

Microprocessor and Assembly Language

Chapter-Three Addressing Modes

Addressing modes

- Each instruction in microprocessor contains **operations** and **operands**.
- **Operation** specifies the type of action to be performed.
For example: ADD, SUB, MOV, INC, LOAD, STORE
- **Operands** are the data on which the operation is to be performed.
- **Addressing mode** indicates a **way of locating data** or **operands**.
- Depending up on the **data type used** in the instruction and the **memory addressing modes**, any instruction may belong to one or more addressing modes.
- There are several addressing (movement) instructions used in 8086. For instance;

Addressing modes

- The **data movement** instructions include **MOV**, **MOVSX**, **MOVZX**, **PUSH**, **POP**, **BSWAP**, **XCHG**, **XLAT**, **IN**, **OUT**, **LFS**, **LGS**, **LSS**, **LAHF**, **SAHF**.
- **String movement** instructions: **MOVS**, **LODS**, **STOS**, **INS** and **OUTS**.
- In this chapter, we will see how the **MOV (move data)** instruction is used to describe the data - addressing modes.
- The **MOV** instruction transfers **bytes or words of data** between **two registers** or between **registers and memory** in the 8086.
- **MOV (move data)** instruction is used to describe the data-addressing modes.
- Note: In describing program addressing modes, the **CALL** and **JUMP** instructions are used **to modify the flow of the program**.

Data addressing modes

Type	Instruction	Source	Address Generation	Destination
Register	MOV AX,BX	Register BX		Register AX
Immediate	MOV CH,3AH	Data 3AH		Register CH
Direct	MOV [1234H],AX	Register AX	$DS \times 10H + DISP$ 10000H + 1234H	Memory address 11234H
Register indirect	MOV [BX],CL	Register CL	$DS \times 10H + BX$ 10000H + 0300H	Memory address 10300H
Base-plus-index	MOV [BX+SI],BP	Register SP	$DS \times 10H + BX + SI$ 10000H + 0300H + 0200H	Memory address 10500H
Register relative	MOV CL,[BX+4]	Memory address 10304H	$DS \times 10H + BX + 4$ 10000H + 0300H + 4	Register CL
Base relative-plus-index	MOV ARRAY[BX+SI],DX	Register DX	$DS \times 10H + ARRAY + BX + SI$ 10000H + 1000H + 0300H + 0200H	Memory address 11500H
Scaled index	MOV [EBX+2 × ESI],AX	Register AX	$DS \times 10H + EBX + 2 \times ESI$ 10000H + 00000300H + 00000400H	Memory address 10700H

Notes: EBX = 00000300H, ESI = 00000200H, ARRAY = 1000H, and DS = 1000H

Data addressing modes

- In writing movement instruction, the **source** is always to the **right** and the **destination** is to the **left**.



- An **opcode**, or operation code, tells the microprocessor **which operation** to perform.
- We naturally assume that things move from left to right, whereas here they move from right to left.
- Notice that a **comma always separates the destination from the source** in an instruction.
- Also, note that memory-to-memory transfers are **not allowed** by any instruction except for the **MOVS** instruction.

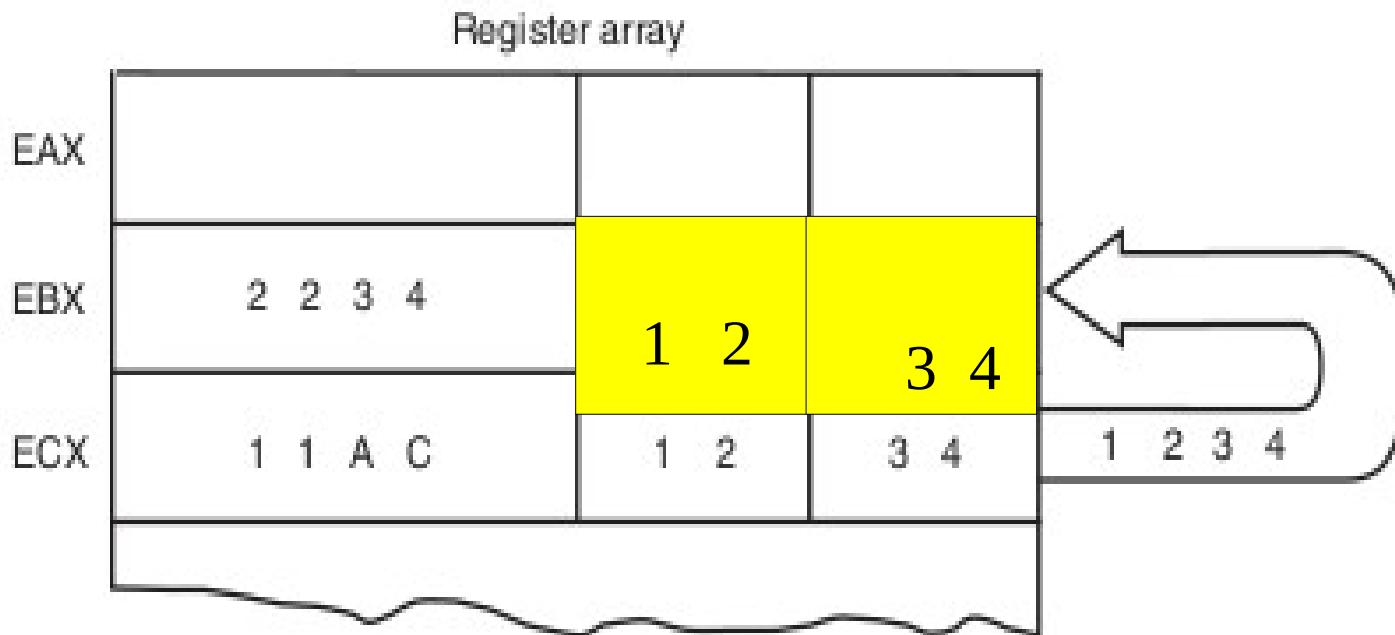
Data addressing modes(Cont...)

- The different data-addressing modes using the MOV instruction are explained here below.
- ➔ **Register addressing** transfers a **copy of byte or word** from the source register(content of memory) to the destination register.
 - ☞ E.g **MOV CX, DX** copies the **word-sized contents** of **DX** into **CX**. (in 16bit architectures, 8086)
- In the **80386 and above**, a **double-word** can be transferred from the source to the destination as follows.
 - ☞ E.g: **MOV ECX, EDX** copies the **double-word sized contents** of register **EDX** into register **ECX**.
- In the Pentium 4, operated in the **64-bit** mode, any 64-bit register is also allowed.
 - ☞ E.g: **MOV RDX, RCX** copy the **quad-word contents** of register **RCX** into register **RDX**

Data addressing modes(Cont...)

- Note when we perform MOV instruction, the **value of source register will not change** while the value of destination register changes.

Look below example how the operation of the **MOV BX, CX** instruction changes the destination register's contents. This instruction moves (copies) a 1234H from register CX into register BX. This **erases the old contents** (76AFH) of register **BX**, but the contents of **CX** remain **unchanged**.



Data addressing modes(Cont...)

➔ **Immediate addressing** transfers the source, an **immediate data**(byte, word, double word, or quad-word) into the destination register(memory location).

☛ E.g **MOV AL, 22H** copies a **byte-sized** 22H into register AL.

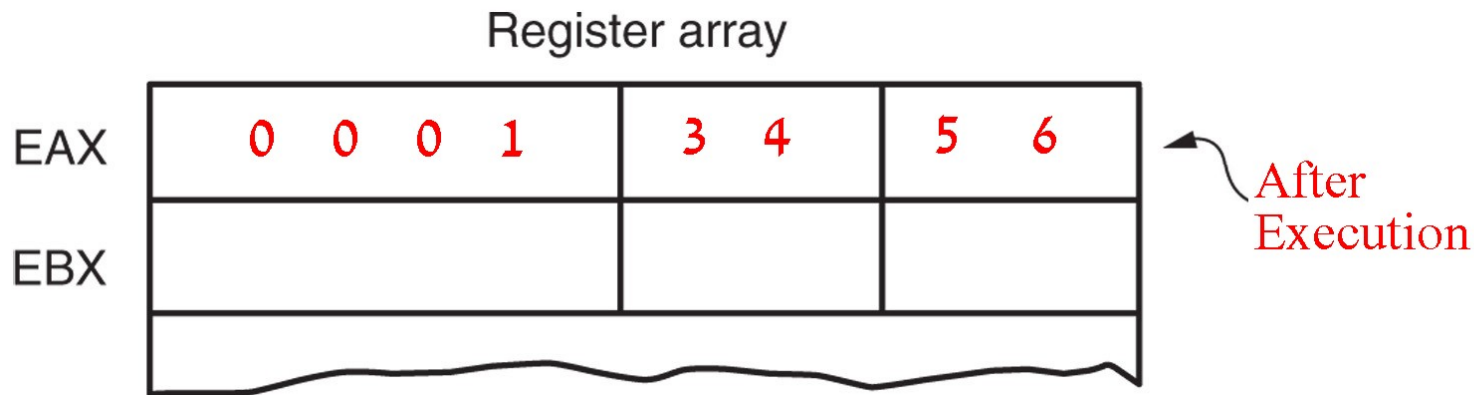
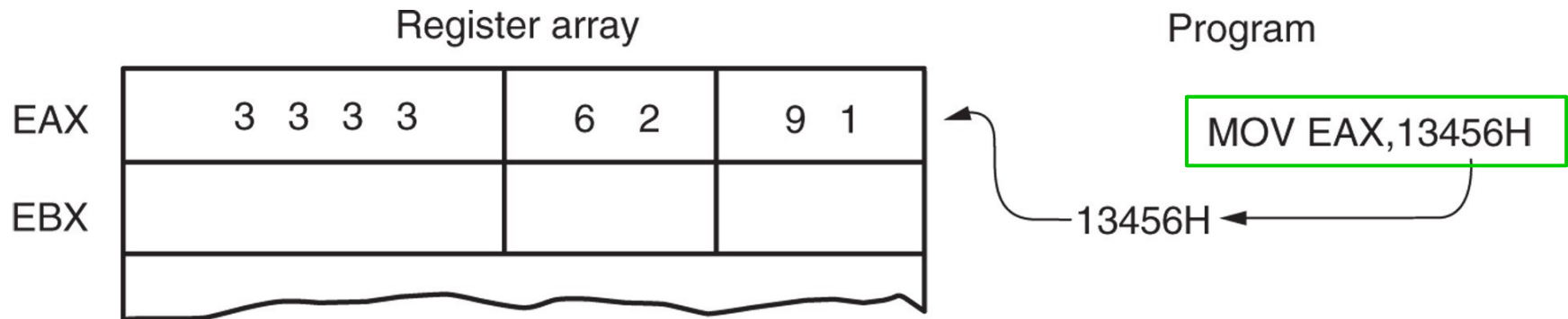
In the **80386 and above**, a **double-word** of immediate data can be transferred into a destination.

☛ E.g: **MOV EBX, 12345678H** copies a **double-word** sized 12345678H into EBX

Note that: the following example is also **valid** instruction.

MOV EAX, 13456H //ofcourse it looks likes a size mismatch but, the assembler by default assign a **000** value to the left most bytes and will make it(the source immediate data) a 16 bit. That is,

Data addressing modes(Cont...)



- As with the MOV instruction illustrated in above figure, the source data **overwrites** the destination data.

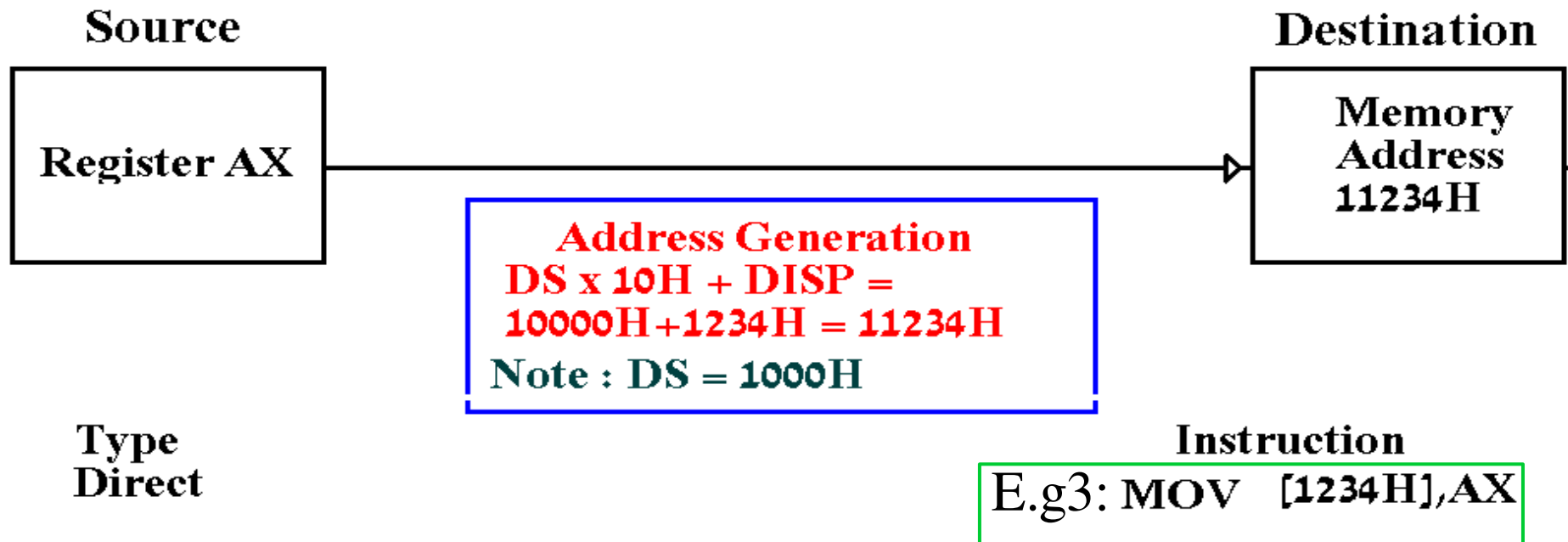
Data addressing modes(Cont...)

➔ **Direct addressing** moves a **byte or word** between a memory location and a register.

E.g1: **MOV CX, LIST** copies the **word-sized** contents of **memory location LIST** into register **CX**.

✓ In the **80386 and above**, a **double-word** of memory location can be transferred into a destination.

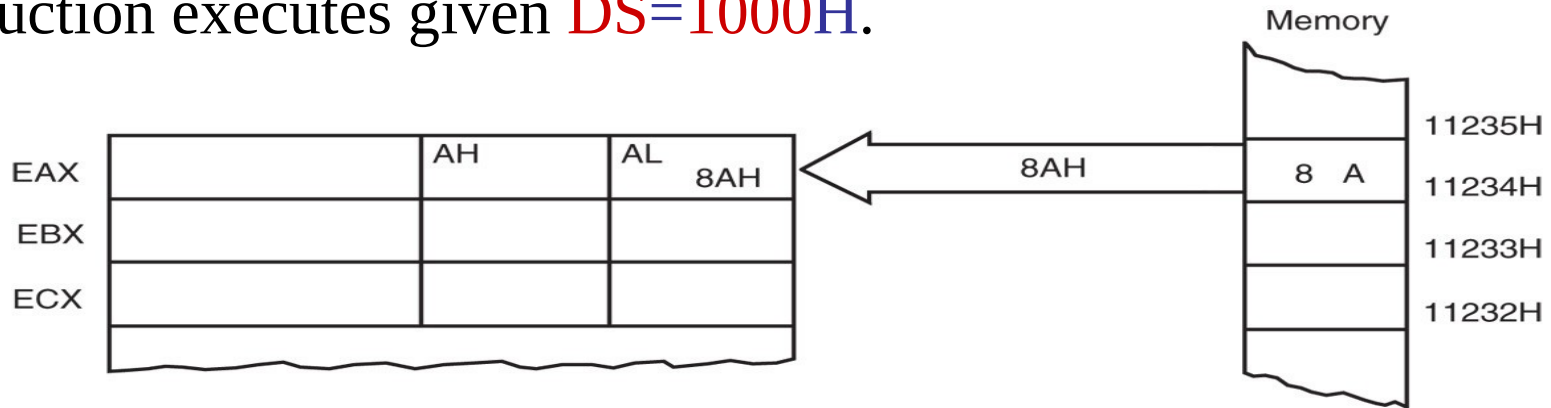
E.g2: **MOV ESI, LIST**



Locating Data with direct Addressing(Cont...)

- On the above E.g3, the address is formed by adding the **displacement** to the default **data segment (DS)** address or an **alternate segment** address.

E.g4: Look what happen when the operation of the **MOV AL,[1234H]** instruction executes given **DS=1000H**.



- This instruction transfers a **copy** contents of memory location **11234H** into **AL**.
 - the effective address is formed by adding 1234H (the **offset address**) and 10000H (the **data segment** address of 1000H times 10H) in a system operating in the **real mode**.

Note the difference between the following two:

MOV AX, 1234H

and

MOV AX, [1234H]

- The square brackets around the 1234H denotes the contents of the memory location.
- The **data segment**(DS is used by **default** with register indirect addressing or any other mode that uses **BX**, **DI**, or **SI** to address memory.

Data addressing modes(Cont...)

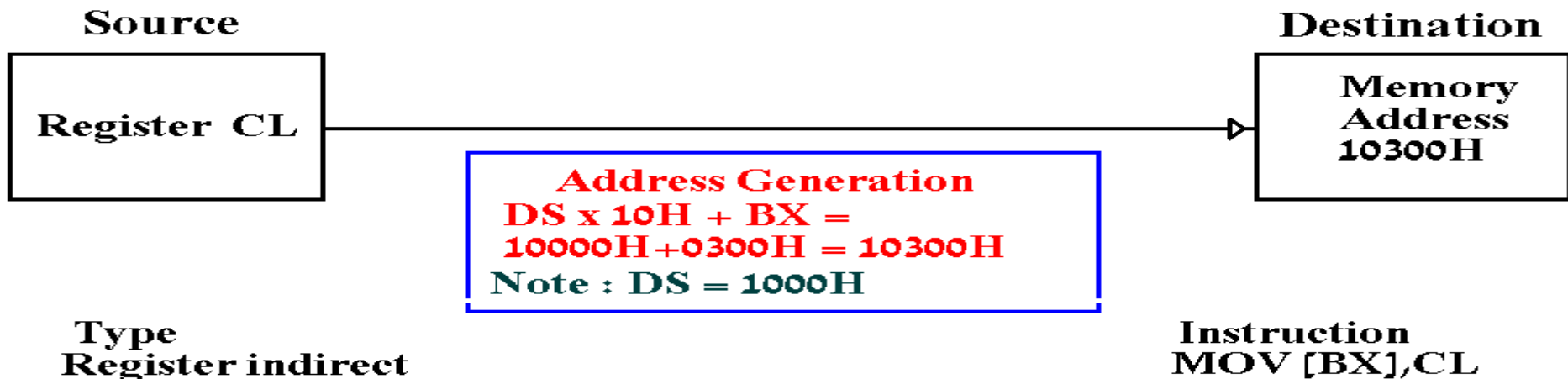
➔ **Register indirect addressing** transfers a byte or word between a **register** and a **memory location addressed by an index or base register**.

- The index and base registers are BP, BX, DI, and SI.

E.g1 **MOV AX, [BX]** copies the word-sized data from the data segment offset address indexed by BX into register AX.

E.g2 **MOV [BX], CX**

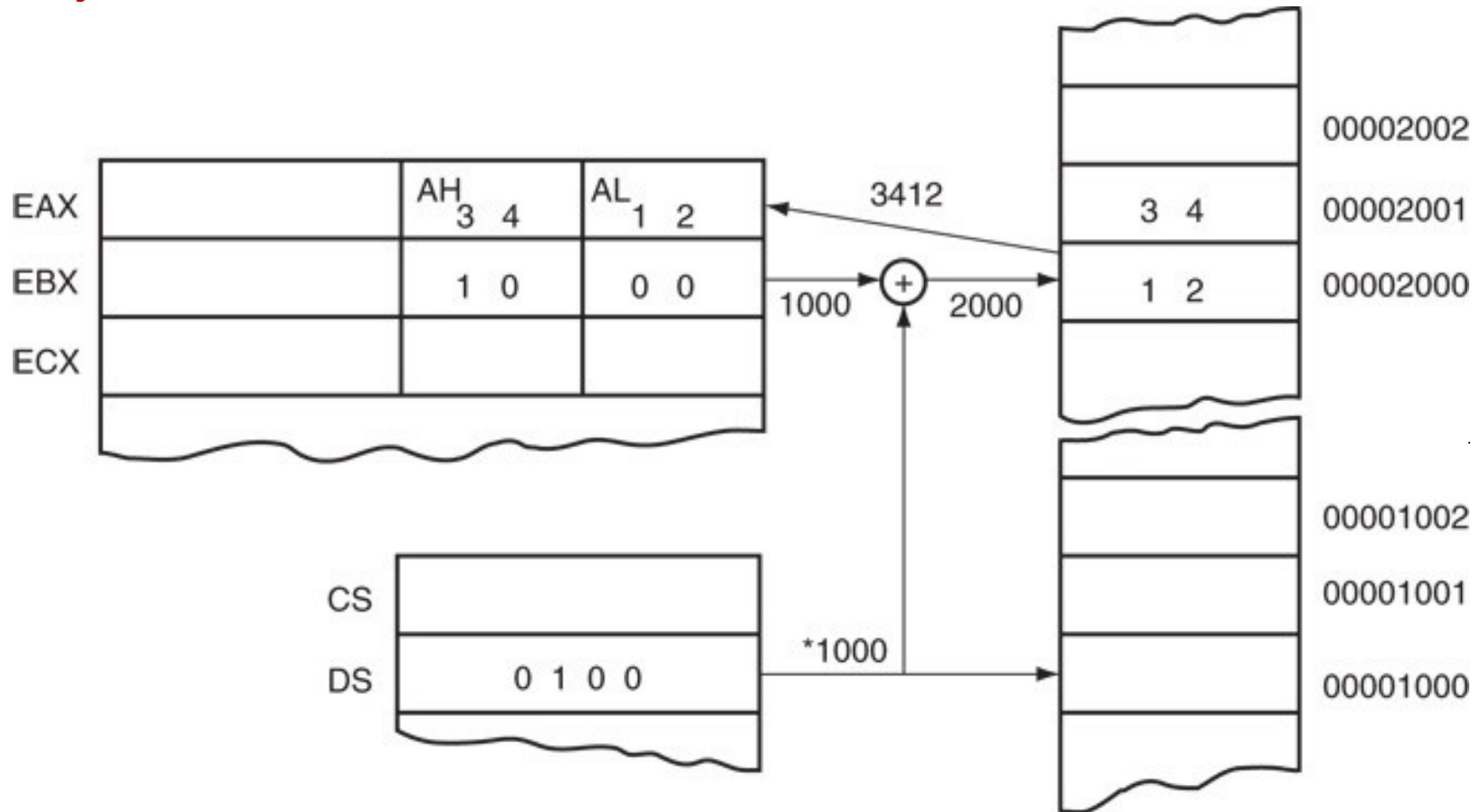
Destination Address = $DS \times 10H + BX$, let the $BX=0300H$ now:



Locating Data with register indirect Addressing

Look the following example of the **MOV AX,[BX]** instruction when **BX = 1000H** and **DS = 0100H**. Note: this instruction is shown after the contents of memory are transferred to AX.

Memory location = DS x 10 + BX = 01000+1000 = 00002000 and 00002001



*After DS is appended with a 0.

Note the difference between the two;

MOV BX , CX

and

MOV [BX],CX

- In 8086 **each memory location is only of a byte(8 bits) size**, therefore we need two different memory locations to represent 16 bit data.
- Note that: MOV AX, AL or MOV EAX, AL instructions are **not allowed** because the registers are of different sizes.

Data addressing modes(Cont...)

→ **Base-plus-index addressing** transfers a byte or word between a register and the memory location addressed by a base register (BP or BX) plus an index register (DI or SI).

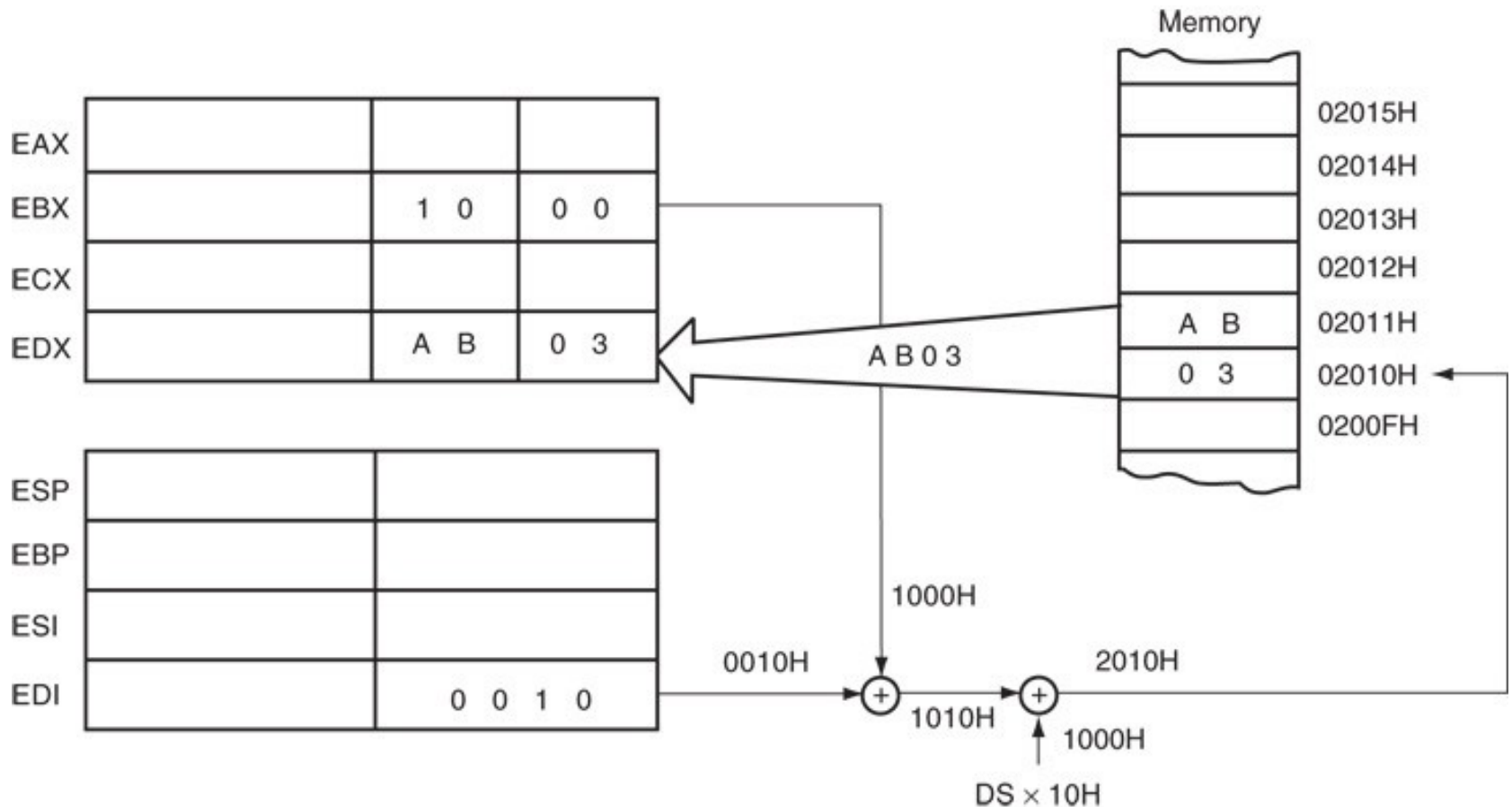
E.g1 `MOV [BX+DI], CL` copies the byte-sized contents of register CL into the data segment memory location addressed by BX plus DI.

- Similar to **indirect addressing** because it **indirectly** addresses memory data.
- The **base register** often holds the **beginning** address of a memory array.
- The **index register** holds the **relative position** of an element in the **array** whenever **BP** addresses memory data, **both** the **stack segment** register and **BP** generate the effective address

Locating Data with Base-Plus-Index Addressing

Below is an example of how the **base-plus-index** addressing mode functions for the **MOV DX, [BX + DI]** instruction. Notice that memory address 02010H is accessed because we assumed **DS=0100H**, **BX=1000H** and **DI=0010H**.

Address = DS x 10 + BX + DI = 01000+01000+0010 = 02010H and 02011H



Data addressing modes(Cont...)

- ➔ **Register relative addressing** moves a byte or word between a **register and the memory location** addressed by an index **or** base register **plus** a displacement.
- **Similar** to base-plus-index addressing and displacement addressing data in a segment of memory are addressed by **adding** the **displacement** to the contents of a **base** or **index register** (BP, BX, DI, or SI).

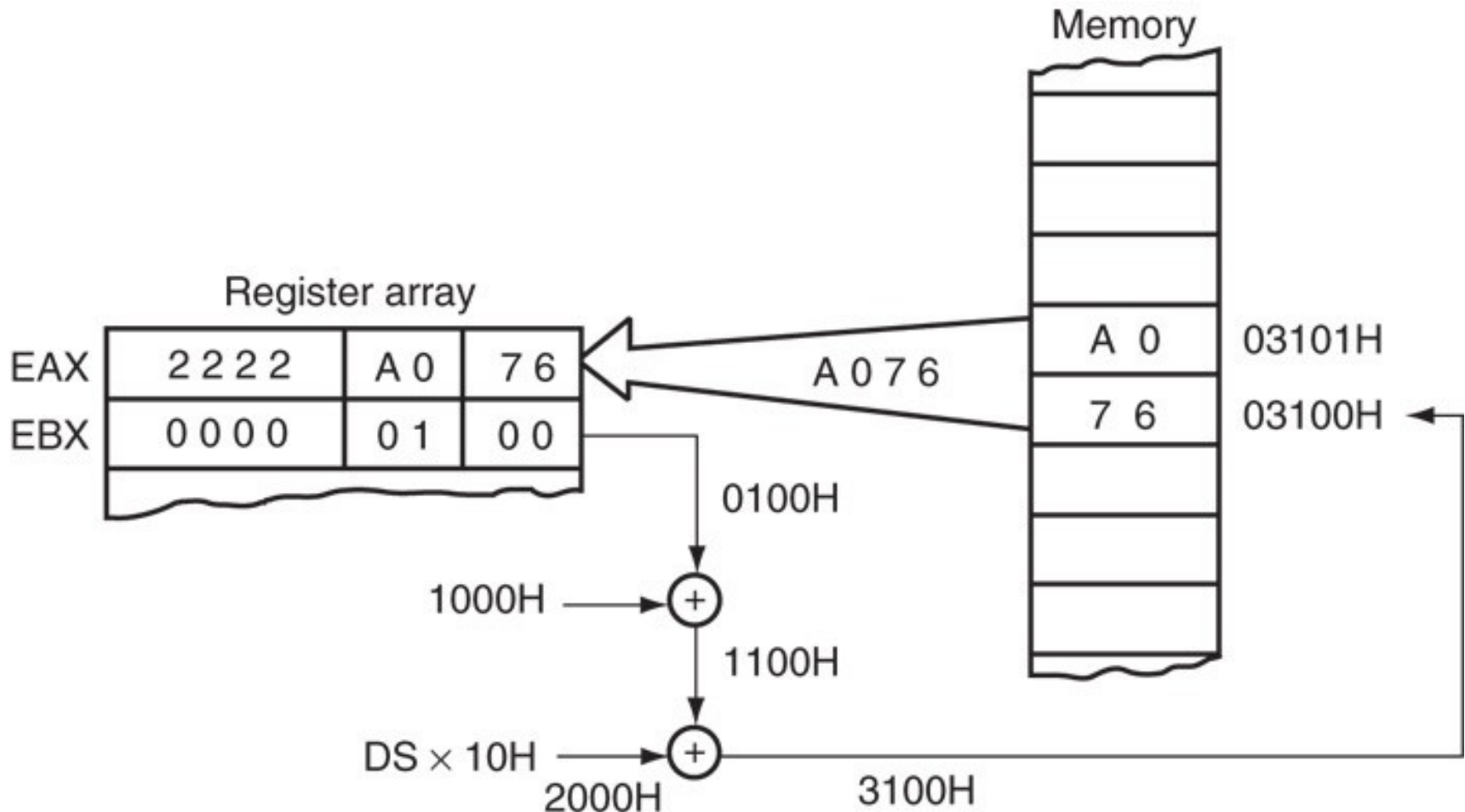
E.g **MOV AX, [BX+4]** This instruction loads AX from the data segment address formed by BX plus 4. or

MOV AX, ARRAY[BX]. This instruction loads AX from the data segment memory location in ARRAY plus the contents of BX.)

Locating Data with Register relative addressing

Below is operation of **MOV AX, [BX+1000H]** instruction, when **BX=0100H** and **DS=0200H**.

Address Generation = **DS X 10H + [BX + offset]** = **02000 + 100 + 1000 = 3100H** and **3101H**



Data addressing modes(Cont...)

- ➔ **Base relative-plus-index addressing** transfers a byte or word between a **register and the memory location** addressed by a base **and** index register **plus** a displacement.
- Similar to base-plus-index addressing.
 - ✓ adds a **displacement**
 - ✓ uses a **base register** and an **index register** to form the **memory address**.
- ☛ E.g **MOV AX, ARRAY [BX+DI]** or **MOV AX, [BX+DI+4]**.

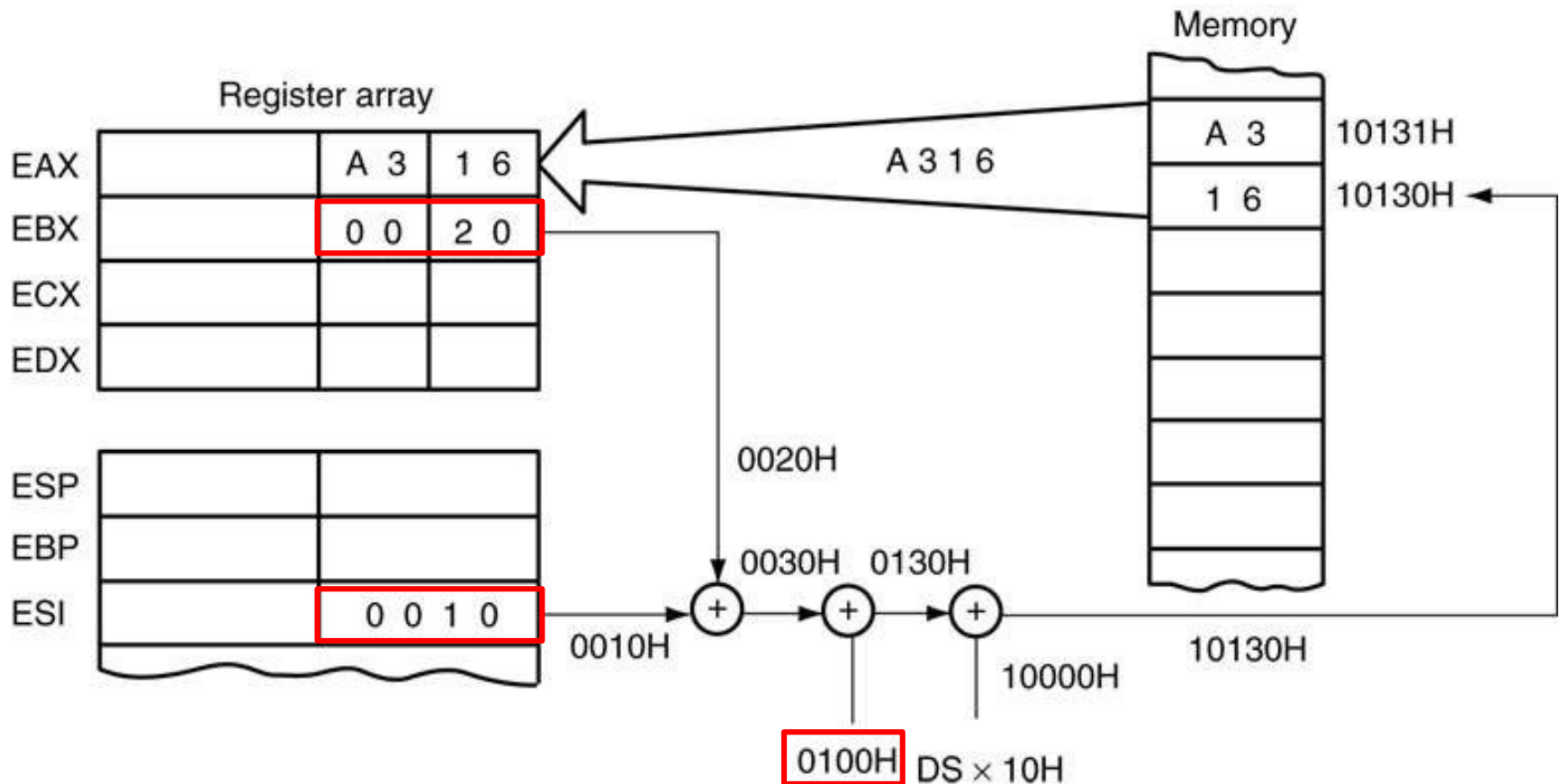
These instructions load AX from data segment memory location.

The first instruction uses an address formed by adding ARRAY, BX, and DI and the second by adding BX, DI, and 4.)

Locating Data with Base-relative-plus-index addressing

Below is an example of base relative-plus-index addressing using a **MOV AX,[BX+SI+100H]** instruction.

Note: we have used **DS=1000H**, **BX=0020H** and **SI=0010H** and as shown below.



Data addressing modes(Cont...)

- ➔ **Scaled-index addressing** is available only in the 80386 through the Pentium 4 microprocessor.
- The second register of a pair of registers is modified by the scale factor of **2x**, **4x**, **8x** to generate the operand memory address. That is;
 - The second register (**index**) is **multiplied** by a **scaling factor**.
 - ✓ The **scaling factor** can be **1x**, **2x**, **4x**, **8x**
 - ☞ E.g **MOV EDX, [EAX + 4 x EBX]** loads EDX from the data segment memory location addressed by EAX plus 4*EBX.

Program Memory Addressing Modes

- As we have seen earlier, **MOV** (move data) instruction will be used to describe the data addressing modes.
- But what is program memory addressing?
- The **Program Memory Addressing modes** are those that are used in **branch instructions**. These branch instructions are instructions which are responsible for **changing the regular flow of the instruction execution** and **shifting the control to some other location**.
- **CALL** and **JUMP** instructions will be used to describe program memory addressing modes.
- There are also **PUSH** and **POP** instructions which will be used to describe the **operation of the stack memory**.

Program Memory Addressing (Cont...)

- The **Program Memory Addressing modes** have three distinct forms:
 1. Direct program memory addressing,
 2. Relative program memory addressing, and
 3. Indirect program memory addressing.
- This section introduces these three addressing forms, using the **JMP** instruction to illustrate their operation.

Direct Program Memory Addressing

- Its used by early microprocessors for all **jumps** and **calls**.
- The instructions for direct program memory addressing store the address with the opcode.
- E.g, if a **program jumps** to memory location **10000H** for the next instruction, the address (10000H) is stored following the opcode in the memory.
- The **opcode for jump** instruction is **JMP**. Figure below shows the direct intersegment JMP instruction and the 4 bytes required to store the address 10000H.
- This JMP instruction(as shown below) loads **CS** with **1000H** and **IP** with **0000H** to jump to memory location **10000H** for the next instruction.

FIGURE 3-14 The 5-byte machine language version of a JMP [10000H] instruction.



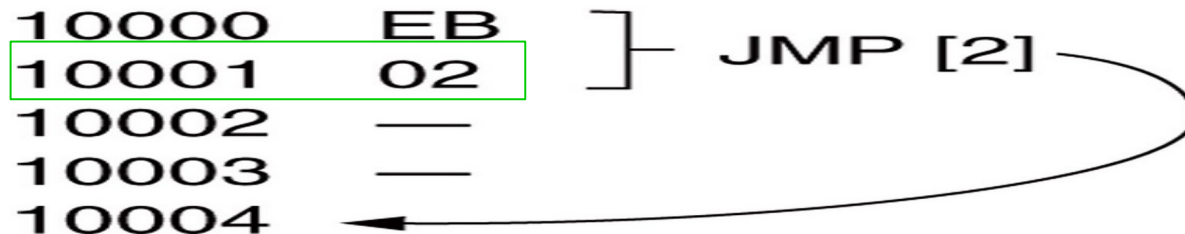
Note: This is for 16bit microprocessor, real mode.

○ Cont....

- An **intersegment jump** is a jump to any memory location (segment) within the entire memory system. Also called **direct/far jump**.
- The **direct jump** is often called a **far jump** because **it can jump to any memory location for the next instruction**.
- In the **real mode**, a far jump accesses any location within the first **1M byte** of memory by changing both CS and IP.
- In **protected mode operation**, the far jump accesses a **new code segment descriptor** from the descriptor table, allowing it to jump to any memory location in the entire **4G-byte** address range.
- Likewise in **64-bit mode** for the Pentium 4 and above, jump can be to any memory location in the system. The CS is still used, but not for the address of the jump. The CS register contains pointer to a new descriptor from descriptor table, but not the address of the jump or call.
- 64-bit processor can handle more data than 32-bit processor, it can access
- 4 billion times the physical memory of 32-bit processor.

Relative Program Memory Addressing

- Relative program memory addressing is **not available in all early** microprocessors, but it is available to this family of microprocessors(8086 and above).
- The term relative means “**relative to the instruction pointer (IP).**”
- For example, if a JMP instruction skips the next **2 bytes** of memory, the **address in relation to the instruction pointer is a 2 that adds to the instruction pointer.** This develops the address of the next program instruction.



This is a **JMP [2]** instruction(**IP=10001H** as indicated from above). This instruction skips over the **2 bytes** of memory that follow the JMP instruction (hence accesses **10004H**).

Relative Program Memory Addressing(Cont...)

- As we can see from the above, the relative JMP[2] instruction jumps over the next two bytes of memory from 10001 and access 10004.
- ✓ Notice that the JMP instruction is a **1-byte instruction**, with a 1-byte or a 2-byte displacement that adds to the instruction pointer.
- ✓ This program memory addressing schemes are classified as an **Intra-segment jumps**.
- An **intra-segment jump** is a jump to anywhere within the current code segment.
- There are two types:
 1. short jump and
 2. near jump.

Relative Program Memory Addressing(Cont...)

- A **1-byte displacement** is used in **short jumps**, and **2-byte displacement** is used with **near jumps** and **calls**.
- An 8-bit displacement (short) has a jump range of between ± 127 bytes from the next instruction;
- A 16-bit displacement (near) has a range of $\pm 32K$ bytes.
- In the 80386 and above, a 32-bit displacement allows a range of $\pm 2G$ bytes.
- That is the 80386 and above, the displacement can also be a 32-bit value, allowing them to use relative addressing to any location within their 4G-byte code segments.

Indirect Program Memory Addressing

- The microprocessor allows several forms of **program indirect memory addressing** for the **JMP** and **CALL** instructions.
- It can be an **inter-segment indirect jump** or an **intra-segment indirect jump**(depends on the value of segment register).
- As the name suggests, in this addressing mode, the **address is not present directly in the instruction**.
- It is rather stored in any of the register (AX, BX, CX, DX, SP, BP, DI, or SI); any relative register([BP], [BX], [DI], or [SI]);. So, the contents of the Instruction Pointer (IP) will be replaced by the contents of that register.
- In **80386** and **above**, an **extended register** can be used to **hold** the address or indirect address of a relative JMP or CALL.

Indirect Program Memory Addressing

For example, the **JMP EAX** jumps to the location addressed by register **EAX**.

- That is if EAX contains 12345678H and if JMP EAX executes, the microprocessor jumps to the offset address 12345678H.
- However, if a relative register holds the address, the jump is also considered to be a typical **indirect jump(indirect-indirect or double-indirect Jump)**.

For example, **JMP [BX]** refers to the memory location within the data segment at the **offset** address **contained** in **BX**.

- In this case the Jump address will be found by using the values of segment register(DS) and offset address(BX in this case).

◉ Indirect Program Memory Addressing(Cont...)◉

- That is, this **offset address** is a **16-bit** number: used as the offset address in the intrasegment jump.
- This type of jump is sometimes called an *indirect-indirect* or *double-indirect* jump.

- More Examples;

***JMP AX** Jumps to the current code segment location addressed by the contents of AX*

***JMP CX** Jumps to the current code segment location addressed by the contents of CX*



Stack Memory addressing modes

- The stack plays an important role in all microprocessors.
 - holds **data** temporarily and stores **return addresses** used by procedures.
- Stack memory is LIFO (**last-in, first-out**) memory.
 - describes the way data are stored and removed from the stack.
 - Similar to **stack of plates**.



Stack Memory		
Stack Segment		
		0000
		0001
	≡	≡
Stack top	24	FFFC
	63	FFFD
	54	FFFE
	11	FFFF

Stack Memory addressing modes(Cont...)

- In stack memory, the last location where the data is stored is called **stack top**. And the address found at the stack top is **stack top address**.
- Note: On the previous slide the contents shown in left side (yellow colored) are known as **offset address**.
- The offset address of stack top is stored inside **stack pointer register**.
- Data are placed on the stack with a **PUSH instruction**; and removed with a **POP instruction**.
- The **Call** instruction also uses the stack to **hold** the **return address** for the procedures and a **RET** (return) instruction to **remove** the **return address** from the stack.

Stack Memory addressing modes(Cont...)

- Thus, stack memory is maintained by **two** registers:
 - **Stack Pointer** register (SP or ESP): Used to hold offset address of stack top.
 - **Stack Segment** register (SS): store information about the memory segment that stores the call stack of currently executed program.
- As we have seen in our previous chapter, in 8086 mp the offset is a 16-bit address.
- Look how stack pointer stores offset address of the stack top:



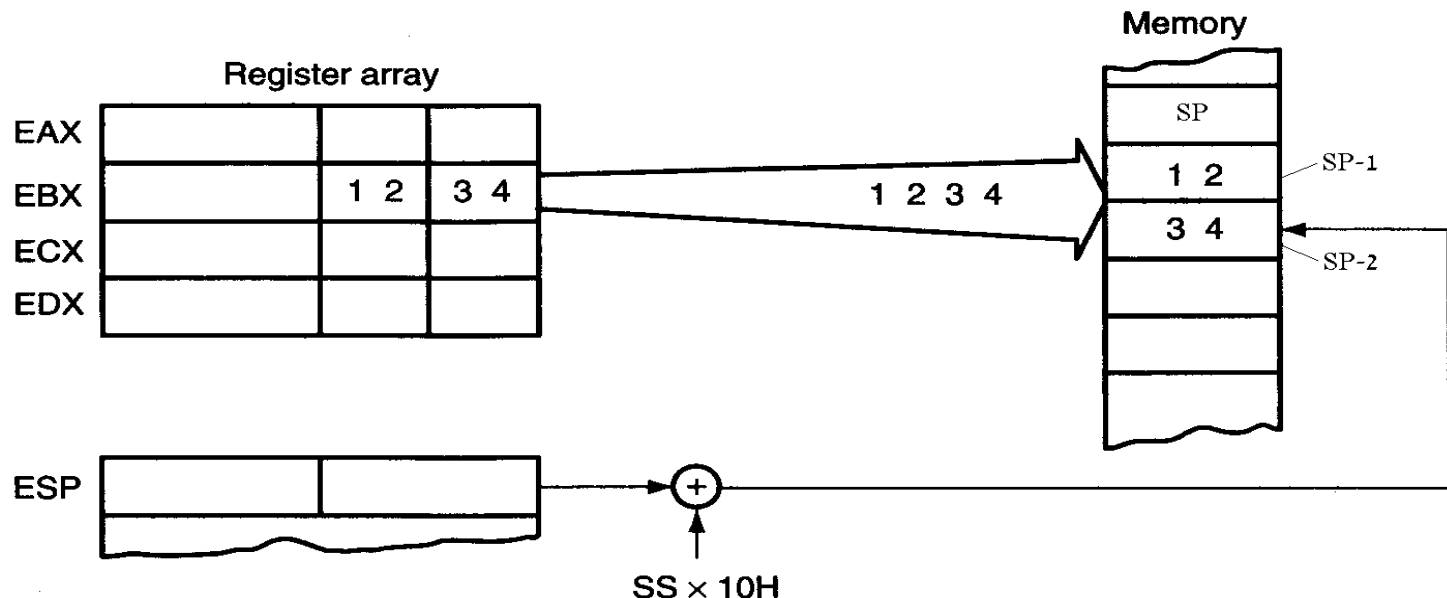
Stack Memory addressing modes(Cont...)

- As we have seen in earlier, in protected mode operation, the **SS** register holds a **selector** that accesses a **descriptor** for the base address of the stack segment.

(a) **PUSH BX** places the contents of **BX** onto the stack;

Whenever a **word of data is pushed** onto the stack,

- > The **high-order** 8 bits are placed in the location addressed by **SP - 1**
- > The **low-order** 8 bits are placed in the location addressed by **SP - 2**



Stack Memory addressing modes(Cont...)

- Then the **SP** is **decremented** by **2** so the next word is stored in the next available stack location.

Example2: What happens when **PUSH DX** executes having the following informations and **DX = 3401H**?

Stack Memory	
Stack Segment	
FFFC	24
FFFD	63
FFFE	54
FFFF	11

Stack Memory addressing modes(Cont...)

Solution: $DX = 3401$ means, $DH=34$ and $DL = 01$.

Then, the $SP = FFFC$.

DH would be stored at $SP-1 = FFFC-1 = FFFB$.

DL would be stored at $SP-2 = FFFC-2 = FFFA$. I.e,

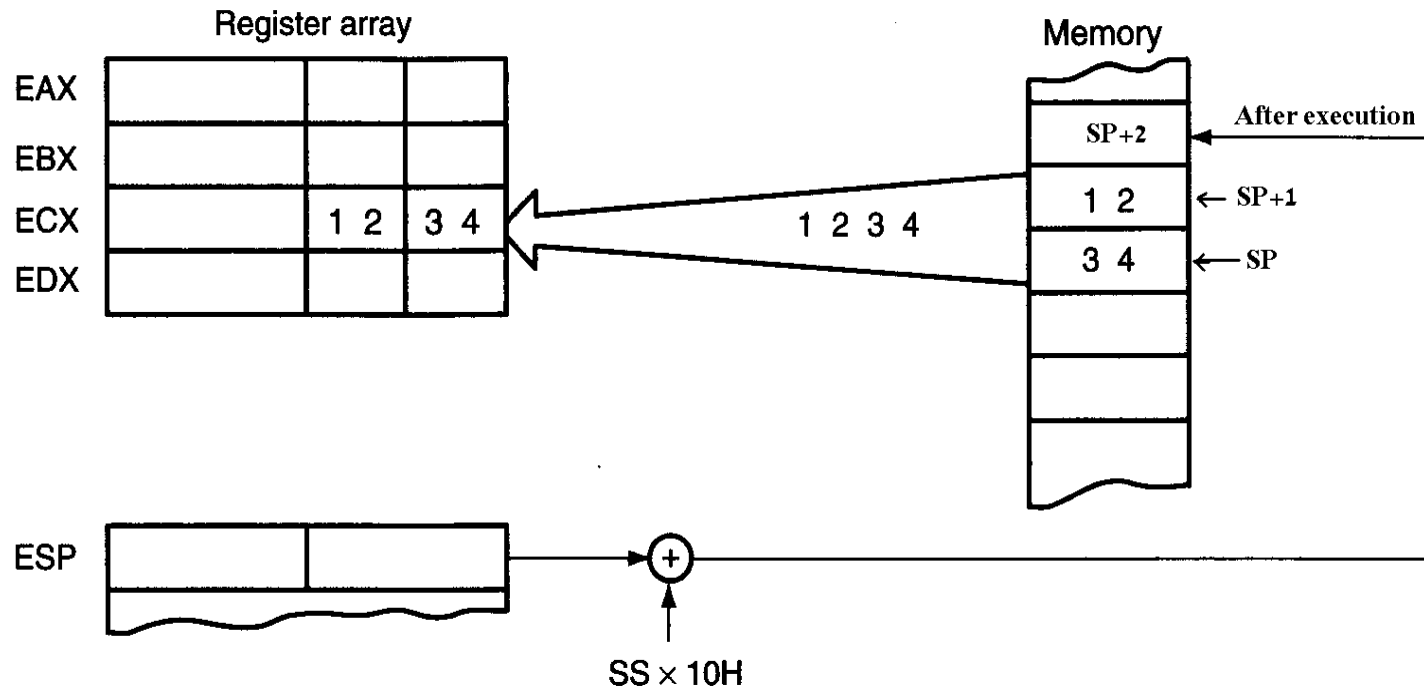
Stack Memory	
Stack Segment	
FFFA	01
FFFB	34
FFFC	24
FFFD	63
FFFE	54
FFFF	11

Stack Memory addressing modes(Cont...)

(b) **POP CX** removes data from the stack and places them into **CX**.
Instruction is shown after execution.

When data are popped from the stack,

- The **low-order** 8 bits are removed from the location addressed by **SP**.
- The **high-order** 8 bits are removed from the location addressed by **SP+1**;



Stack Memory addressing modes(Cont...)

- Then the **SP** is **incremented** by **2** so the next word is stored in the next available stack location.

Example2: What happens when **POP CX** executes having the following informations?

Stack Memory	
Stack Segment	
FFFC	24
FFFD	63
FFFE	54
FFFF	11

Stack Memory addressing modes(Cont...)

Note that: the value of CX need not have to be given since its known that it stores 16bit value, on CL and CH.

- **Solution:** know again the $SP = FFFC$.

Step1: **CL** would be removed from the location specified by **SP**, i.e **FFFC**. Since POP starts from stack top. Then,

Step2: **CH** would be removed from the location specified by **SP+1**.

I.e, $FFFC+1 = FFFD$.

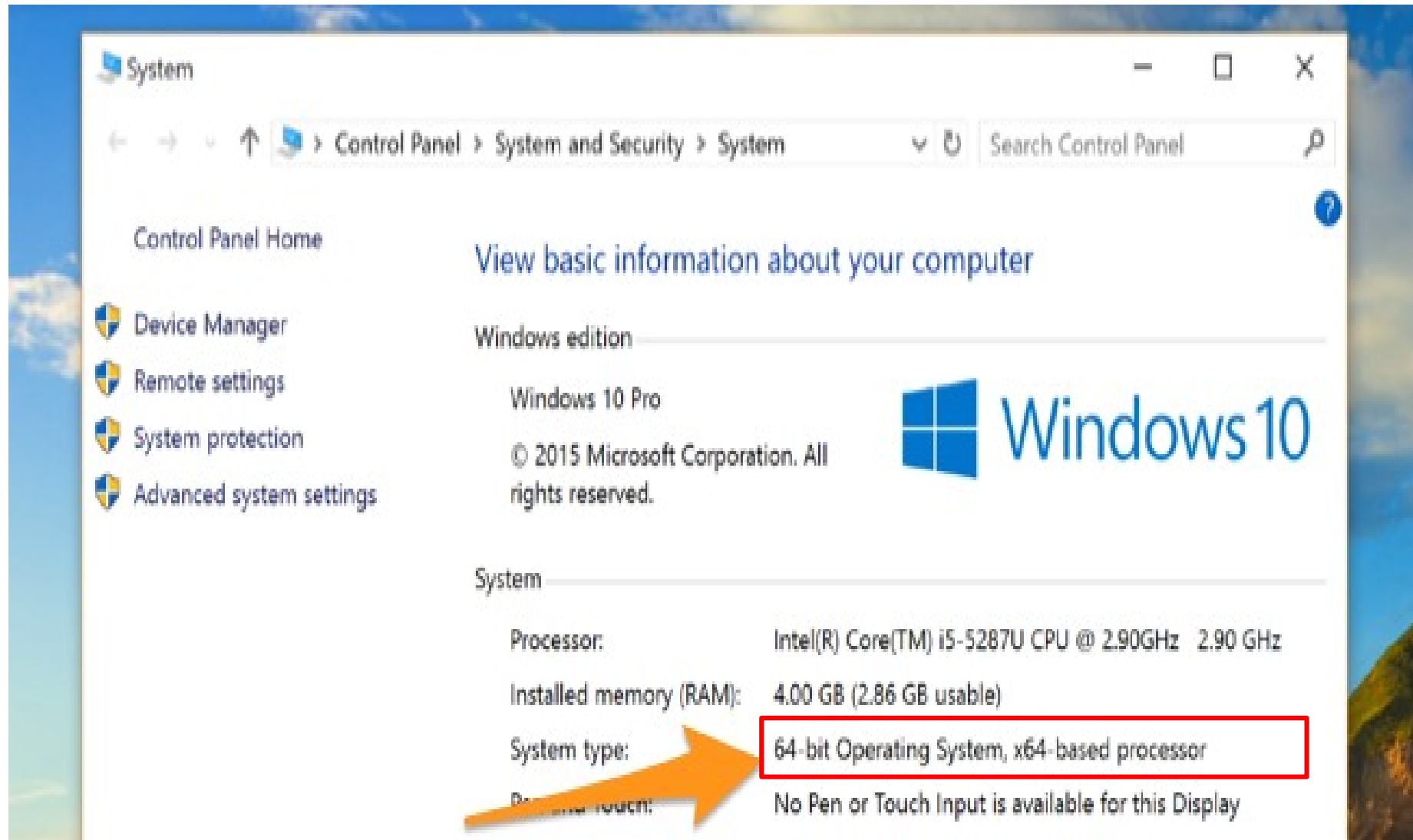
Stack Memory	
Stack Segment	
FFFC	
FFFD	63
FFFE	54
FFFF	11

Stack Memory	
Stack Segment	
FFFC	
FFFD	
FFFE	54
FFFF	11

Stack Memory addressing modes(Cont...)

- Note that PUSH and POP store or retrieve **words** of data - **never bytes** - in 8086 - 80286.
- 80386 and above allow **words** or **doublewords** to be transferred to and from the stack.
- **Data** may be **pushed** onto the **stack** from **any 16-bit** register or **segment** register in 8086/80286 .
 - And in 80386 and above, from any **32-bit** extended register.

Need to know about your system?



The screenshot shows the Windows 10 'System' window. The title bar reads 'System'. The breadcrumb navigation shows 'Control Panel > System and Security > System'. The left sidebar contains links to 'Control Panel Home', 'Device Manager', 'Remote settings', 'System protection', and 'Advanced system settings'. The main content area is titled 'View basic information about your computer'. Under 'Windows edition', it shows 'Windows 10 Pro' and '© 2015 Microsoft Corporation. All rights reserved.' with the Windows 10 logo. Under 'System', it lists: 'Processor: Intel(R) Core(TM) i5-5287U CPU @ 2.90GHz 2.90 GHz', 'Installed memory (RAM): 4.00 GB (2.86 GB usable)', 'System type: 64-bit Operating System, x64-based processor' (highlighted with a red box and an orange arrow), and 'Pen and touch: No Pen or Touch Input is available for this Display'.

System

Control Panel > System and Security > System

Control Panel Home

View basic information about your computer

Windows edition

Windows 10 Pro

© 2015 Microsoft Corporation. All rights reserved.

System

Processor: Intel(R) Core(TM) i5-5287U CPU @ 2.90GHz 2.90 GHz

Installed memory (RAM): 4.00 GB (2.86 GB usable)

System type: 64-bit Operating System, x64-based processor

Pen and touch: No Pen or Touch Input is available for this Display