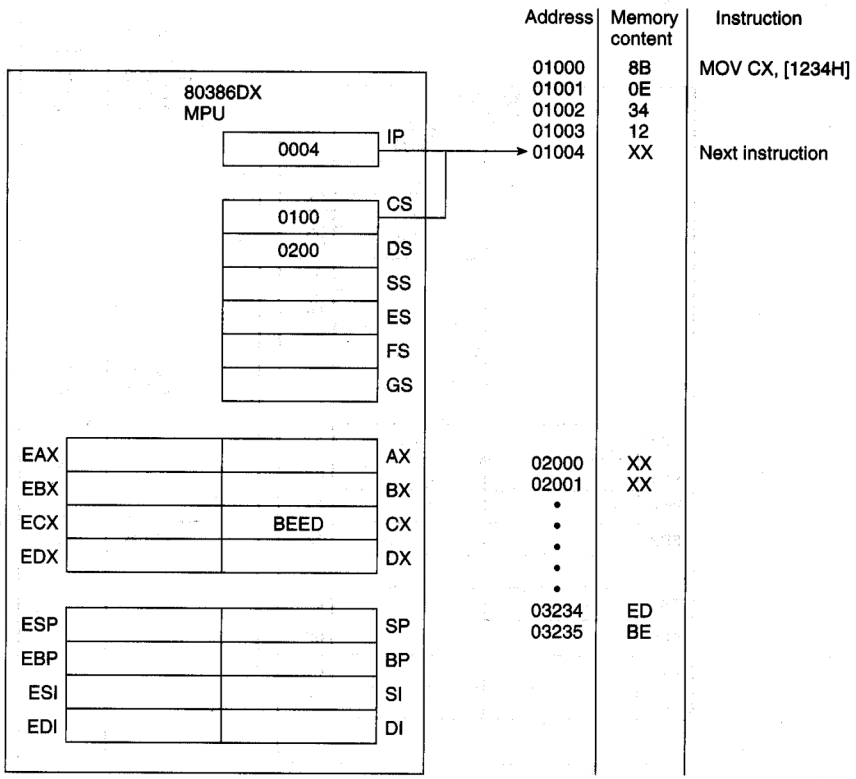
    PA = ES:[DA]

Direct Addressing Mode (Example: MOV CX, [1234H])



- State before fetch and execution
  - Instruction
    CS = 0100H
    IP = 0000H
    CS:IP = 0100:0000H = 01000H
    (01000H,01001H) = Opcode = 8B0E
    (01003H,01002) = DA = 1234H
  - Source operand—direct address
    DS = 0200H
    DA = 1234H
    PA = DS:DA = 0200H:1234H
    = 02000H + 1234H
    = 03234H
    (03235H,03234H) = BEEDH
  - Destination operand—register operand addressing
    (CX) = XXXX → don't care state

Direct Addressing Mode (Example: MOV CX, [1234H])



- Example (continued)
- State after execution
  - Instruction

    CS:IP = 0100:0004 = 01004H

    01004H → points to next sequential instruction
  - Source operand

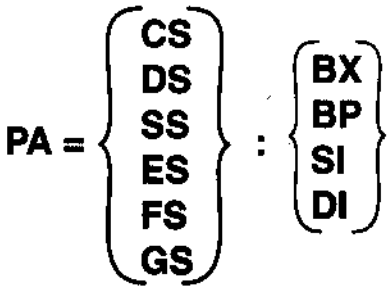
    (03235H,03234H) = BEEDH → unchanged
  - Destination operand

    (CX) = BEED

    Rest of bits in ECX unaffected

Register Indirect Addressing Mode

PA = Segment base: Indirect address



- Register indirect addressing mode
  - Similar to direct addressing in that the affective address is combined with the contents of DS to obtain the physical address
  - Effective address resides in either a base or index register
- Physical address computation

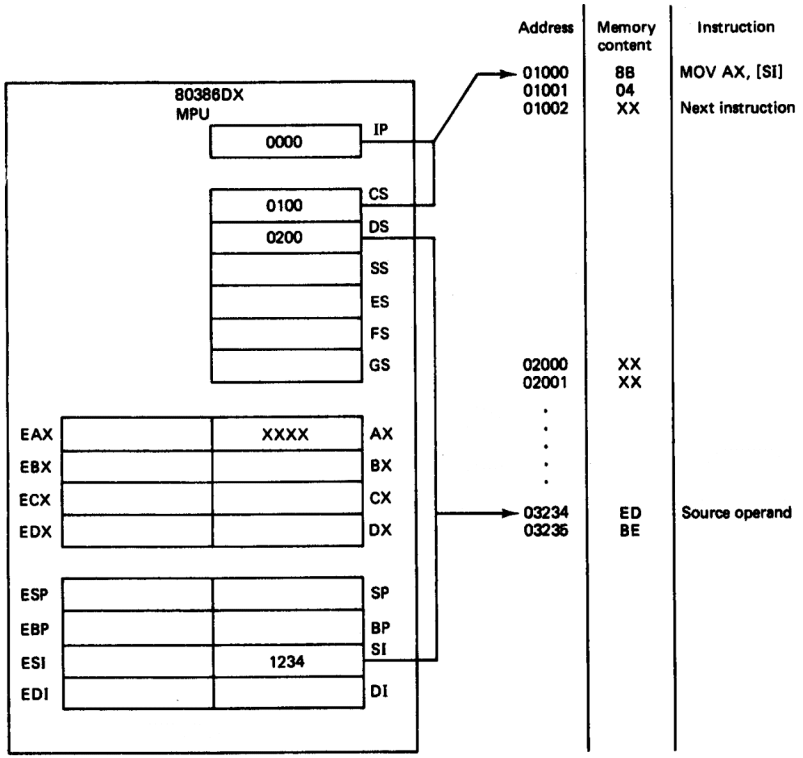
  PA = SBA:EA → 20-bit address

  PA = SBA:[Rx] → 16-bit offset
  - Segment base address is DS by default

    PA = DS:[Rx]
  - Segment override prefix (SEG) is required to enable use of another segment register

    PA = ES:[Rx]

Register Indirect Addressing Mode (example: MOV AX, [SI])



- State before fetch and execution
  - Instruction

    CS = 0100H, IP = 0000H

    CS:IP = 0100:0000H = 01000H

    (01000H,01001H) = Opcode = 8B04H
  - Source operand—register indirect addressing

    DS = 0200H, SI = 1234H

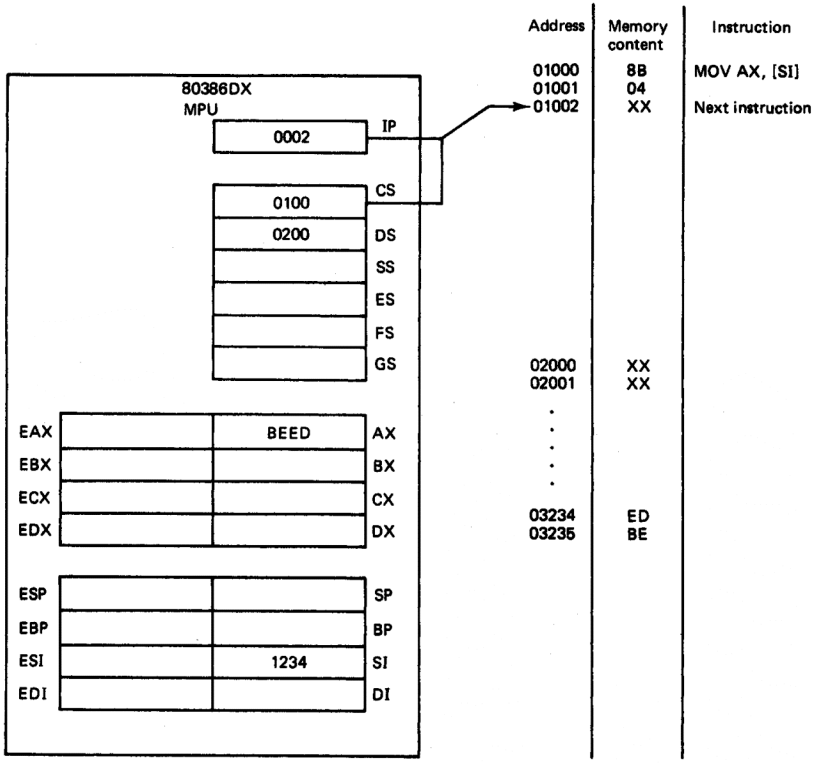
    PA = DS:SI = 0200H:1234H

    = 02000H +1234H = 03234H

    (03235H,03234H) = BEEDH
  - Destination operand—register operand addressing

    (AX) = XXXX → don't care state

Register Indirect Addressing Mode (example: MOV AX, [SI])



- Example (continued)
- State after execution
  - Instruction

    CS:IP = 0100:0002 = 01002H

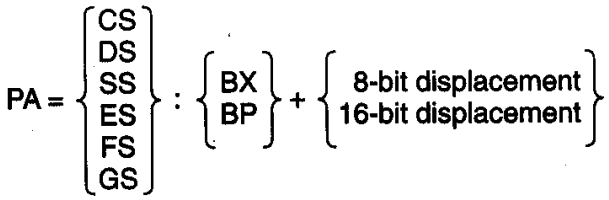
    01002H → points to next sequential instruction
  - Source operand

    (03235H,03234H) = BEEDH → unchanged
  - Destination operand

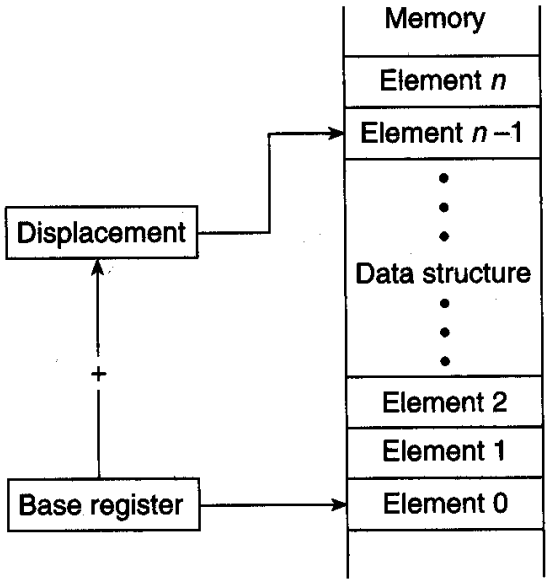
    (AX) = BEED

    Rest of bits in EAX unaffected

Base Addressing Mode



(a)



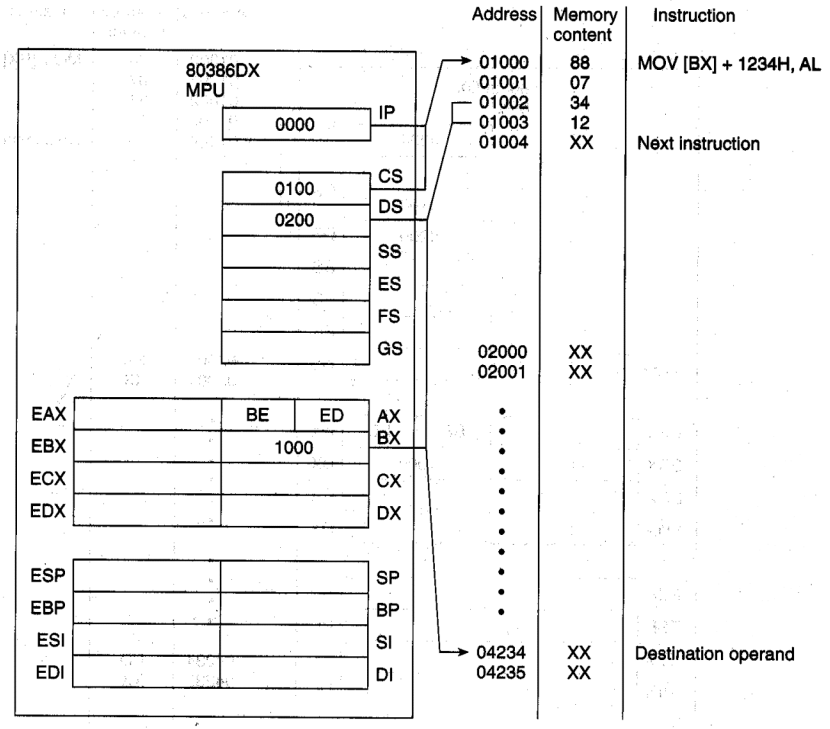
(b)

- **Based addressing mode**
  - Effective address formed from contents of a base register and a displacement
    - Base register is either BX or BP (stack)
    - Direct/indirect displacement is 8-bit or 16bit
- **Physical address computation**

  PA = SBA:EA → 20-bit address

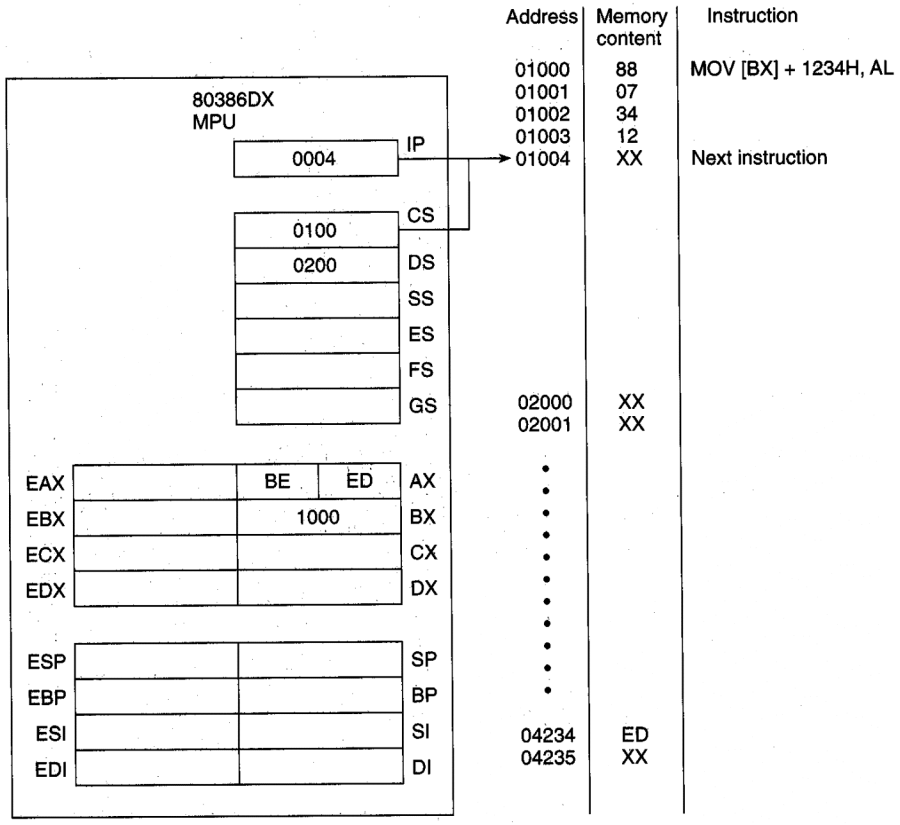
  PA = SBA:[BX or BP] + DA
- **Accessing a data structure**
  - Based addressing makes it easy to access elements of data in an array
  - Address in base register points to start of the array
  - Displacement selects the element within the array
  - Value of the displacement is simply changed to access another element in the array
  - Program changes value in base register to select another array

Base Addressing Mode (Example: MOV [BX] + 1234H, AL)



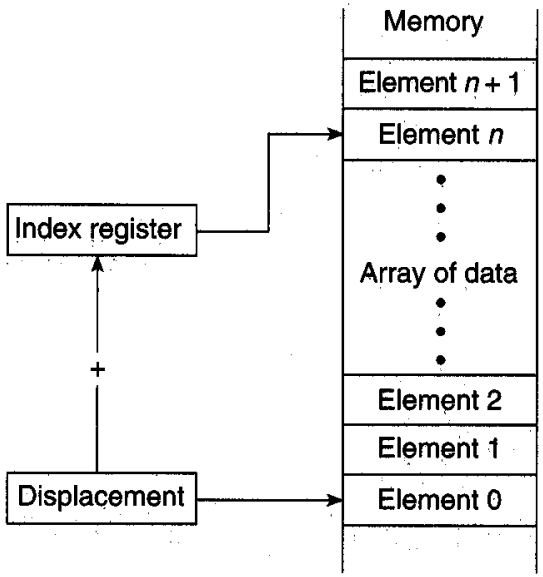
- Instruction
  - CS = 0100H, IP = 0000H
  - CS:IP = 0100:0000H = 01000H
  - (01000H,01001H) = Opcode = 8807H
  - (01003H,01002H) = displacement = 1234H
- Destination operand—based addressing
  - DS = 0200H,BX = 1000H
  - DA = 1234H
  - PA = DS:BX+DA = 0200H:1000H+1234H
    = 02000H+1000H+1234H
    = 04234H
  - (04234H) = XXH
- Source operand—register operand addressing
  - (AL) = ED

Base Addressing Mode (Example: MOV [BX] + 1234H, AL)



- State after execution
  - Instruction
    CS:IP = 0100:0004 = 01004H
    01004H → points to next sequential instruction
  - Destination operand
    (04234H) = EDH
  - Source operand
    (AL) = EDH → unchanged

Indexed Addressing Mode



(a)

PA = Segment base: index + displacement

$$PA = \left\{\begin{matrix}CS\\DS\\SS\\ES\\FS\\GS\end{matrix}\right\} : \left\{\begin{matrix}SI\\DI\end{matrix}\right\} + \left\{\begin{matrix}\text{8-bit displacement}\\\text{16-bit displacement}\end{matrix}\right\}$$

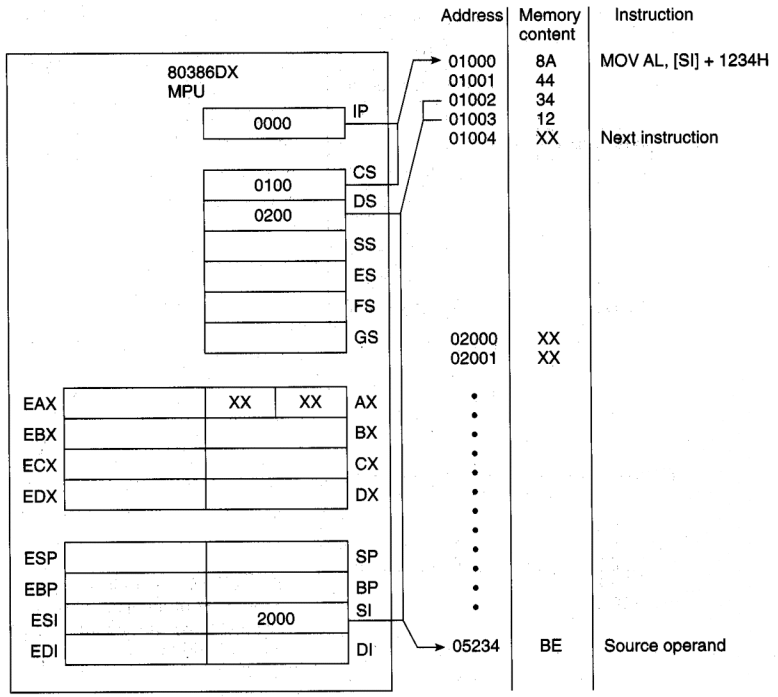
(b)

- **Indexed addressing mode**
  - **Similar to based addressing, it makes accessing elements of data in an array easy**
  - **Displacement points to the beginning of array in memory**
  - **Index register selects element in the array**
  - **Program simply changes the value of the displacement to access another array**
  - **Program changes (recomputes) value in index register to select another element in the array**
- **Effective address formed from direct displacement and contents of an index register**
  - **Direct displacement is 8-bit or 16-bit**
  - **Index register is either SI → source operand or DI → destination operand**
- **Physical address computation**

  PA = SBA:EA → 20-bit address

  PA = SBA: DA + [SI or DI]

Indexed Addressing Mode (Example: MOV AL,[SI] +1234H)



- State before fetch and execution
  - Instruction

    CS = 0100H, IP = 0000H

    CS:IP = 0100:0000H = 01000H

    (01000H,01001H) = Opcode = 8A44H

    (01003H,01002H) = Direct displacement = 1234H
  - Source operand—indexed addressing

    DS = 0200H,SI = 2000H,DA = 1234H

    PA = DS:SI+DA = 0200H:2000H+1234H

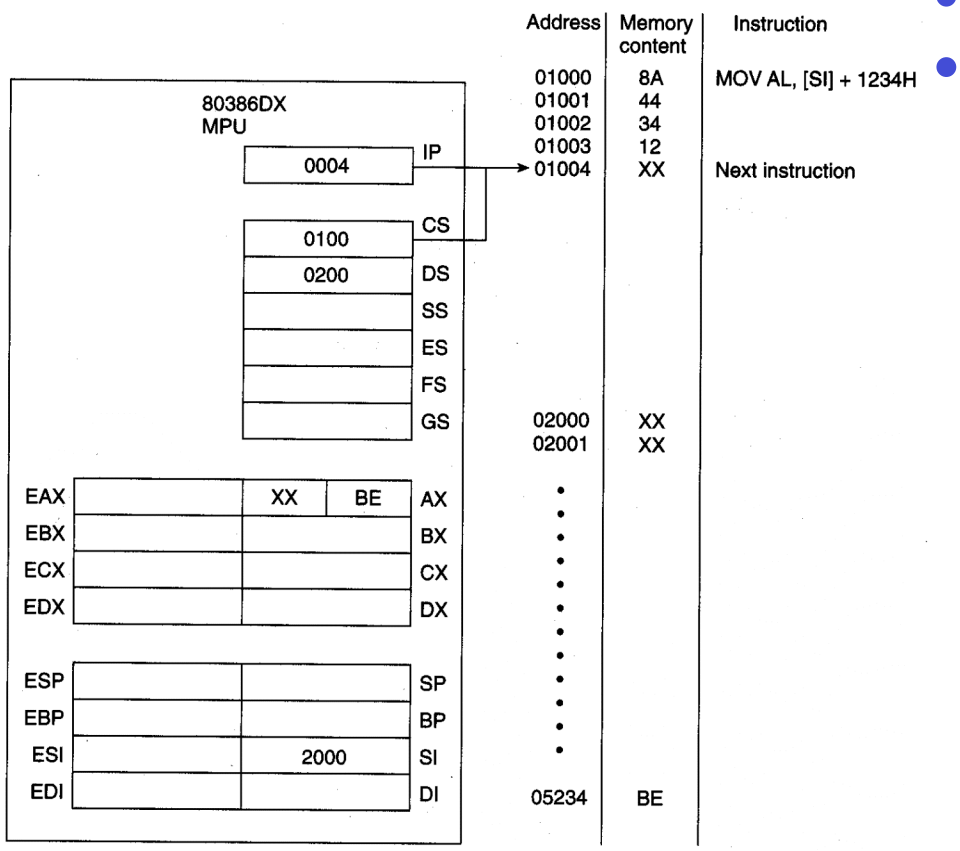
    = 02000H+2000H+1234H

    = 05234H

    (05234H) = BEH
  - Destination operand—register operand addressing

    (AL) = XX → don't care state

Indexed Addressing Mode (Example: MOV AL,[SI] +1234H)



- Example (continued)
- State after execution
  - Instruction

    CS:IP = 0100:0004 = 01004H

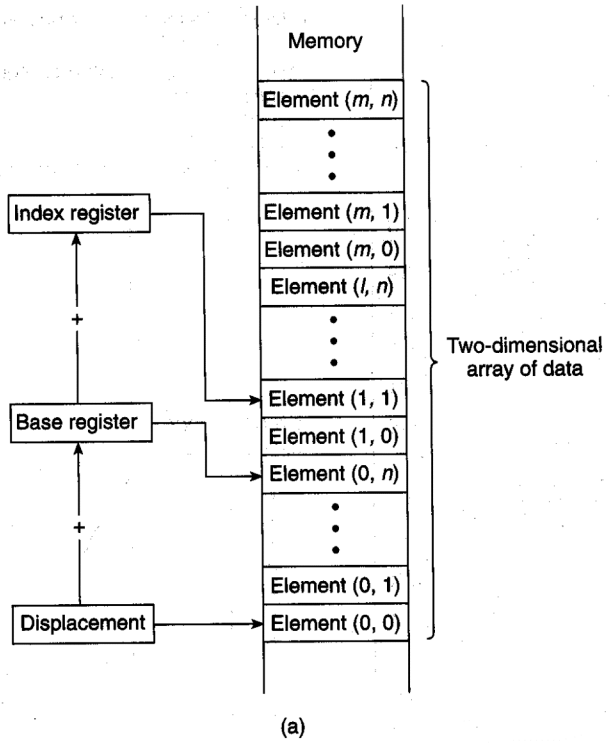
    01004H → points to next sequential instruction
  - Source operand

    (05234H) = BEH → unchanged
  - Destination operand

    (AL) = BEH

    Rest of bits in EAX unaffected

Based-Indexed Addressing Mode



(a)

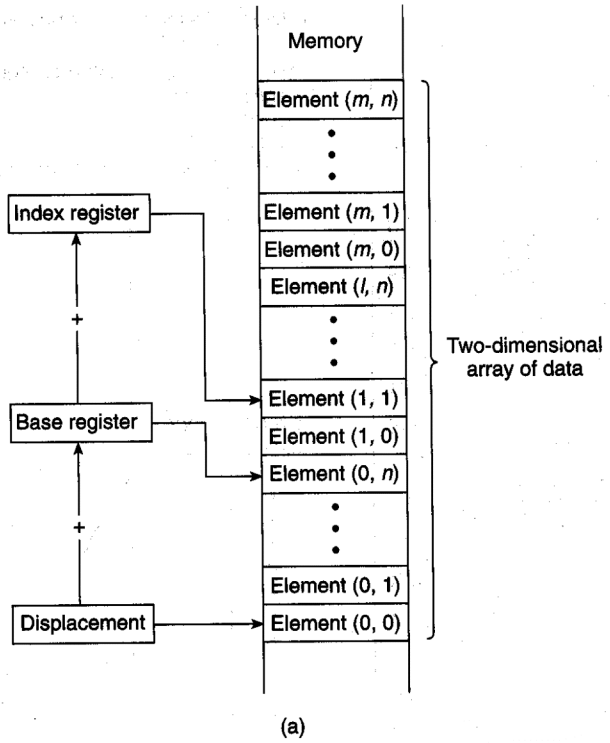
PA = Segment base: base + index + displacement

$$PA = \left\{\begin{matrix}CS\\DS\\SS\\ES\\FS\\GS\end{matrix}\right\} : \left\{\begin{matrix}BX\\BP\end{matrix}\right\} + \left\{\begin{matrix}SI\\DI\end{matrix}\right\} + \left\{\begin{matrix}\text{8-bit displacement}\\\text{16-bit displacement}\end{matrix}\right\}$$

(b)

- Combines the functions of based and indexed addressing modes
- Enables easy access to two-dimensional arrays of data
- Displacement points to the beginning of array in memory
- Base register selects a row (*m*) of elements
- Index register selects an element in column (*n*)
- Program simply changes the value of the displacement to access another array
- Program changes (recomputes) value in base register to select another row of elements
- Program changes (recomputes) the value of the index register to select the element in another column

Based-Indexed Addressing Mode



(a)

PA = Segment base: base + index + displacement

$$PA = \left\{\begin{matrix}CS\\DS\\SS\\ES\\FS\\GS\end{matrix}\right\} : \left\{\begin{matrix}BX\\BP\end{matrix}\right\} + \left\{\begin{matrix}SI\\DI\end{matrix}\right\} + \left\{\begin{matrix}\text{8-bit displacement}\\\text{16-bit displacement}\end{matrix}\right\}$$

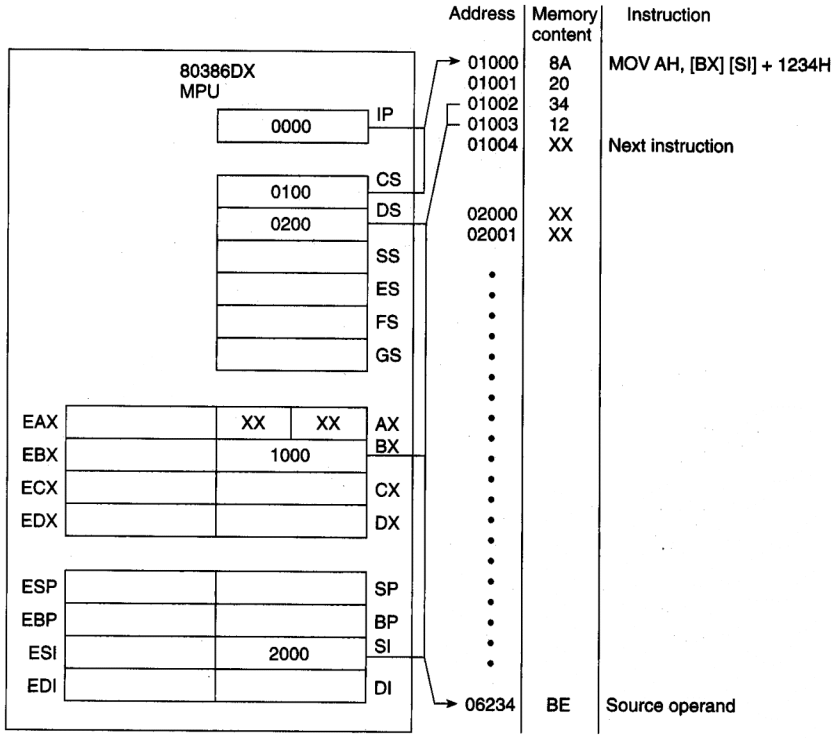
(b)

- Effective address formed from direct displacement and contents of a base register and an index register
  - Direct displacement is 8-bit or 16bit
  - Base register either BX or BP (stack)
  - Index register is either SI → source operand or DI → destination operand
- Physical address computation

  PA = SBA:EA → 20-bit address

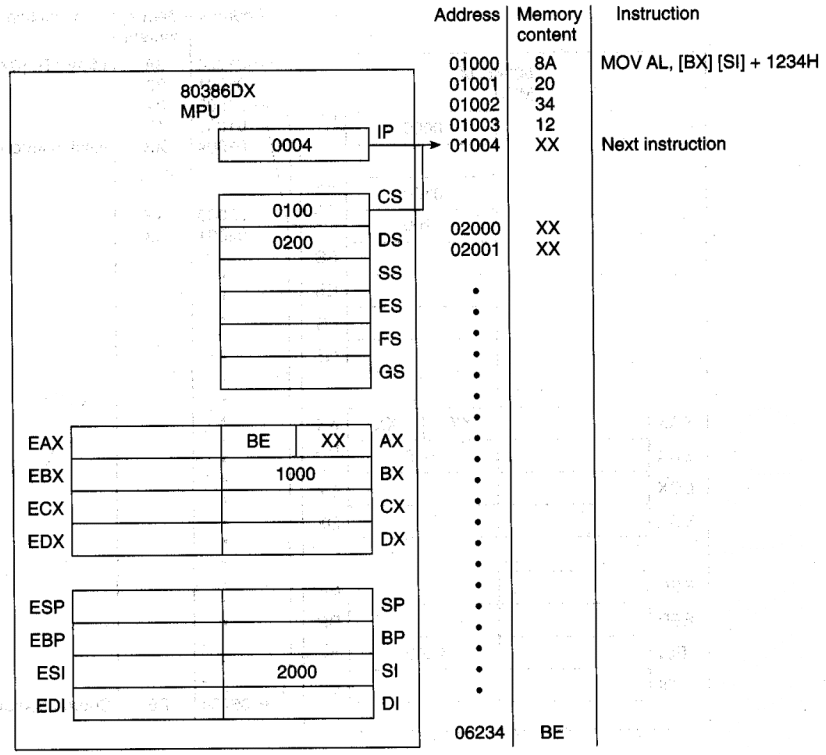
  PA = SBA:DA + [BX or BP] + [SI or DI]

Based- Indexed Addressing Mode Example(MOV AH,[BX][SI] +1234H)



- **State before fetch and execution**
  - **Instruction**
    CS = 0100H, IP = 0000H
    CS:IP = 0100:0000H = 01000H
    (01000H,01001H) = Opcode = 8A20H
    (01003H,01002H) = Direct displacement = 1234H
  - **Source operand—based-indexed addressing**
    DA = 1234H,DS = 0200H,BX = 1000H
    SI = 2000H
    PA = DS:DA +BX +SI
    = 0200H:1234H + 1000H + 2000H
    = 02000H+1234H +1000H + 2000H
    = 06234H
    (06234H) = BEH
  - **Destination operand—register operand addressing**
    (AH) = XX → don't care state

Based- Indexed Addressing Mode Example(MOV AH,[BX][SI] +1234H)



- Example (continued)
- State after execution
  - Instruction

    CS:IP = 0100:0004 = 01004H

    01004H → points to next sequential instruction
  - Source operand

    (06234H) = BEH → unchanged
  - Destination operand

    (AH) = BEH

    Rest of bits in EAX unaffected

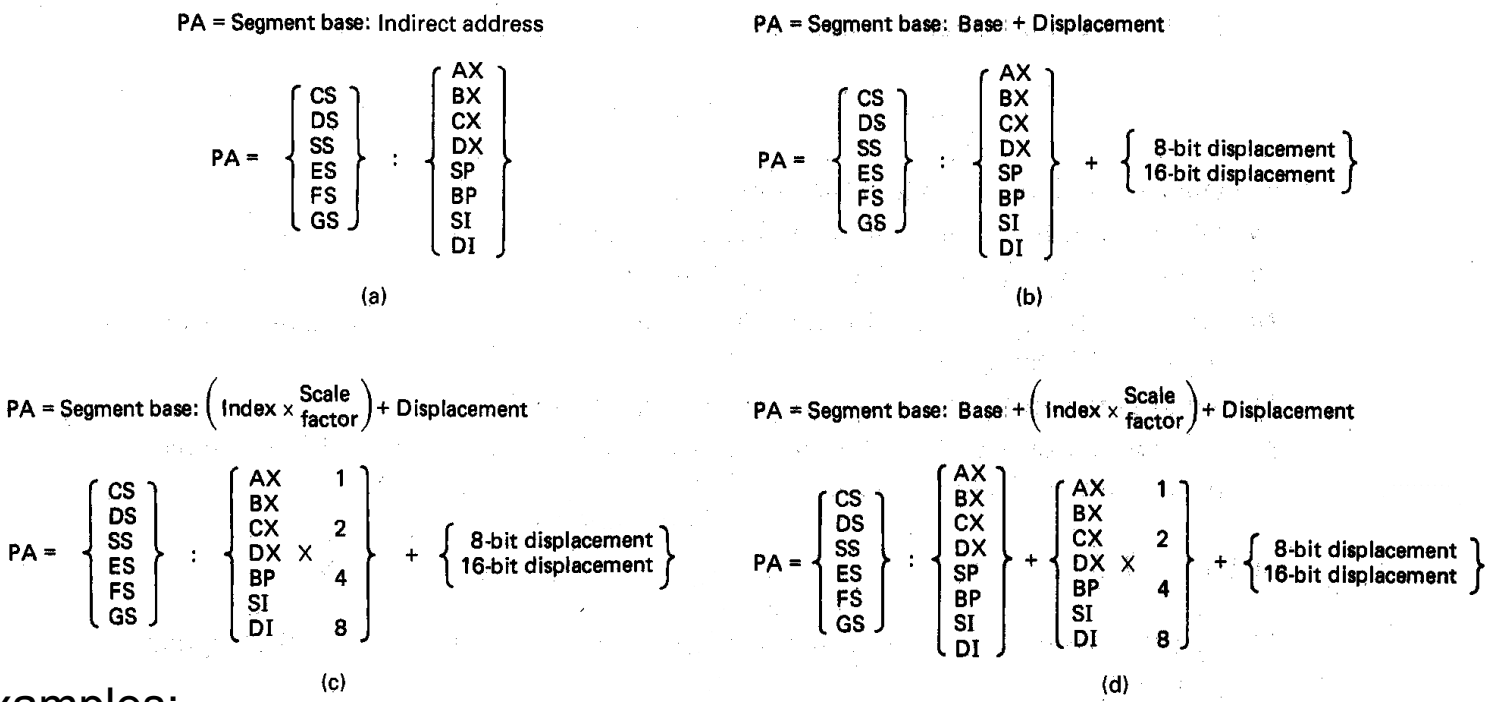
32-bit Memory Operand Addressing Modes

PA = Segment base: { Base + (Index x Scale factor) + Displacement }

PA = { CS, SS, DS, ES, FS, GS } : { AX, BX, CX, DX, SP, BP, SI, DI } + { AX, BX, CX, DX, BP, SI, DI } X { 1, 2, 4, 8 } + { 8-bit displacement, 16-bit displacement }

- Enhanced over the 16 bit memory addressing
- Calculate the 20-bit physical address (PA) at which the operand is stored in memory
- Effective address computation changed
  - Generalized the selection of the register used for the base and index elements
  - Include a fourth element—scale factor
    - Scale factor multiplies the index component
    - Allowed scale factor values are 1,2,4, or 8
- Use of 32-bit memory addressing modes produce code that in not compatible with the 8088/8086/80286

32-bit Memory Operand Addressing Modes



- Examples:

  #1 MOV [AX], BX → destination operand uses AX for indirect address

  #2 MOV BL,[AX] +1234H → source operand uses AX for base address component

  #3 MOV Al,1234H +[SI X 2]; where: DS =0200H, SI =2000H

  PA = DS(0)+1234H+SI X 2) = 02000H+1234H+(2000H X 2) = 07234H