

A slide with a vertical gradient background from yellow to blue. On the left, there is a vertical strip with a yellow-to-blue gradient and a central image of a key on a textured surface. To the right of this strip, the text "ASSEMBLY LANGUAGE PROGRAMMING" is written in bold, dark blue, sans-serif capital letters. Below the title is a list of five items:

- 3.1 Software: The Microcomputer Program
- 3.2 Assembly Language Programming Development on the PC
- 3.3 The Instruction Set
- 3.4 The MOV Instruction
- 3.5 Addressing Modes

At the bottom left, there is a horizontal yellow line above the text "611 37100 微處理機原理與應用 Lecture 03-2". At the bottom right, there is a logo for National Taiwan University, Department of Biomedical Engineering, featuring a stylized 'E' and 'B' and the name "國立台灣大學 生物機電系 林達德".

### 3.1 Software : The Microcomputer Program

- The sequence of commands used to tell a microcomputer what to do is called a **program**. Each command in a program is an **instruction**.
- The 8088 microprocessor performs operations for 117 basic instructions.
- Programs must be coded in **machine language** before they can be run by the 8088. However, programs are normally written in 8088 **assembly language** or a high-level language such as C.
- A single machine language instruction can take one to six bytes of code.

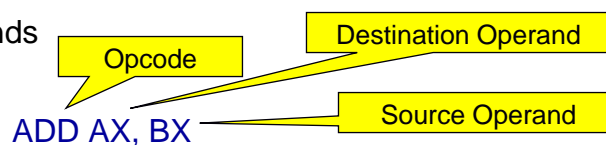
611 37100 微處理機原理與應用 Lecture 03-3

國立台灣大學  
生物機電系  
林達德



### 3.1 Software : The Microcomputer Program

- An instruction can be divided into two parts:
  - ❖ Operation code (opcode) – one- to five-letter **mnemonic**
  - ❖ Operands



- Format of an assembly statement:

**LABEL: INSTRUCTION ; COMMENT**

e.g. `START: MOV AX, BX ; Copy BX into AX`

611 37100 微處理機原理與應用 Lecture 03-4

國立台灣大學  
生物機電系  
林達德



## 3.1 Software : The Microcomputer Program

### ■ Assembly 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= 2000H ;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
```

611 37100 微處理機原理與應用 Lecture 03-5

國立台灣大學  
生物機電系  
林達德



## 3.1 Software : The Microcomputer Program

### ■ Assembly source program (continued)

```
;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
```

611 37100 微處理機原理與應用 Lecture 03-6

國立台灣大學  
生物機電系  
林達德



### 3.1 Software : The Microcomputer Program

- Assembly language must be converted by an **assembler** to an equivalent machine language program for execution by the 8088.
- A directive is a statement that is used to control the translation process of the assembler.

e.g.           DB 64 DUP(?)

- The machine language output produced by the assembler is called **object code**.

- Listing of the assembled program

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



### 3.1 Software : The Microcomputer Program

- Listing of an assembled program

```

microsoft (R) Macro Assembler Version 6.11      12/14/02 15:10:26      Page 1 - 1
BLOCK-MOVE 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
= 0010          N      = 16          ;Bytes to be moved
= 0100          BLK1ADDR= 100H      ;Source block offset address
= 0120          BLK2ADDR= 120H      ;Destination block offset addr
= 2000          DATASEGADDR= 2000H  ;Data segment start address

0000          STACK_SEG  SEGMENT    STACK 'STACK'
0000 0040 [    DB      64 DUP(?)
                00
                ]

0040          STACK_SEG  ENDS
0000          CODE_SEG   SEGMENT    'CODE'
0000          BLOCK     PROC      FAR
                                ASSUME CS:CODE_SEG,SS:STACK_SEG

```



## 3.1 Software : The Microcomputer Program

### ■ Listing of an assembled program

```

                                ;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
                                ;Set up the data segment address
0005 B8 2000          MOV   AX, DATASEGADDR
0008 8E D8            MOV   DS, AX
                                ;Set up the source and destination offset addresses
000A BE 0100          MOV   SI, BLK1ADDR
000D BF 0120          MOV   DI, BLK2ADDR
                                ;Set up 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

```

611 37100 微處理機原理與應用 Lecture 03-9

國立台灣大學  
生物機電系  
林達德



## 3.1 Software : The Microcomputer Program

### ■ Listing of an assembled program

Microsoft (R) Macro Assembler Version 6.11 12/14/02 15:10:26  
BLOCK-MOVE PROGRAM Symbols 2 - 1

#### Segments and Groups:

Name	Size	Length	Align	Combine	Class
CODE_SEG	16 Bit	001D	Para		Private 'CODE'
STACK_SEG	16 Bit	0040	Para		Stack 'STACK'

#### Procedures, parameters and locals:

Name	Type	Value	Attr	Combine	Class
BLOCK	P Far	0000		CODE_SEG	Length= 001D

#### Symbols:

Name	Type	Value	Attr
BLK1ADDR	Number	0100h	
BLK2ADDR	Number	0120h	
DATASEGADDR	Number	2000h	
NXTPT	L Near	0013	CODE_SEG
N	Number	0010h	

0 Warnings

0 Errors

611 37100 微處理機原理與應用 Lecture 03-10

國立台灣大學  
生物機電系  
林達德





### 3.1 Software : The Microcomputer Program

- Assembly language versus high-level language
  - ❖ It is easier to write program with high-level language.
  - ❖ Program written in assembly language usually takes up less memory space and executes much faster.
  - ❖ Device service routines are usually written in assembly language.
  - ❖ Assembly language is used to write those parts of the application that must perform real-time operations, and high-level language is used to write those parts that are not time critical.

611 37100 微處理機原理與應用 Lecture 03-11

國立台灣大學  
生物機電系  
林達德



### 3.2 Assembly Language Programming Development on the PC

- Describing the problem
- Planning the solution
- Coding the solution with assembly language
- Creating the source program
- Assembling the source program into an object module
- Producing a run module
- Verifying the solution
- Programs and files involved in the program development cycle

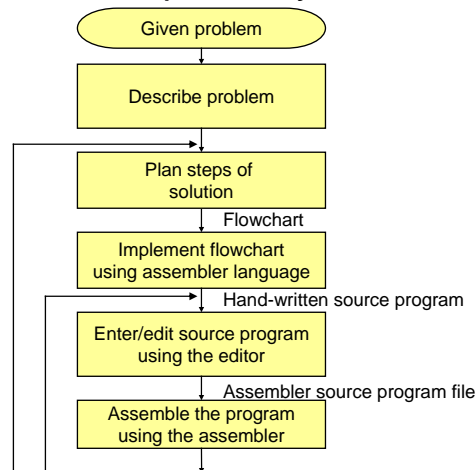
611 37100 微處理機原理與應用 Lecture 03-12

國立台灣大學  
生物機電系  
林達德



## 3.2 Assembly Language Programming Development on the PC

### Program development cycle



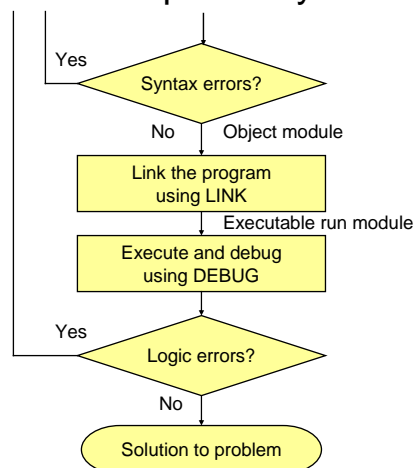
611 37100 微處理機原理與應用 Lecture 03-13

國立台灣大學  
生物機電系  
林達德



## 3.2 Assembly Language Programming Development on the PC

### Program development cycle



611 37100 微處理機原理與應用 Lecture 03-14

國立台灣大學  
生物機電系  
林達德



## 3.2 Assembly Language Programming Development on the PC

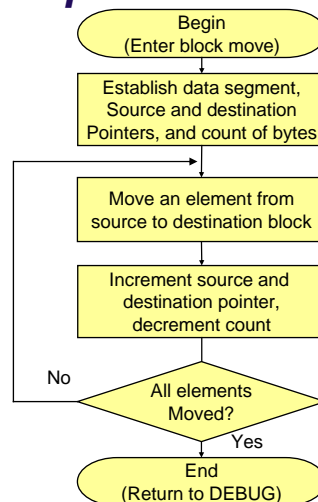
- Describing the problem
  - ❖ Most applications are described with a written document called an **application specification**.
- Planning the solution
  - ❖ A **flow chart** is an outline that both documents the operations that must be performed by software to implement the planned solution and shows the sequence in which they are performed.

611 37100 微處理機原理與應用 Lecture 03-15

國立台灣大學  
生物機電系  
林達德



## 3.2 Assembly Language Programming Development on the PC



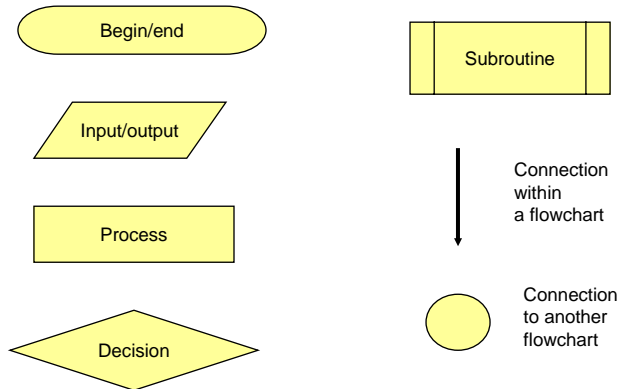
Flow chart of a block-move program

611 37100 微處理機原理與應用 Lecture 03-16

國立台灣大學  
生物機電系  
林達德



## 3.2 Assembly Language Programming Development on the PC



Commonly used flowchart symbols

611 37100 微處理機原理與應用 Lecture 03-17

國立台灣大學  
生物機電系  
林達德



## 3.2 Assembly Language Programming Development on the PC

- Coding the solution with assembly language
  - ❖ Two types of statements are used in the source program
    - The assembly language instructions
    - The directives
  - ❖ The **assembly language instructions** are used to tell the microprocessor what operations are to be performed to implement the application.
  - ❖ A **directive** is the instruction to the assembler program used to convert the assembly language program into machine code.

611 37100 微處理機原理與應用 Lecture 03-18

國立台灣大學  
生物機電系  
林達德



## 3.2 Assembly Language Programming Development on the PC

### ■ Coding the solution with assembly language

#### ❖ The **assembly language instructions**

[Example]

```
MOV  AX, DATASEGMENT
MOV  DS, AX
MOV  SI, BLK1ADDR
MOV  DI, BLK2ADDR
MOV  CX, N
```

#### ❖ The **directive**

[Example]

```
BLOCK PROC FAR
or
BLOCK ENDP
```

611 37100 微處理機原理與應用 Lecture 03-19

國立台灣大學  
生物機電系  
林達德



## 3.2 Assembly Language Programming Development on the PC

### ■ Creating the source program

- ❖ The EDIT editor
- ❖ The Notepad editor in Windows
- ❖ The Microsoft PWB (Programmer's Work Bench)

### ■ Assembling the source program into an object module

- ❖ The Microsoft MASM assembler
- ❖ The Microsoft PWB (Programmer's Work Bench)
- ❖ The **assembler source file** and the **object module**

611 37100 微處理機原理與應用 Lecture 03-20

國立台灣大學  
生物機電系  
林達德



## 3.2 Assembly Language Programming Development on the PC

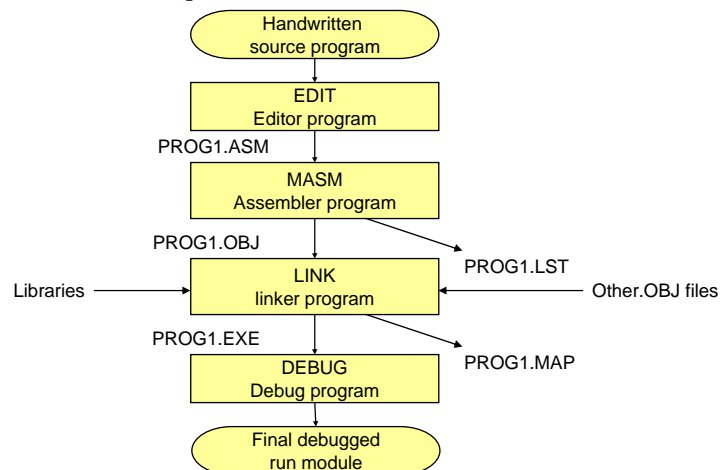
- Producing a run module
  - ❖ The object module must be processed by the LINK program to produce an executable **run module**.
- Verifying the solution
- Programs and files involved in the program development cycle
  - ❖ PROG1.ASM (Editor)
  - ❖ PROG1.OBJ (Assembler)
  - ❖ PROG1.LST (Assembler)
  - ❖ PROG1.EXE (Linker)
  - ❖ PROG1.MAP (Linker)

611 37100 微處理機原理與應用 Lecture 03-21

國立台灣大學  
生物機電系  
林達德



## 3.2 Assembly Language Programming Development on the PC



The development programs and users files

611 37100 微處理機原理與應用 Lecture 03-22

國立台灣大學  
生物機電系  
林達德



### 3.3 The Instruction Set

- The instruction set of a microprocessor defines the basic operations that a programmer can specify to the device to perform
- Instruction set groups
  - ❖ Data transfer instructions
  - ❖ Arithmetic instructions
  - ❖ Logic instructions
  - ❖ String manipulation instructions
  - ❖ Control transfer instructions
  - ❖ Processor control instructions



### 3.3 The Instruction Set

#### ■ Data transfer instructions

Mnemonic and Description	Instruction Code			
<b>DATA TRANSFER</b>				
<b>MOV – Move:</b>	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 1 0 d w	mod reg r/m		
Immediate to Register/Memory	1 1 0 0 0 1 1 w	mod 0 0 r/m	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 w	addr-low	addr-high	
Accumulator to Memory	1 0 1 0 0 0 1 w	addr-low	addr-high	
Register/Memory to Segment Register	1 0 0 0 1 1 1 0	mod 0 reg r/m		
Segment Register to Register/Memory	1 0 0 0 1 1 0 0	mod 0 reg r/m		
<b>PUSH – Push:</b>				
Register/Memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m		
Register	0 1 0 1 0 reg			
Segment Register	0 0 0 reg 1 1 0			
<b>POP – Pop:</b>				
Register/Memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m		
Register	0 1 0 1 1 reg			
Segment Register	0 0 0 reg 1 1 1			
<b>XCHG – Exchange:</b>				
Register/Memory with Register	1 0 0 0 0 1 1 w	mod reg r/m		
Register with Accumulator	1 0 0 1 0 reg			



### 3.3 The Instruction Set

#### ■ Data transfer instructions

Mnemonic and Description	Instruction Code	
<b>DATA TRANSFER</b>		
<b>IN – Input from:</b>		
Fixed Port	11 10 0 10 w	port
Variable Port	11 10 1 10 w	
<b>OUT – Output to:</b>		
Fixed Port	11 10 0 11 w	port
Variable Port	11 10 1 11 w	
<b>XLAT</b> – Translate Byte to AL	11 01 0 11 1	
<b>LEA</b> – Load EA to Register	10 00 1 10 1	mod reg r/m
<b>LDS</b> – Load Pointer to DS	11 00 0 10 1	mod reg r/m
<b>LES</b> – Load Pointer to ES	11 00 0 10 0	mod reg r/m
<b>LAHF</b> – Load AH with Flags	10 01 1 11 1	
<b>SAHF</b> – Store AH into Flags	10 01 1 11 0	
<b>PUSHF</b> – Push Flags	10 01 1 10 0	
<b>POPF</b> – Pop Flags	10 01 1 10 1	



### 3.3 The Instruction Set

#### ■ Arithmetic instructions

Mnemonic and Description	Instruction Code			
<b>ARITHMETIC</b>	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
<b>ADD – Add:</b>				
Reg./Memory with Register to Either	00 00 0 0 d w	mod reg r/m		
Immediate to Register/Memory	10 00 0 0 s w	mod 0 0 r/m	data	data if s w – 01
Immediate to Accumulator	00 00 0 10 w	data	data if w – 1	
<b>ADC – Add with Carry:</b>				
Reg./Memory with Register to Either	00 01 0 0 d w	mod reg r/m		
Immediate to Register/Memory	10 00 0 0 s w	mod 0 1 0 r/m	data	data if s w – 01
Immediate to Accumulator	00 01 0 10 w	data	data if w – 1	
<b>INC – Increment:</b>				
Register/Memory	11 11 1 11 w	mod 0 0 r/m		
Register	01 0 00 reg			
<b>AAA – ASCII Adjust for Add</b>	00 11 0 11 1			
<b>BAA – Decimal Adjust for Add</b>	00 10 0 11 1			
<b>SUB – Subtract:</b>				
Reg./Memory and Register to Either	00 10 1 0 d w	mod reg r/m		
Immediate from Register/Memory	10 00 0 0 s w	mod 0 1 1 r/m	data	data if s w – 01
Immediate from Accumulator	00 10 1 10 w	data	data if w – 1	
<b>SSB – Subtract with Borrow</b>				
Reg./Memory and Register to Either	00 01 1 0 d w	mod reg r/m		
Immediate from Register/Memory	10 00 0 0 s w	mod 0 1 1 r/m	data	data if s w – 01
Immediate from Accumulator	00 01 1 11 w	data	data if w – 1	



## 3.3 The Instruction Set

### Arithmetic instructions

Mnemonic and Description	Instruction Code			
	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
<b>ARITHMETIC</b>				
<b>DEC</b> – Decrement				
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 1 r/m		
Register	0 1 0 0 1 reg			
<b>NEG</b> – Change sign	1 1 1 1 0 1 1 w	mod 0 1 1 r/m		
<b>CMP</b> – Compare:				
Register/Memory and Register	0 0 1 1 0 d w	mod reg r/m		
Immediate with Register/Memory	1 0 0 0 0 s w	mod 1 1 1 r/m	data	data if s w = 01
Immediate with Accumulator	0 0 1 1 1 0 w	data	data if w = 1	
<b>AAS</b> – ASCII Adjust for Subtract	0 0 1 1 1 1 1			
<b>DAS</b> – Decimal Adjust for Subtract	0 0 1 0 1 1 1			
<b>MUL</b> – Multiply (Unsigned)	1 1 1 1 0 1 1 w	mod 1 0 0 r/m		
<b>IMUL</b> – Integer Multiply (Signed)	1 1 1 1 0 1 1 w	mod 1 0 1 r/m		
<b>AAM</b> – ASCII Adjust for Multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0		
<b>DIV</b> – Divide (Unsigned)	1 1 1 1 0 1 1 w	mod 1 1 0 r/m		
<b>IDIV</b> – Integer Divide (Signed)	1 1 1 1 0 1 1 w	mod 1 1 1 r/m		
<b>AAD</b> – ASCII Adjust for Divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0		
<b>CBW</b> – Convert Byte to Word	1 0 0 1 1 0 0 0			
<b>CWD</b> – Convert Word to Double Word	1 0 0 1 1 0 0 1			

611 37100 微處理機原理與應用 Lecture 03-27

國立台灣大學  
生物機電系  
林達德



## 3.3 The Instruction Set

### Logic instructions

Mnemonic and Description	Instruction Code			
	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
<b>LOGIC</b>				
<b>NOT</b> – Invert	1 1 1 1 0 1 1 w	mod 0 1 0 r/m		
<b>SHL/SAL</b> – Shift Logical/Arithmetic Left	1 1 0 1 0 0 v w	mod 1 0 0 r/m		
<b>SHR</b> – Shift Logical Right	1 1 0 1 0 0 v w	mod 1 0 1 r/m		
<b>SAR</b> – Shift Arithmetic Right	1 1 0 1 0 0 v w	mod 1 1 1 r/m		
<b>ROL</b> – Rotate Left	1 1 0 1 0 0 v w	mod 0 0 0 r/m		
<b>ROR</b> – Rotate Right	1 1 0 1 0 0 v w	mod 0 0 1 r/m		
<b>RCL</b> – Rotate Through Carry Flag Left	1 1 0 1 0 0 v w	mod 0 1 0 r/m		
<b>RCR</b> – Rotate Through Carry Right	1 1 0 1 0 0 v w	mod 0 1 1 r/m		
<b>AND</b> – And:				
Reg./Memory and Register to Either	0 0 1 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w = 1
Immediate to Accumulator	0 0 1 0 1 0 w	data	data if w = 1	
<b>TEST</b> – And Function to Flags, No Result:				
Register/Memory and Register	1 0 0 0 1 0 w	mod reg r/m		
Immediate Data and Register/Memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1
Immediate Data and Accumulator	1 0 1 0 1 0 w	data	data if w = 1	
<b>OR</b> – Or:				
Reg./Memory and Register to Either	0 0 0 1 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 w	mod 0 1 1 r/m	data	data if w = 1
Immediate to Accumulator	0 0 0 0 1 1 w	data	data if w = 1	
<b>XOR</b> – Exclusive or:				
Reg./Memory and Register to Either	0 0 1 1 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w = 1
Immediate to Accumulator	0 0 1 1 0 1 w	data	data if w = 1	

611 37100 微處理機原理與應用 Lecture 03-28

國立台灣大學  
生物機電系  
林達德



## 3.3 The Instruction Set

### ■ String manipulation instructions

Mnemonic and Description	Instruction Code
<b>STRING MANIPULATION</b>	
REP – Repeat	1111001z
MOVS – Move Byte/Word	1010010w
CMPS – Compare Byte/Word	1010011w
SCAS – Scan Byte/Word	1010111w
LODS – Load Byte/Wd to AL/AX	1010110w
STOS – Stor Byte/Wd from AL/A	1010101w

611 37100 微處理機原理與應用 Lecture 03-29

國立台灣大學  
生物機電系  
林達德



## 3.3 The Instruction Set

### ■ Control transfer instructions

Mnemonic and Description	Instruction Code		
<b>CONTROL TRANSFER</b>			
<b>CALL – Call:</b>			
Direct within Segment	11101000	disp-low	disp-high
Indirect within Segment	11111111	mod010r/m	
Direct Intersegment	10011010	offset-low	offset-high
		seg-low	seg-high
Indirect Intersegment	11111111	mod011r/m	
<b>JMP – Unconditional Jump:</b>			
Direct within Segment	76543210	76543210	76543210
Direct within Segment-Short	11101001	disp-low	disp-high
Indirect within Segment	11101011	disp	
		mod100r/m	
Direct Intersegment	11101010	offset-low	offset-high
		seg-low	seg-high
Indirect Intersegment	11111111	mod101r/m	

611 37100 微處理機原理與應用 Lecture 03-30

國立台灣大學  
生物機電系  
林達德



## 3.3 The Instruction Set

### Control transfer instructions

Mnemonic and Description	Instruction Code		
<b>RET</b> – Return from CALL:			
Within Segment	11000011		
Within Segment Adding Immediate to SP	11000010	data-low	data-high
Intersegment	11001011		
Intersegment Adding Immediate to SP	11001010	data-low	data-high
<b>JE/JZ</b> – Jump on Equal/Zero	01110100	disp	
<b>JL/JNGE</b> – Jump on Less/Not Greater or Equal	01111100	disp	
<b>JLE/JNG</b> – Jump on Less or Equal/Not Greater	01111110	disp	
<b>JB/JNAE</b> – Jump on Below/Not Above or Equal	01110010	disp	
<b>JBE/JNA</b> – Jump on Below or Equal/Not Above	01110110	disp	
<b>JP/JPE</b> – Jump on Parity/Parity Even	01111010	disp	
<b>JO</b> – Jump on Overflow	01110000	disp	
<b>JS</b> – Jump on Sign	01111000	disp	
<b>JNE/JNZ</b> – Jump on Not Equal/Not Zero	01110101	disp	
<b>JNL/JGE</b> – Jump on Not Less/Greater or Equal	01111101	disp	
<b>JNLE/JG</b> – Jump on Not Less or Equal/Greater	01111111	disp	

611 37100 微處理機原理與應用 Lecture 03-31

國立台灣大學  
生物機電系  
林達德



## 3.3 The Instruction Set

### Control transfer instructions

Mnemonic and Description	Instruction Code	
<b>JNB/JAE</b> – Jump on Not Below/Above or Equal	01110011	disp
<b>JNBE/JA</b> – Jump on Not Below or Equal/Above	01110111	disp
<b>JNP/JPO</b> – Jump on Not Par/Par Odd	01111011	disp
<b>JNO</b> – Jump on Not Overflow	01110001	disp
<b>JNS</b> – Jump on Not Sign	01111001	disp
<b>LOOP</b> – Loop CX Times	11100010	disp
<b>LOOPZ/LOOPE</b> – Loop While Zero/Equal	11100001	disp
<b>LOOPNZ/LOOPNE</b> – Loop While Not Zero/Equal	11100000	disp
<b>JCXZ</b> – Jump on CX Zero	11100011	disp
<b>INT</b> – Interrupt		
Type Specified	11001101	type
Type 3	11001100	
<b>INTO</b> – Interrupt on Overflow	11001110	
<b>IRET</b> – Interrupt Return	11001111	

611 37100 微處理機原理與應用 Lecture 03-32

國立台灣大學  
生物機電系  
林達德



### 3.3 The Instruction Set

- Process control instructions

Mnemonic and Description	Instruction Code	
	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
<b>PROCESSOR CONTROL</b>		
CLC – Clear Carry	1 1 1 1 1 0 0 0	
CMC – Complement Carry	1 1 1 1 0 1 0 1	
STC – Set Carry	1 1 1 1 1 0 0 1	
CLD – Clear Direction	1 1 1 1 1 1 0 0	
STD – Set Direction	1 1 1 1 1 1 0 1	
CLI – Clear Interrupt	1 1 1 1 1 0 1 0	
STI – Set Interrupt	1 1 1 1 1 0 1 1	
HLT – Halt	1 1 1 1 0 1 0 0	
WAIT – Wait	1 0 0 1 1 0 1 1	
ESC – Escape (to External Device)	1 1 0 1 1 x x x	mod x x x r/m
LOCK – Bus Lock Prefix	1 1 1 1 0 0 0 0	



### 3.4 The MOV Instruction

- The move (MOV) instruction is used to transfer a byte or a word of data from a source operand to a destination operand.

Mnemonic	Meaning	Format	Operation	Flags affected
MOV	Move	MOV D, S	(S) → (D)	None

e.g. MOV DX, CS

MOV [SUM], AX



### 3.4 The MOV Instruction

- Note that the MOV instruction cannot transfer data directly between external memory.

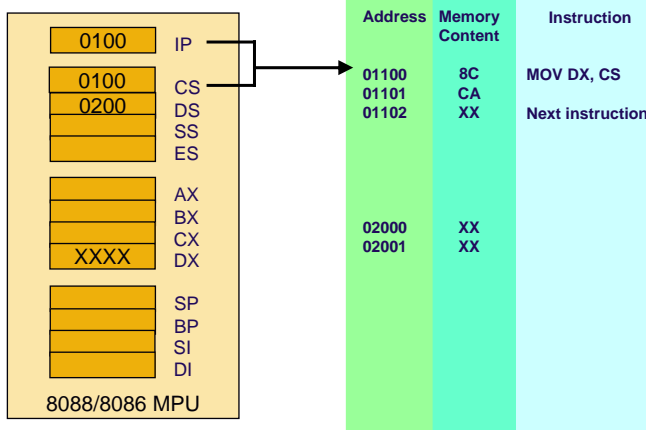
Destination	Source
Memory	Accumulator
Accumulator	Memory
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Seg-reg	Reg16
Seg-reg	Mem16
Reg16	Seg-reg
Memory	Seg-reg

Allowed operands for MOV instruction



### 3.4 The MOV Instruction

- MOV DX, CS

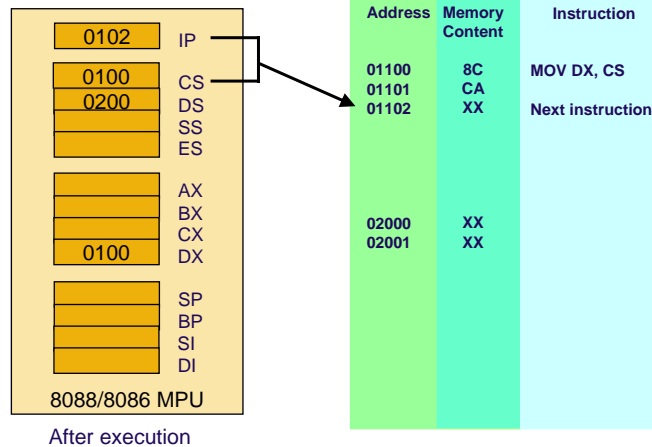


Before execution



### 3.4 The MOV Instruction

#### ■ MOV DX, CS



611 37100 微處理機原理與應用 Lecture 03-37

國立台灣大學  
生物機電系  
林達德



### 3.5 Addressing Modes

- Register operand addressing mode
- Immediate operand addressing mode
- Memory operand addressing mode
  - ❖ Direct addressing mode
  - ❖ Register indirect addressing mode
  - ❖ Based addressing mode
  - ❖ Indexed addressing mode
  - ❖ Based-indexed addressing mode

611 37100 微處理機原理與應用 Lecture 03-38

國立台灣大學  
生物機電系  
林達德



### 3.5 Addressing Modes

■ Register operand addressing mode

The operand to be accessed is specified as residing in an internal register of 8088.

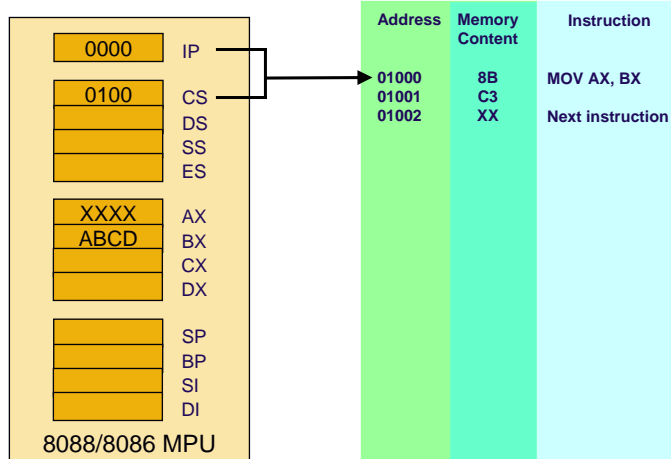
e.g. `MOV AX, BX`

Register	Operand Sizes	
	Byte (Reg 8)	Word (Reg 16)
Accumulator	AL, AH	AX
Base	BL, BH	BX
Count	CL, CH	CX
Data	DL, DH	DX
Stack pointer	-	SP
Base pointer	-	BP
Source index	-	SI
Destination index	-	DI
Code segment	-	CS
Data segment	-	DS
Stack segment	-	SS
Extra segment	-	ES



### 3.5 Addressing Modes

■ Register operand addressing mode

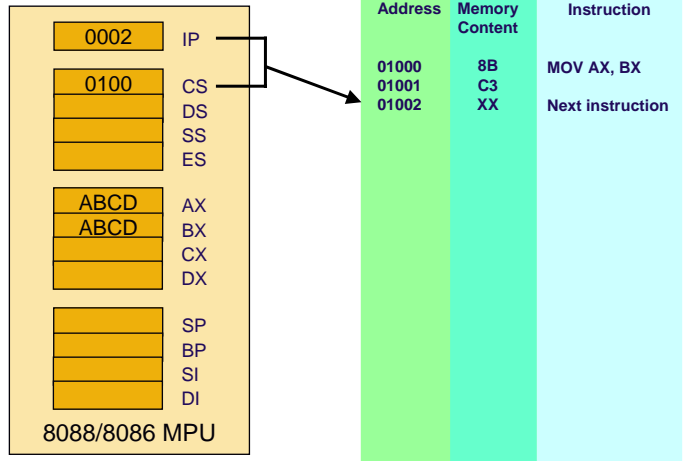


Before execution



### 3.5 Addressing Modes

#### Register operand addressing mode

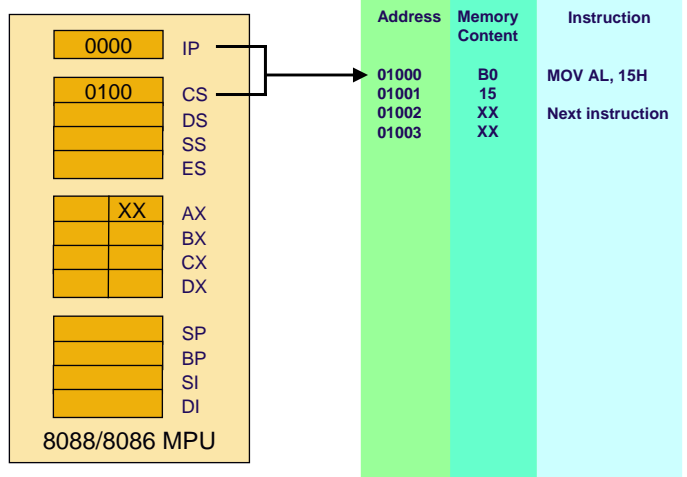


After execution



### 3.5 Addressing Modes

#### Immediate operand addressing mode

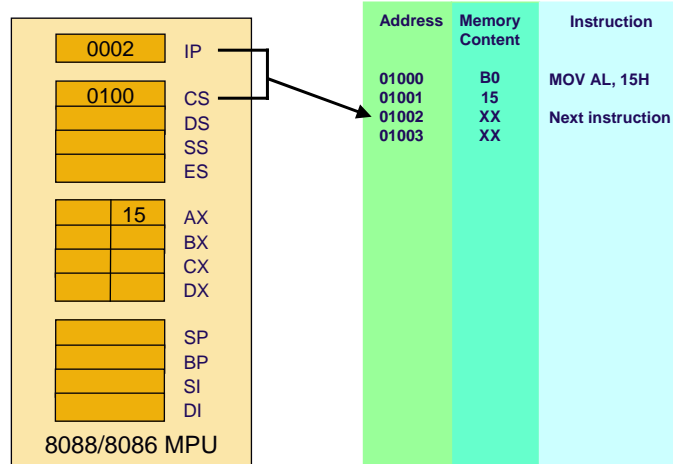


Before execution



## 3.5 Addressing Modes

### ■ Immediate operand addressing mode



After execution

611 37100 微處理機原理與應用 Lecture 03-43

國立台灣大學  
生物機電系  
林達德



## 3.5 Addressing Modes

### ■ Memory addressing modes

To reference an operand in memory, the 8088 must calculate the physical address (PA) of the operand and then initiate a read or write operation to this storage location.

Physical Address (PA) = Segment Base Address (SBA) + Effective Address (EA)

**PA = Segment Base : Base + Index + Displacement**

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

**EA = Base + Index + Displacement**

Physical and effective address computation for memory operands

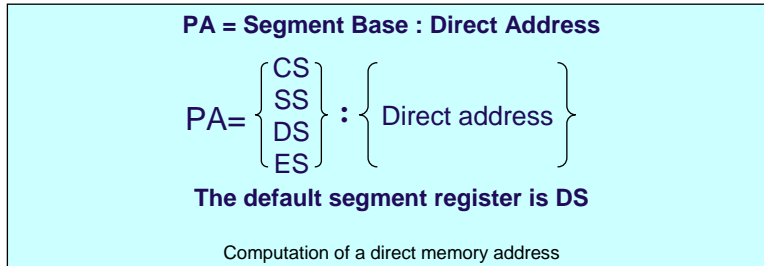
611 37100 微處理機原理與應用 Lecture 03-44

國立台灣大學  
生物機電系  
林達德



### 3.5 Addressing Modes

- Memory addressing modes - Direct addressing mode

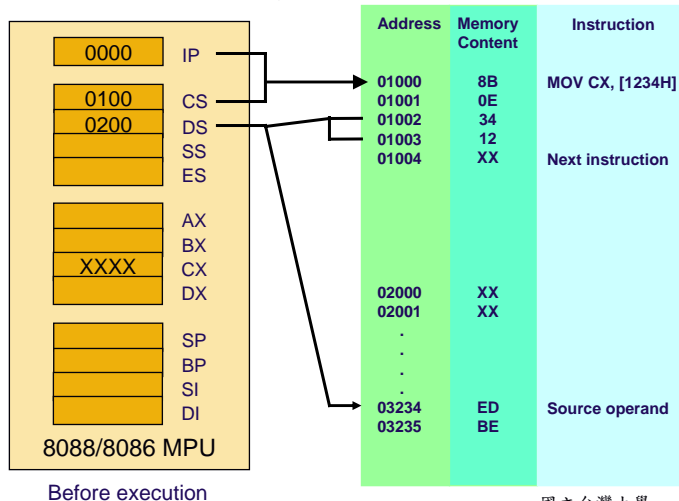


e.g. `MOV AX, [1234H]`



### 3.5 Addressing Modes

- Memory addressing modes - Direct addressing mode

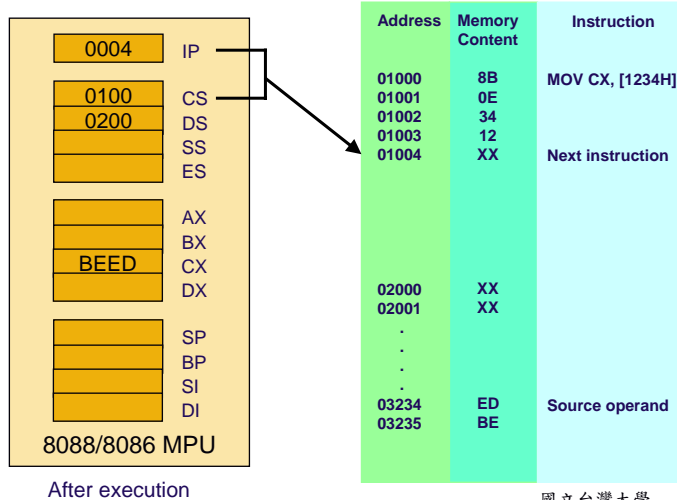


Before execution



## 3.5 Addressing Modes

### Memory addressing modes - Direct addressing mode



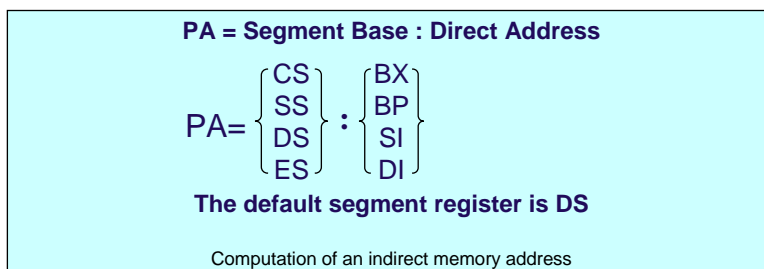
611 37100 微處理機原理與應用 Lecture 03-47

國立台灣大學  
生物機電系  
林達德



## 3.5 Addressing Modes

### Memory addressing modes - Register indirect addressing mode



e.g. MOV AX, [SI]

611 37100 微處理機原理與應用 Lecture 03-48

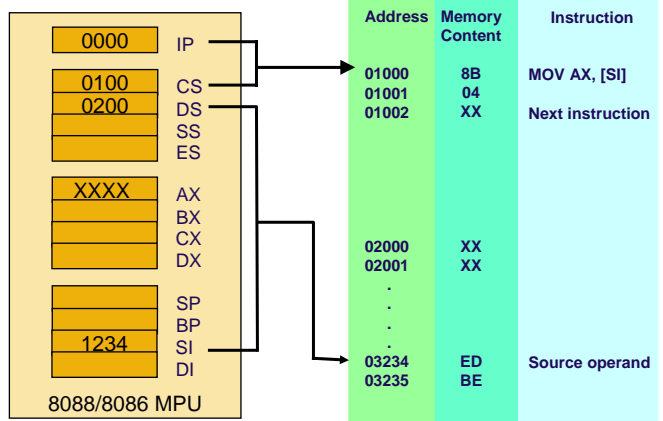
國立台灣大學  
生物機電系  
林達德



### 3.5 Addressing Modes

- Memory addressing modes - Register indirect addressing mode

$$PA = 02000_{16} + 1234_{16} = 03234_{16}$$



Before execution

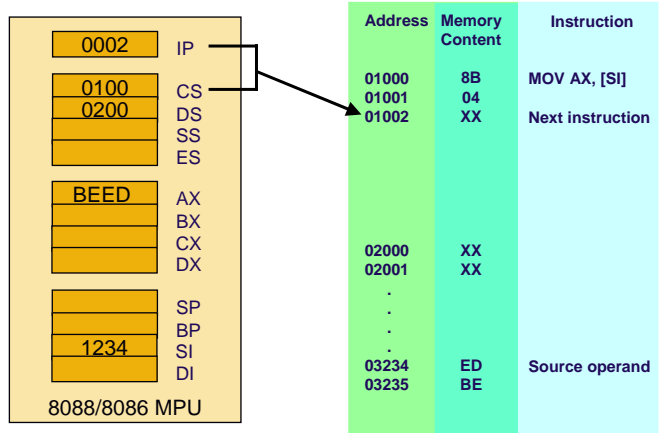
國立台灣大學  
生物機電系  
林達德



### 3.5 Addressing Modes

- Memory addressing modes - Register indirect addressing mode

$$PA = 02000_{16} + 1234_{16} = 03234_{16}$$



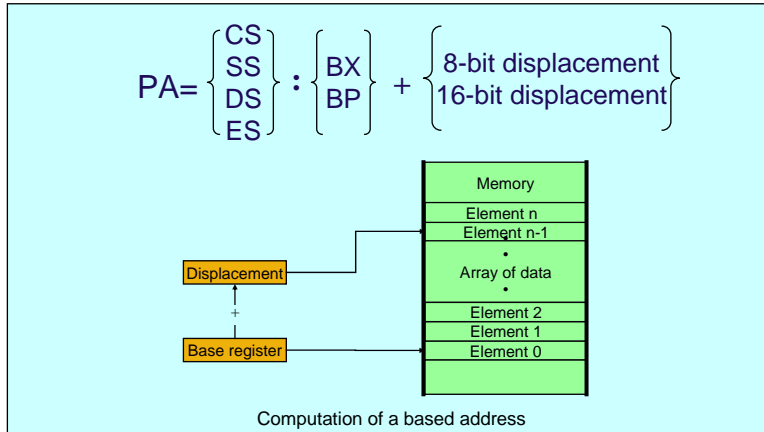
After execution

國立台灣大學  
生物機電系  
林達德



### 3.5 Addressing Modes

- Memory addressing modes - Based addressing mode



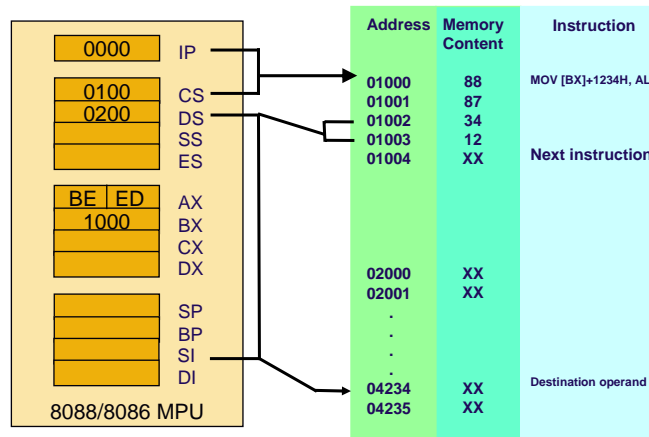
e.g. `MOV [BX]+1234H, AL`



### 3.5 Addressing Modes

- Memory addressing modes - Based addressing mode

$$PA = 02000_{16} + 1000_{16} + 1234_{16} = 04234_{16}$$



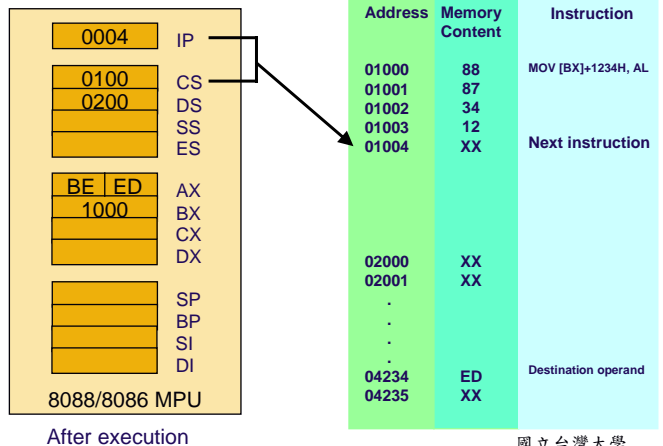
Before execution



### 3.5 Addressing Modes

■ Memory addressing modes - Based addressing mode

$$PA = 02000_{16} + 1000_{16} + 1234_{16} = 04234_{16}$$



After execution

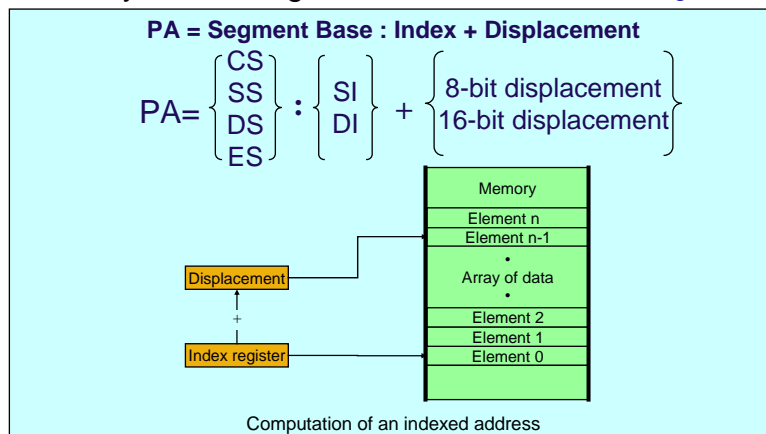


### 3.5 Addressing Modes

■ Memory addressing modes - Indexed addressing mode

$$PA = \text{Segment Base} : \text{Index} + \text{Displacement}$$

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



Computation of an indexed address

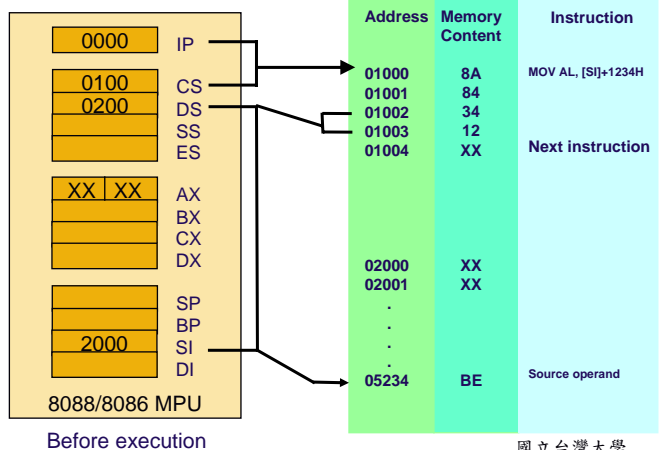
e.g. MOV AL, [SI]+1234H



### 3.5 Addressing Modes

■ Memory addressing modes - Indexed addressing mode

$$PA = 02000_{16} + 2000_{16} + 1234_{16} = 05234_{16}$$



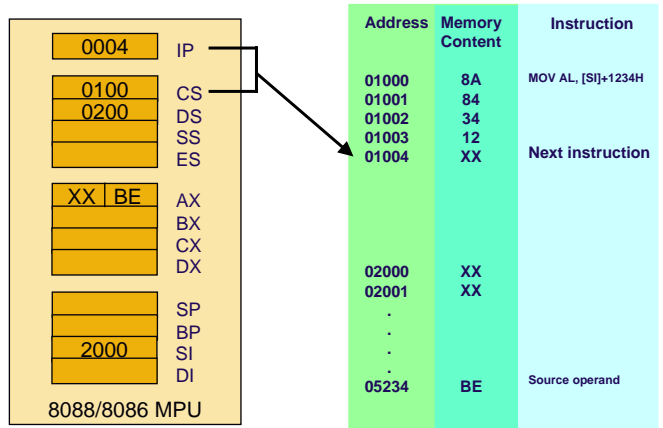
Before execution



### 3.5 Addressing Modes

■ Memory addressing modes - Indexed addressing mode

$$PA = 02000_{16} + 2000_{16} + 1234_{16} = 05234_{16}$$

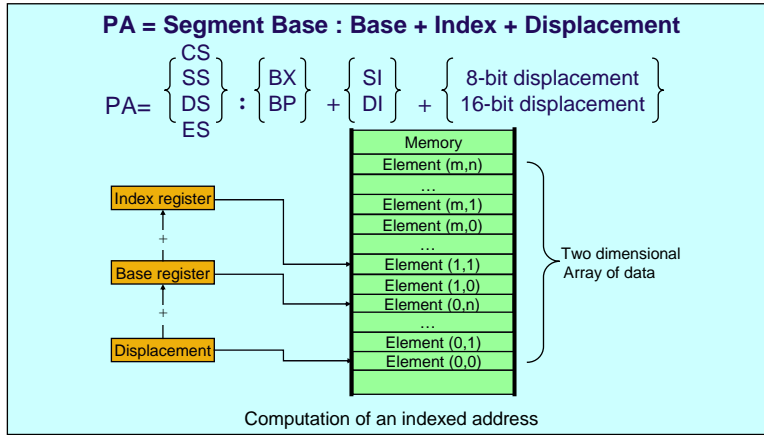


After execution



### 3.5 Addressing Modes

- Memory addressing modes - Based-indexed addressing mode



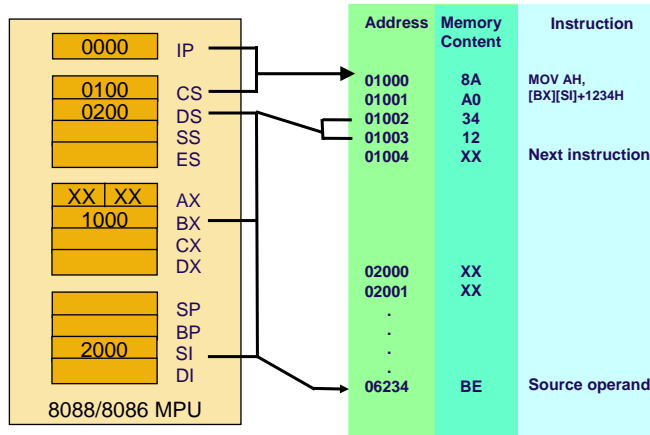
e.g. `MOV AH, [BX][SI]+1234H`



### 3.5 Addressing Modes

- Memory addressing modes - Based-indexed addressing mode

$$PA = 02000_{16} + 1000_{16} + 2000_{16} + 1234_{16} = 06234_{16}$$



Before execution

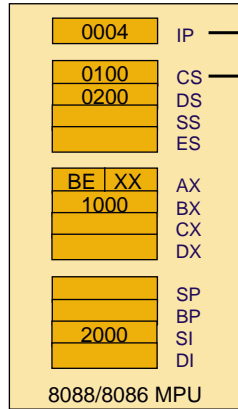




### 3.5 Addressing Modes

■ Memory addressing modes - Based-indexed addressing mode

$$PA = 02000_{16} + 1000_{16} + 2000_{16} + 1234_{16} = 06234_{16}$$



After execution

Address	Memory Content	Instruction
01000	8A	MOV AH, [BX][SI]+1234H
01001	A0	
01002	34	Next instruction
01003	12	
01004	XX	
02000	XX	Source operand
02001	XX	
.	.	
.	.	
06234	BE	

