ADDRESSING MODES/8086

-Mrs. JYOTSNA A. KULKARNI

ASSEMBLY LANGUAGE PROGRAMME INSTRUCTION FORMAT

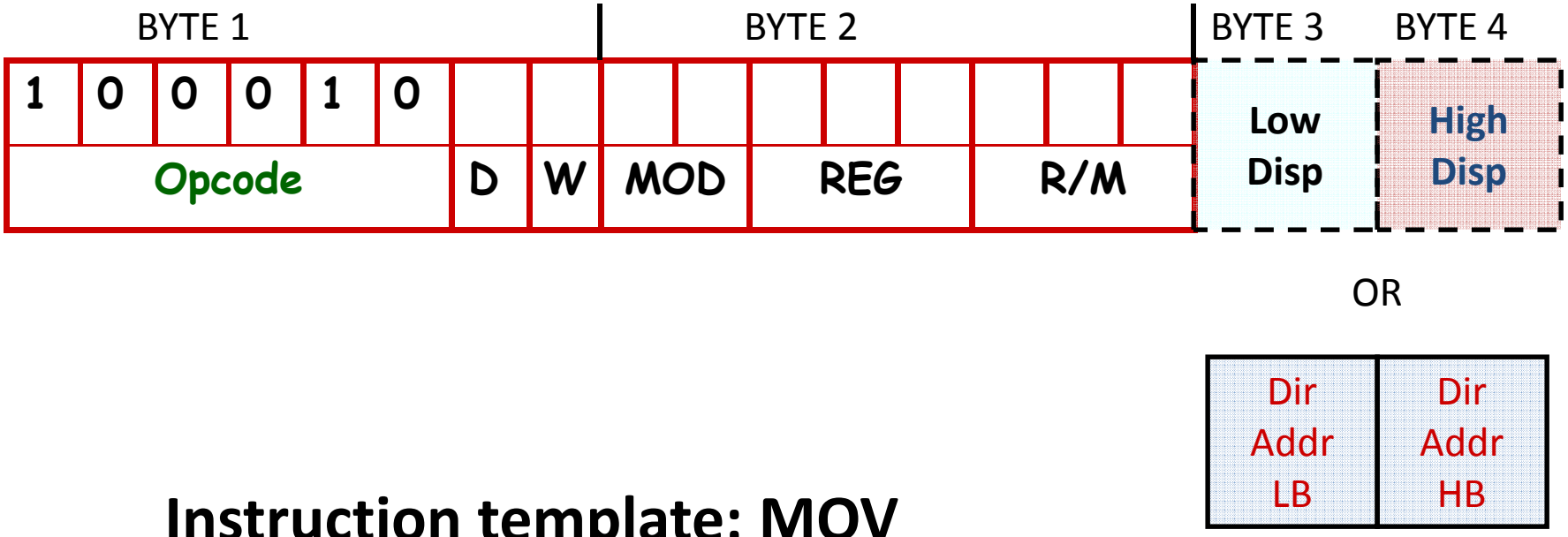
LABEL	OP CODE	OPERAND	COMMENT
NEXT:	ADD	AL, 07H	; ADD a number

- All labels must begin with
  - A letter
  - or one of the following special characters: @, $, -, or ?.
  - a label may any length from 1 to 35 characters
- comments always begin with a semicolon (;)

- **Data Transfer Instructions**
- **Arithmetic Instructions**
- **Logical Instructions**
- **Branch and Program control Instructions**

MOV Instruction

MOV destination , source



Instruction template: MOV

PROGRAM MEMORY-ADDRESSING MODES

- Used with the JMP (jump) and CALL instructions.
- Consist of three distinct forms:
  - direct, relative, and indirect

JMP Instruction

- **JMP - Go to specified address to get next instruction**
  - **Conditional (ex:JA, JNA)**
  - **Unconditional (JMP)**
    - **JMP BX , JMP WORD PTR [BX] ,**
    - **JMP NEXT**
      - **Location of label NEXT**
        - » **NEXT is in same segment (short)**
          - **IP related**
            - **Direct [Opcode dispL dispH]**
            - **Opcode =E9 if IP <- IP+d16**
            - **Opcode =E8 if IP <- IP+d8**
            - **Indirect [Opcode mod 100 r/m memL memH]**
              - **Opcode =FF if IP <- Reg 16**
              - **Opcode =FF if IP <- Mem 16**

NEXT is in some another segment(inter-seg / far)

IP <- offset

CS <-segment base

Opcode =EA for Direct **&** **FF for indirect addressing**

Unconditional inter-segment direct JMP

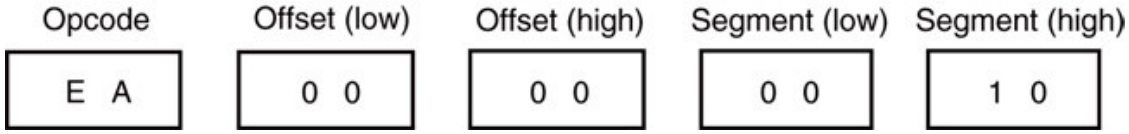
I**ntersegment jump** - jump to any memory location within the entire memory system

Ex: The 5-byte machine language version of a JMP [1000H] instruction.

-loads CS with 1000H

- IP with 0000H

jump to memory location 1000H for the next instruction.



Unconditional inter-segment direct JMP

opcode mod 101 r/m

CALL instruction

- CALL LABEL - Call a procedure (subprogram),save return address on stack
  - **Direct [opcode Displow DispHigh]**
  - **Opcode =E8 if IP <- IP+Disp(16)-SP <- return**
  - **Opcode =FF if IP <- Reg(16) –SP<-return**
  - **Indirect [opcode mod 010 r/m - -]**
    - **Opcode =FF if IP <- Reg 16 –SP<-return**
    - **Opcode =FF if IP <- Mem 16 –SP<-return**
    - **Ex: CALL BP**
    - **CALL WORD PTR [BX]**

Inter-seg CALL

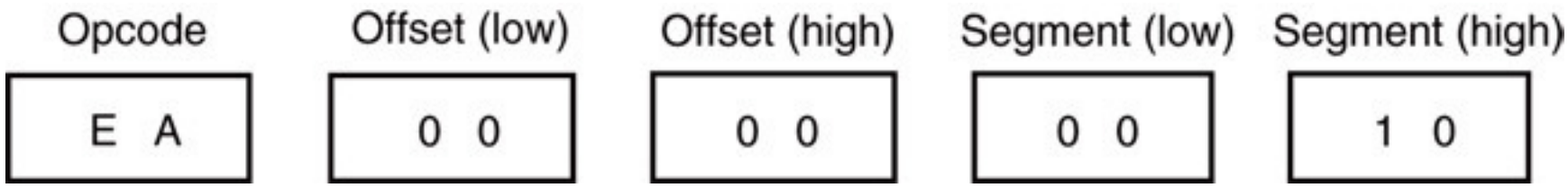
- **LABEL is in some another segment(inter-seg / far)**
  - **IP <- offset (first word from memory)**
  - **CS ,-segment base (second word from memory)**
- **Opcode =9A for Direct [opcode offsetL offsetH SegL SegH]**
- **Opcode= FF for indirect addressing [opcode mod 011 r/m memL memH]**
  - **IP <- offset (first word from memory)**
  - **CS <-segment base (second word from memory)**
  - **Ex: CALL DWORD PTR [BX]**
  - **IP<- [BX] & [BX+1] in DS**
  - **CS <- [BX+2] [BX+3]**

- The only other instruction using direct program addressing is the intersegment or far CALL instruction.
- Usually, the name of a memory address, called a *label*, refers to the location that is called or jumped to instead of the actual numeric address.
- When using a label with the CALL or JMP instruction, most assemblers select the best form of program addressing.

Direct Program Memory Addressing

- Used for all jumps and calls by early microprocessor; also used in high-level languages, such as BASIC.
  - GOTO and GOSUB instructions
- Not used oftenly as relative and indirect program memory addressing.
- The instructions for direct program memory addressing store the address with the opcode.

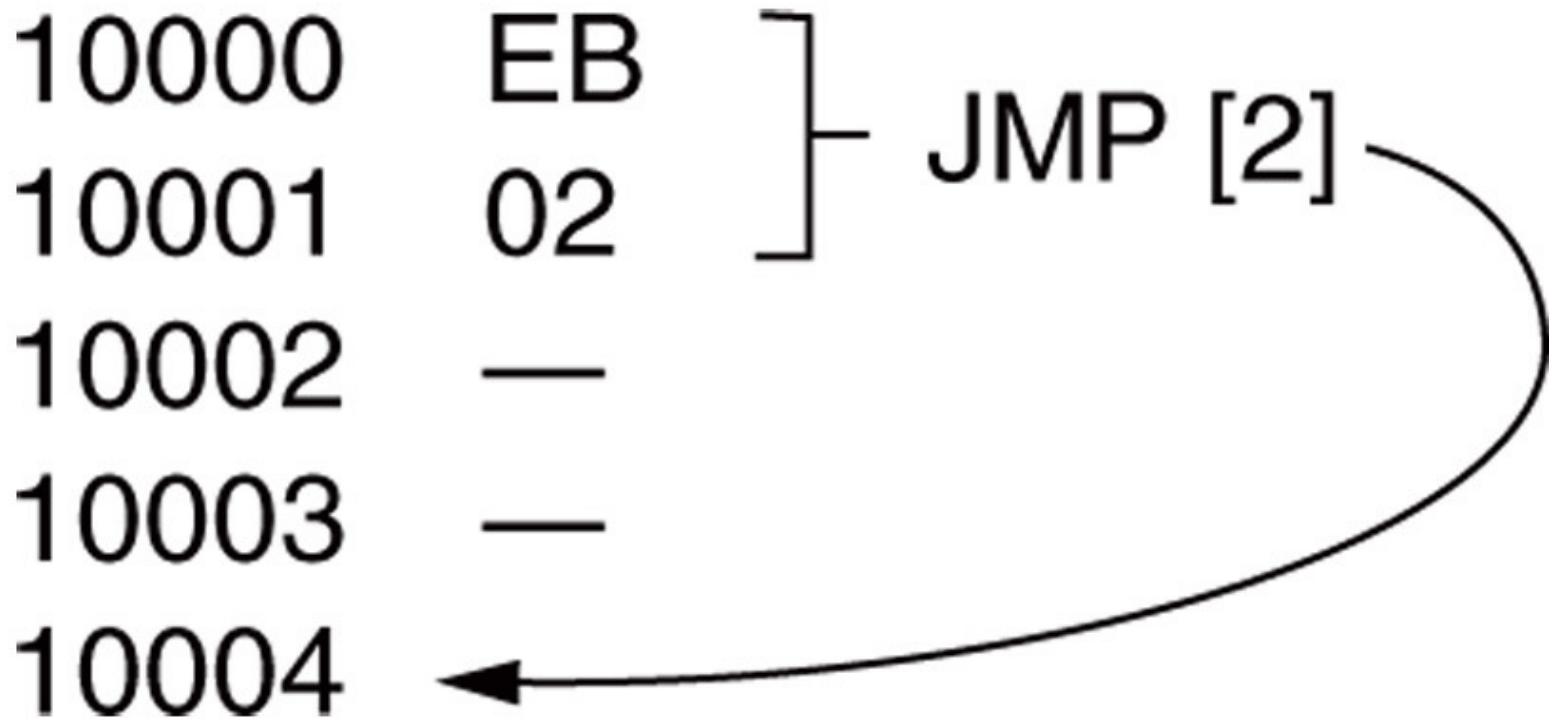
Figure 3–14 The 5-byte machine language version of a JMP [10000H] instruction.



Relative Program Memory Addressing

- The term *relative* means “relative to the instruction pointer (IP)”.
- The JMP instruction is a 1-byte instruction, with a 1-byte or a 2-byte displacement that adds to the instruction pointer.
- **Example:** A JMP [2] instruction. This instruction skips over the 2 bytes of memory that follow the JMP instruction.

Example: A JMP [2] instruction. This instruction skips over the 2 bytes of memory that follow the JMP instruction.



Indirect Program Memory Addressing

- The microprocessor allows several forms of program indirect memory addressing for the JMP and CALL instructions.
- If a relative register holds the address, the jump is considered to be an indirect jump.
- For example, JMP [BX] refers to the memory location within the data segment at the offset address contained in BX.
  - at 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*

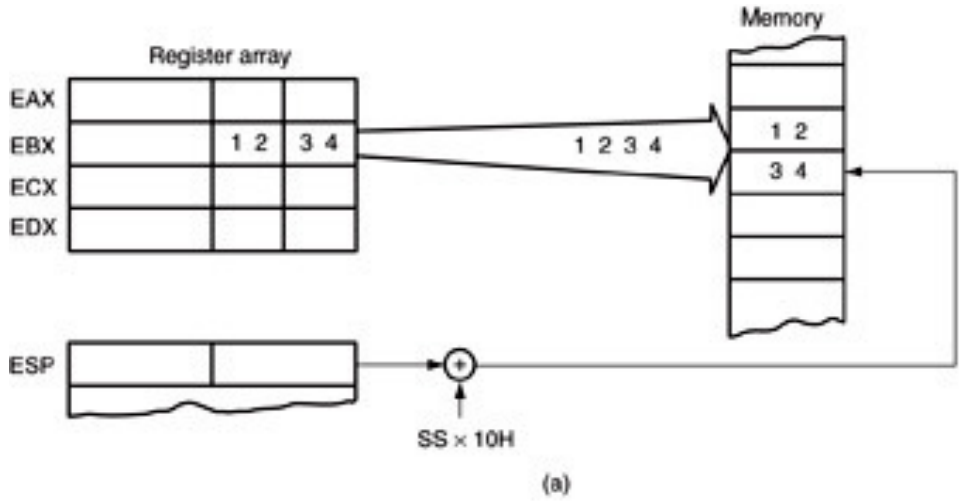
Example: A jump table that stores addresses of various programs. The exact address chosen from the TABLE is determined by an index stored with the jump instruction.

```
TABLE   DW   LOC0
        DW   LOC1
        DW   LOC2
        DW   LOC3
```

STACK MEMORY-ADDRESSING MODES

- The stack holds data temporarily
- Stack 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
- Stack memory is maintained by two registers:
  - the stack pointer (SP)
  - the stack segment register (SS)
- the SP register always points to an area of memory located within the stack segment.
- Data are placed on the stack with a **PUSH instruction**;
- Data are removed on the stack with a **POP instruction.**

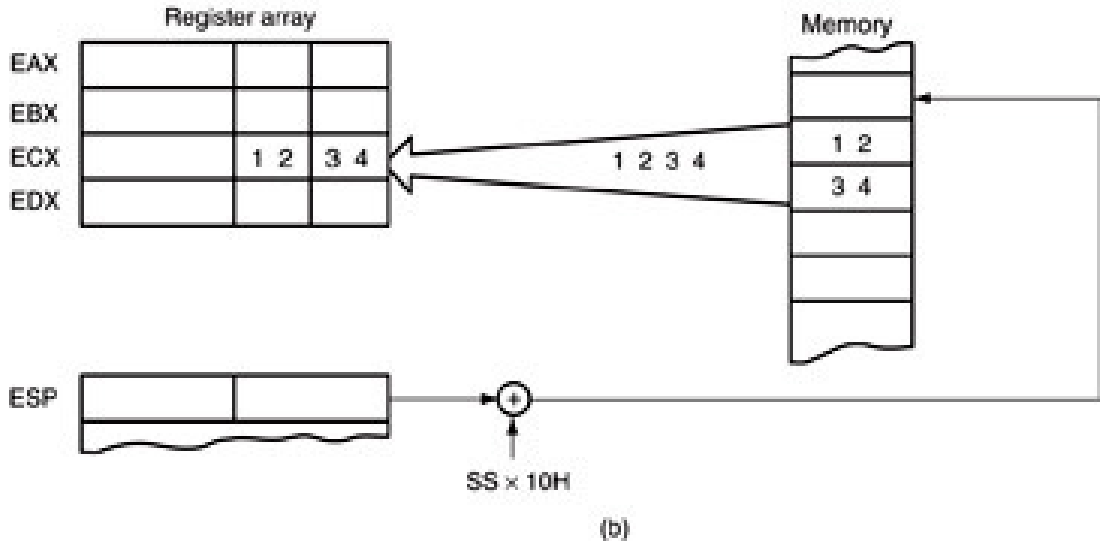
- PUSH
- Whenever a word of data is pushed onto the stack,
  - SP – 1 <- high-order 8 bits in the location addressed
  - SP-2 <- low-order 8 bits in the location addressed
- SP <- SP-2
- Ex: PUSH BX
  - places the contents of BX onto the stack;



POP

When data are popped from the stack,
Low-order 8 bits are removed from the location addressed by SP.
high-order 8 bits are removed from the SP
Next SP <-SP+2

Ex: POP CX removes data from the stack and places them into CX. Both instructions are shown after execution.



- Note that PUSH and POP store or retrieve words of data—never.
- Data may be pushed onto the stack from any 16-bit register or segment register.
- Data may be popped off the stack into any register or any segment register except CS.
- PUSHA and POPA instructions push or pop all except segment registers, on the stack.

{Not available on early 8086 processors.}