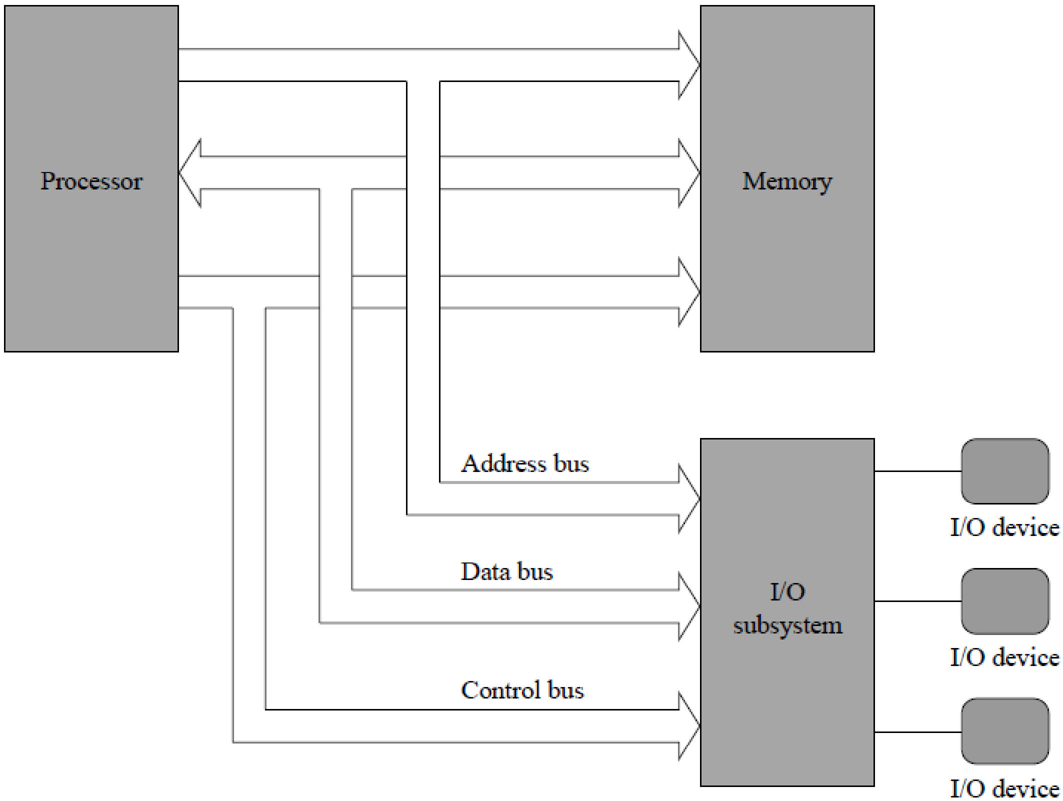
System Bus



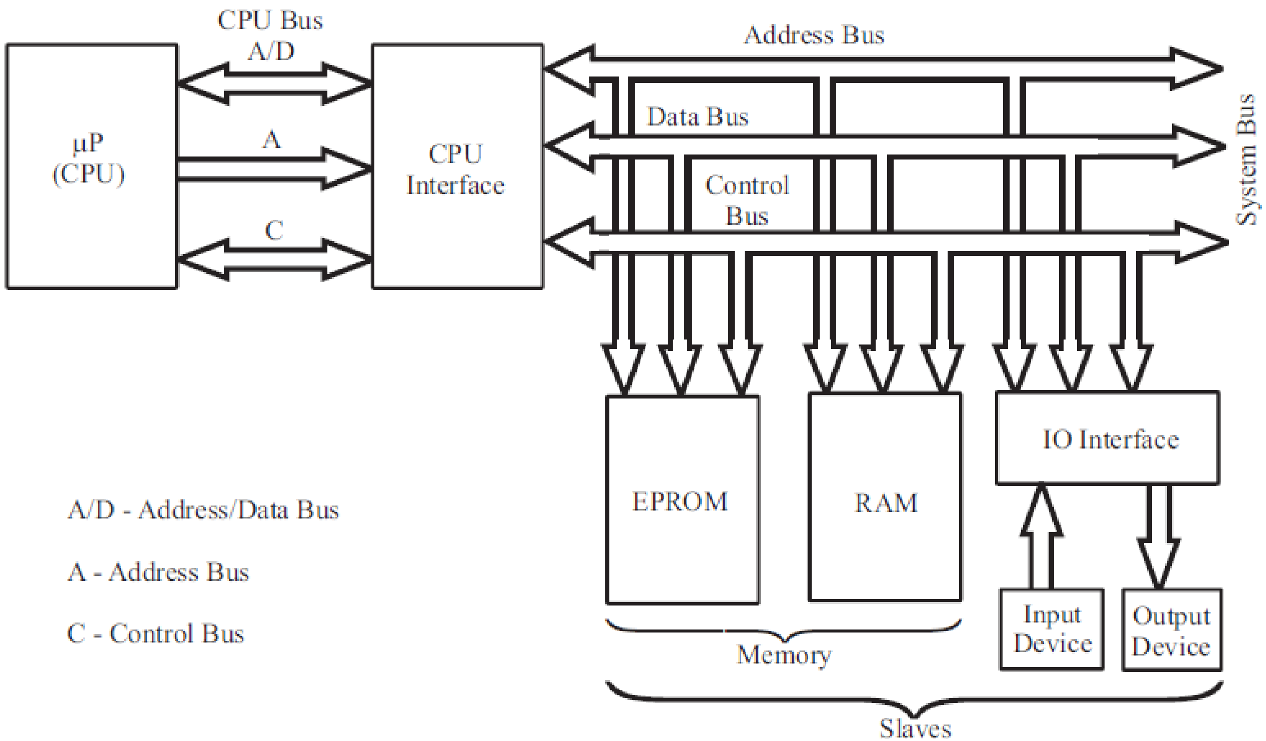
Consists of:

o Data Bus (16-bits) – move data between CPU, memory and I/O

o Address Bus– select memory (20-bits) or I/O location (16-bits)

o Control Bus – control how data is transferred (bus cycle type, data size, direction, etc.)

System Bus



Bus Cycle Types:

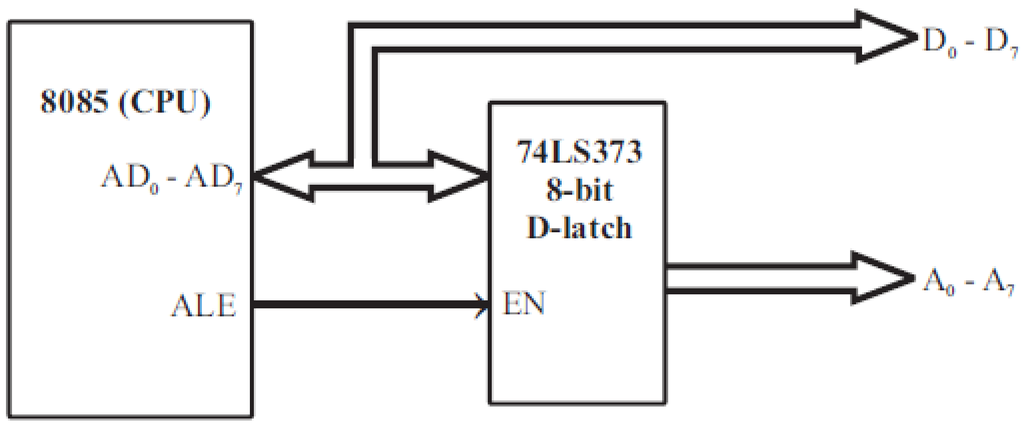
o Memory Read (MEMR) – CPU "fetch" instructions
or
read data from memory

o Memory Write (MEMW) – CPU writes data to
o memory

o I/O Read (IOR) – CPU reads from "port"

o I/O Write (IWR) – CPU writes to "port"

Demultiplexing of address and data lines in an 8085 processor

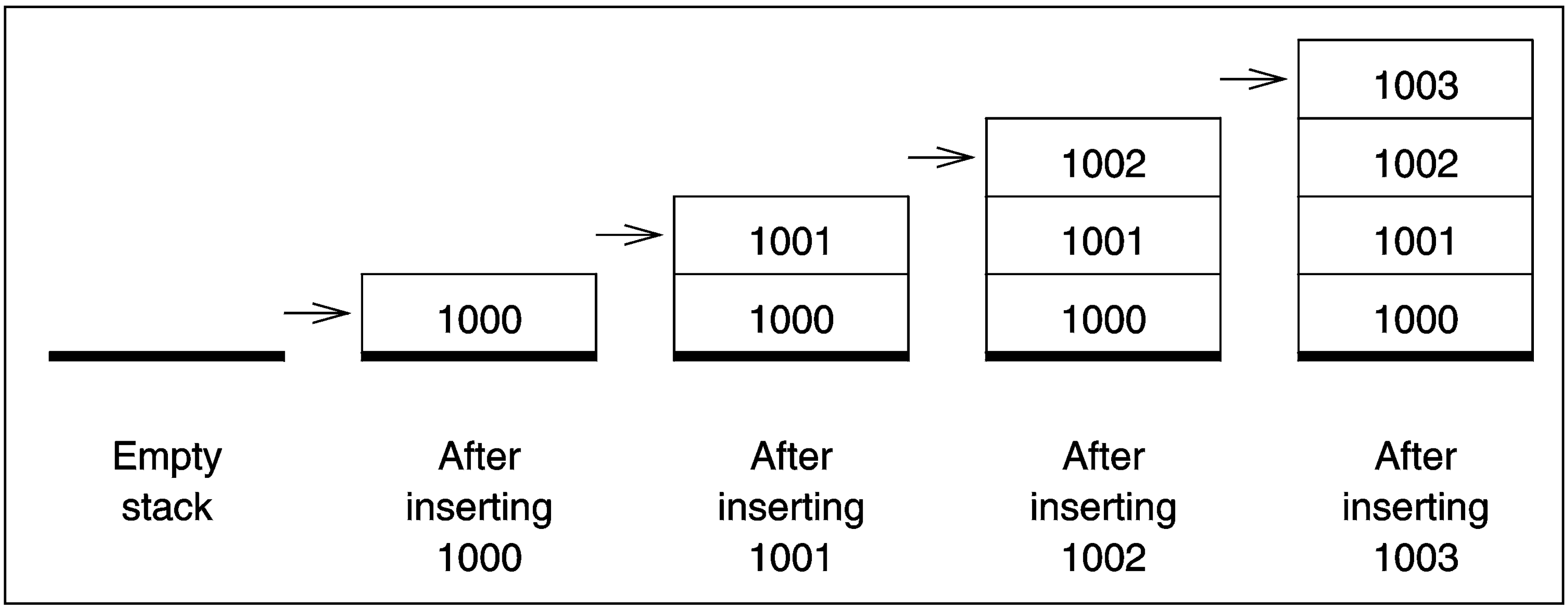


What is a Stack?

- Stack is a last-in-first-out (LIFO) data structure
- If we view the stack as a linear array of elements, both insertion and deletion operations are restricted to one end of the array
- Only the element at the top-of-stack (TOS) is directly accessible
- Two basic stack operations:
  - push (insertion)
  - pop (deletion)

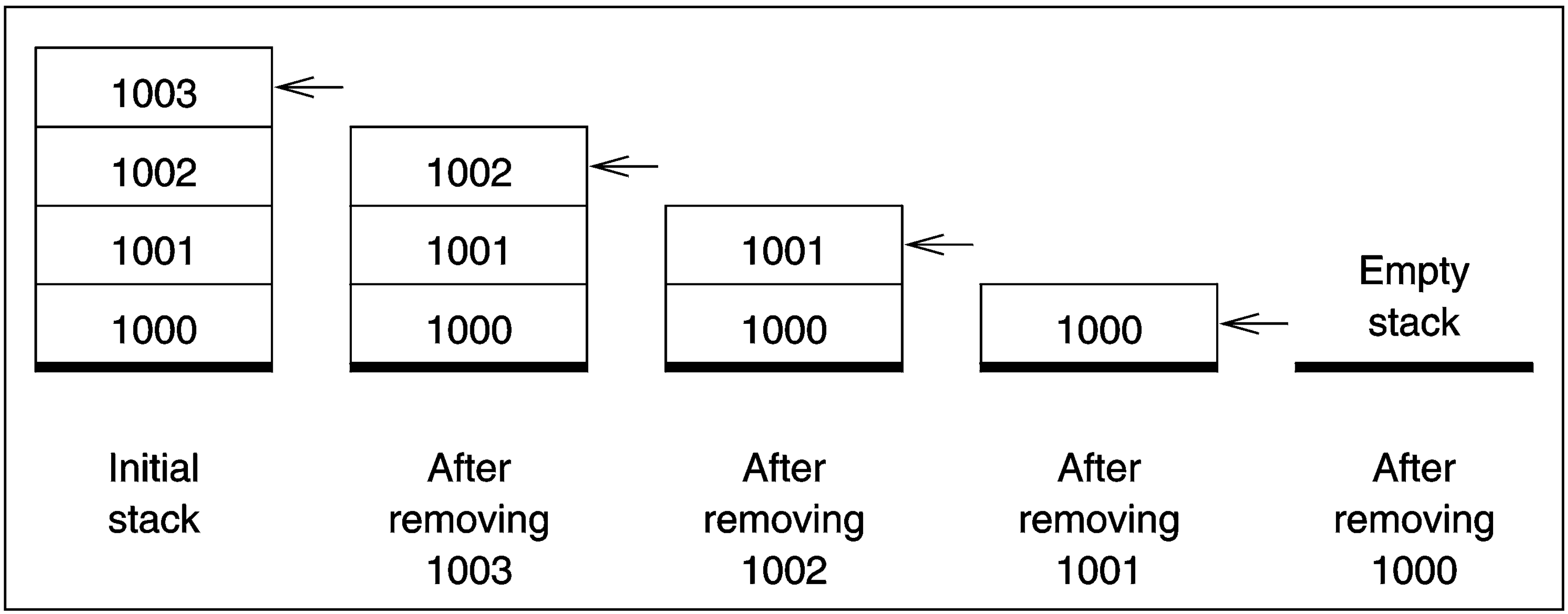
What is a Stack? (cont'd)

- Example
  - Insertion of data items into the stack
    - Arrow points to the top-of-stack



What is a Stack? (cont'd)

- Example
  - Deletion of data items from the stack
    - Arrow points to the top-of-stack



What is a Stack? - 8085

PUSH PSW **PUSH B** **PUSH D** **PUSH H**

Example : **PUSH B**

$$(SP) \leftarrow (SP) - 01$$
$$((SP)) \leftarrow (B)$$
$$(SP) \leftarrow (SP) - 01$$
$$((SP)) \leftarrow (C)$$

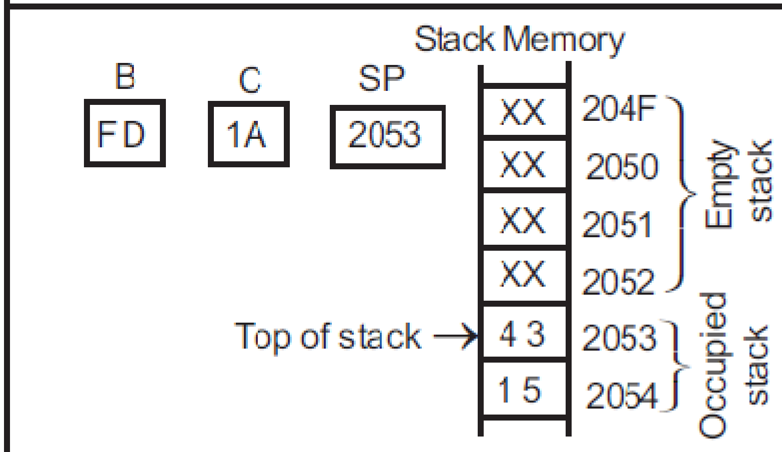
(i) *The content of the SP is decremented by one.*

(ii) *The content of the B-register is moved to the memory addressed by the **S**tack **P**ointer (SP).*

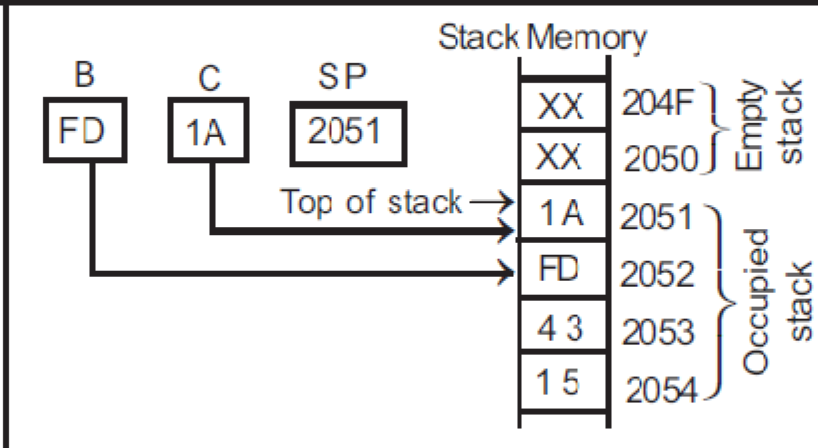
(iii) *Again the content of SP is decremented by one.*

(iv) *The content of the C-register is moved to the memory addressed by SP.*

Before execution



After execution



POP PSW **POP B** **POP D** **POP H**

Example : **POP D**

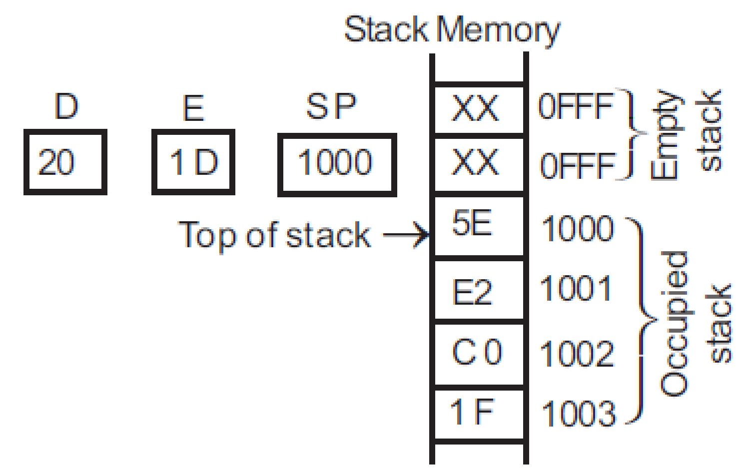
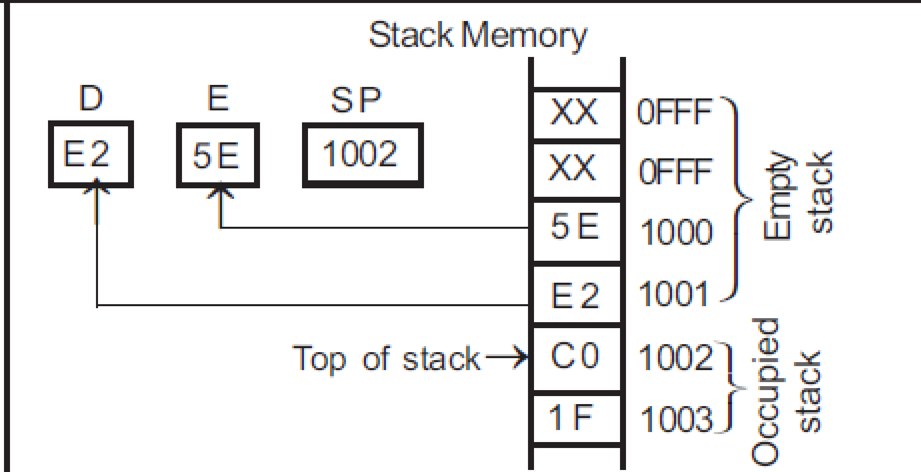
$$(E) \leftarrow ((SP))$$
$$(SP) \leftarrow (SP) + 01$$
$$(D) \leftarrow ((SP))$$
$$(SP) \leftarrow (SP) + 01$$

(i) The content of the memory addressed by the SP is moved to the E-register.

(ii) The content of the SP is incremented by one.

(iii) The content of the memory addressed by the SP is moved to the D-register.

(iv) The content of the SP is incremented by one.

Before execution	**After execution**
	

Example : **CMP M**

Let the content of the HL pair be $C050_H$. Let the content of the memory location $C050_H$ be $7A_H$. The content of the memory location $C050_H$ is compared with the content of the accumulator. Only flags are altered. The content of the accumulator and the memory remains the same.

Before execution	**Comparison**	**After execution**
A: 25 HL: C050 Memory: 7A C050 10 C051 CF = 0 PF = 0 AF = 0 ZF = 0 SF = 0	25_H = 0010 0101 $7A_H$ = 0111 1010 1'complement of $7A_H$ = 1000 0101 2'complement of $7A_H$ = 1000 0101 +1 = 1000 0110 = 86_H 25_H = 0010 0101 $+86_H$ = 1000 0110 $\boxed{0}$ 1010 1011 Complement Carry: $\boxed{1}$ A B	A: 25 HL: C050 Memory: 7A C050 10 C051 CF = 1 PF = 0 AF = 0 ZF = 0 SF = 1

Example : ADD M (A) ← (A) + (M) or (A) ← (A) + ((HL))

Let the content of A be 44_H.
Let the content of memory location $C00A_H$ be 73_H.
The content of the memory location $C00A_H$ is added to the content of the A-register. The result is put back in the A-register.

Before execution	Addition	After execution
A = 44 HL = C00A Memory: 73 C00A, 14 C00B, 27 C00C CF = 0 PF = 0 AF = 0 ZF = 0 SF = 0	44_H = 0100 0100 73_H = 0111 0011 1011 0111 Sum = B7 Carry = 0 (Addition is performed in ALU)	A = B7 HL = C00A Memory: 73 C00A, 14 C00B, 27 C00C CF = 0 PF = 1 AF = 0 ZF = 0 SF = 1

One byte instruction
Register indirect addressing

Two machine cycles: *Opcode fetch - 4T*
Memory read - 3T
7T

CALL addr16 $(SP) \leftarrow (SP) - 1 \quad ; \quad ((SP)) \leftarrow (PC)_H$

$$(SP) \leftarrow (SP) - 1 \quad ; \quad ((SP)) \leftarrow (PC)_L$$

$$(PC) \leftarrow addr16$$

It is unconditional CALL used to call a subroutine program. When this instruction is executed, the address of the next instruction in the program counter is pushed to the stack. The 16-bit address (which is the address of the subroutine program) given in the instruction is loaded in the program counter. Now, the processor will start executing the instructions stored in this call address.

Three byte instruction	**Five machine cycles:**	*Opcode fetch* -	*6T*
Immediate addressing		*Memory read* -	*3T*
		Memory read -	*3T*
		Memory write -	*3T*
		Memory write -	*3T*
			18T

RET	$(PC)_L \leftarrow ((SP))$	;	$(SP) \leftarrow (SP) + 1$
	$(PC)_H \leftarrow ((SP))$	;	$(SP) \leftarrow (SP) + 1$

(RET - Return to the main program)

It is an unconditional return instruction. This instruction is placed at the end of the subroutine program, in order to return to the main program. When this instruction is executed, the top of the stack is poped to (loaded in) the program counter .

> ***Note :*** *While calling the subroutine using CALL instruction, the return address of the main program is pushed to the stack. The return instruction, (RET) pops that to the program counter. Thus the processor resumes the execution of main program.*

One byte instruction	***Three machine cycles:***	*Opcode fetch*	-	*4 T*
Register indirect addressing		*Memory read*	-	*3 T*
		Memory read	-	*3 T*
				10 T

JMP addr16 **(PC) ← addr16**

It is unconditional jump instruction. When this instruction is executed, the address given in the instruction is moved to the program counter. Now, the processor starts executing the instructions stored in this address.

Three byte instruction	***Three machine cycles:***	*Opcode fetch -*	*4 T*
Immediate addressing		*Memory read -*	*3 T*
		Memory read -	*3 T*
			10T

J <condition> addr16

If <condition> is TRUE then,

(PC) ← addr16

It is conditional jump instruction. The conditional jump instruction will check a flag condition. If the flag condition is true, then the address given in the instruction is moved to the program counter. Thus the program control is branched to the jump address. If the flag condition is false, then the next instruction is executed.

Uses of the Stack (8086)

- Often used to free a set of registers

```
;save AX & BX registers on the stack
push    AX
push    BX
;AX and BX registers can now be used
mov     AX,value1
mov     BX,value2
mov     value1,BX
mov     value2,AX
;restore AX & BX registers from the stack
pop     BX
pop     AX
. . .
```