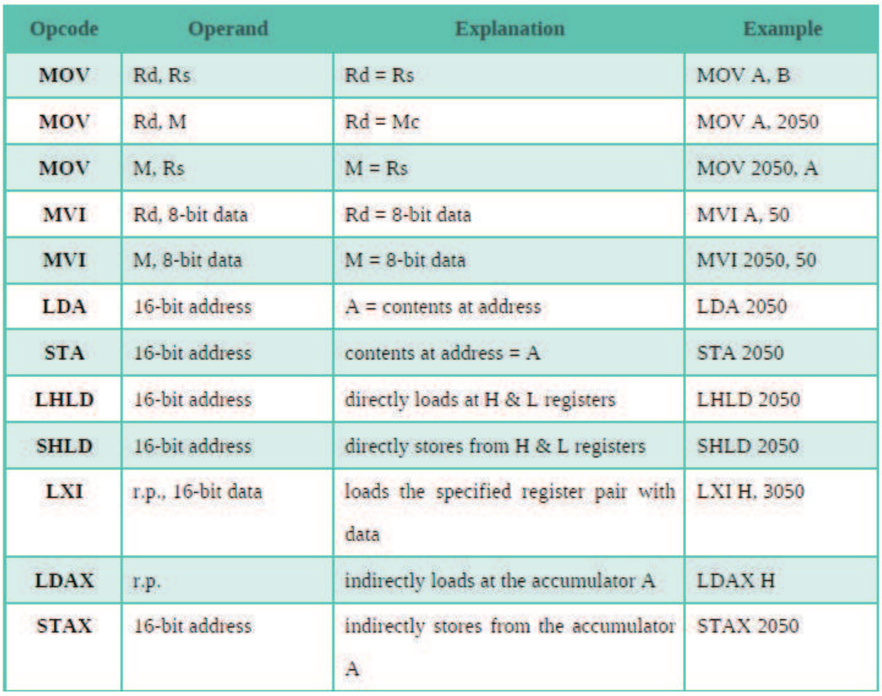
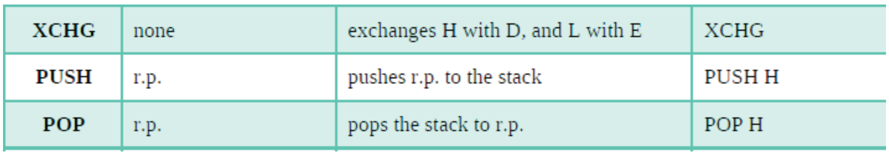
Data Transfer Instructions

Data transfer instructions are the instructions which transfers data in the microprocessor. They are also called copy instructions.

Following is the table showing the list of data transfer instructions:





Opcode	Operand	Explanation	Example
MOV	Rd, Rs	Rd = Rs	MOV A, B
MOV	Rd, M	Rd = Mc	MOV A, 2050
MOV	M, Rs	M = Rs	MOV 2050, A
MVI	Rd, 8-bit data	Rd = 8-bit data	MVI A, 50
MVI	M, 8-bit data	M = 8-bit data	MVI 2050, 50
LDA	16-bit address	A = contents at address	LDA 2050
STA	16-bit address	contents at address = A	STA 2050
LHLD	16-bit address	directly loads at H & L registers	LHLD 2050
SHLD	16-bit address	directly stores from H & L registers	SHLD 2050
LXI	r.p., 16-bit data	loads the specified register pair with data	LXI H, 3050
LDAX	r.p.	indirectly loads at the accumulator A	LDAX H
STAX	16-bit address	indirectly stores from the accumulator A	STAX 2050
XCHG	none	exchanges H with D, and L with E	XCHG
PUSH	r.p.	pushes r.p. to the stack	PUSH H
POP	r.p.	pops the stack to r.p.	POP H

In the table,
R stands for register
M stands for memory
r.p. stands for register pair

Branching Instructions

Branching instructions refer to the act of switching execution to a different instruction sequence as a result of executing a branch instruction.

The three types of branching instructions are:

- Jump (unconditional and conditional)
- Call (unconditional and conditional)
- Return (unconditional and conditional)

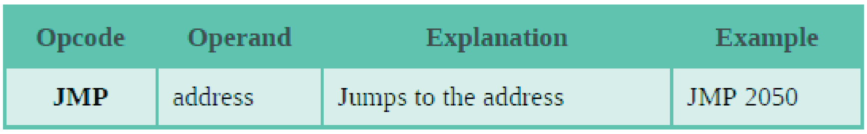
Jump Instructions

The jump instruction transfers the program sequence to the memory address given in the operand based on the specified flag.

Jump instructions are of two types:

Unconditional Jump Instructions
Conditional Jump Instructions.

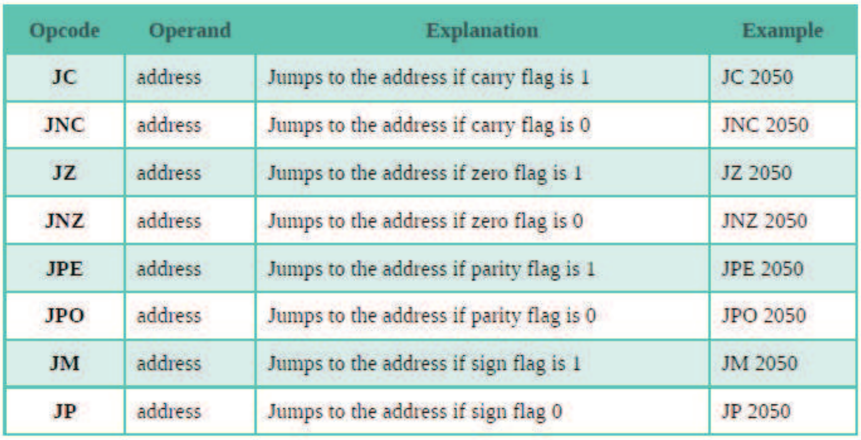
Unconditional Jump Instructions:
Transfers the program sequence to the described memory address.



Opcode	Operand	Explanation	Example
JMP	address	Jumps to the address	JMP 2050

Conditional Jump Instructions

Transfers the program sequence to the described memory address only if the condition in satisfied.



Opcode	Operand	Explanation	Example
JC	address	Jumps to the address if carry flag is 1	JC 2050
JNC	address	Jumps to the address if carry flag is 0	JNC 2050
JZ	address	Jumps to the address if zero flag is 1	JZ 2050
JNZ	address	Jumps to the address if zero flag is 0	JNZ 2050
JPE	address	Jumps to the address if parity flag is 1	JPE 2050
JPO	address	Jumps to the address if parity flag is 0	JPO 2050
JM	address	Jumps to the address if sign flag is 1	JM 2050
JP	address	Jumps to the address if sign flag 0	JP 2050

Call Instructions

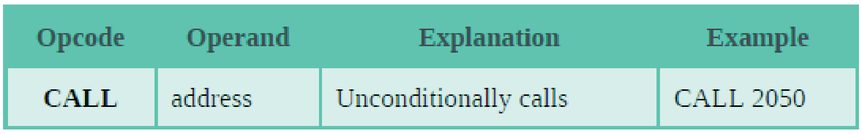
The call instruction transfers the program sequence to the memory address given in the operand. Before transferring, the address of the next instruction after CALL is pushed onto the stack.

Call instructions are 2 types:

Unconditional Call Instructions and Conditional Call Instructions.

Unconditional Call Instructions

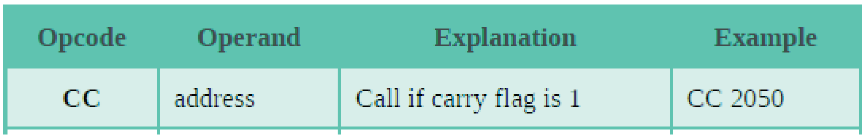
It transfers the program sequence to the memory address given in the operand.

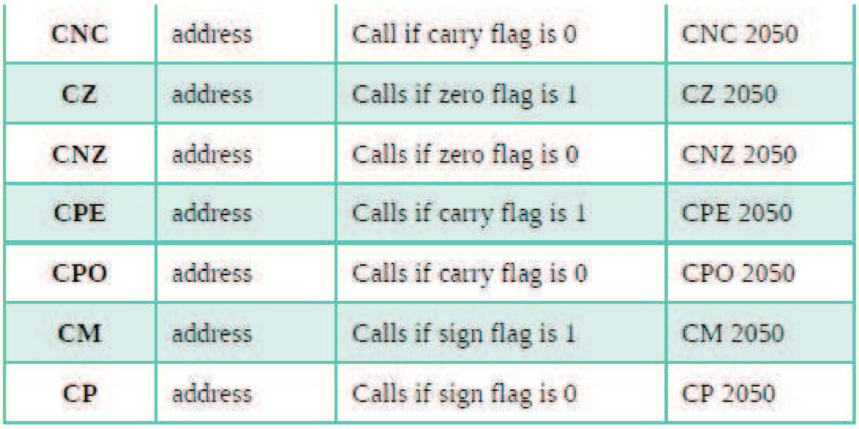


Opcode	Operand	Explanation	Example
CALL	address	Unconditionally calls	CALL 2050

Conditional Call Instructions

Only if the condition is satisfied, the instructions executes.





Opcode	Operand	Explanation	Example
CC	address	Call if carry flag is 1	CC 2050
CNC	address	Call if carry flag is 0	CNC 2050
CZ	address	Calls if zero flag is 1	CZ 2050
CNZ	address	Calls if zero flag is 0	CNZ 2050
CPE	address	Calls if carry flag is 1	CPE 2050
CPO	address	Calls if carry flag is 0	CPO 2050
CM	address	Calls if sign flag is 1	CM 2050
CP	address	Calls if sign flag is 0	CP 2050

Return Instructions

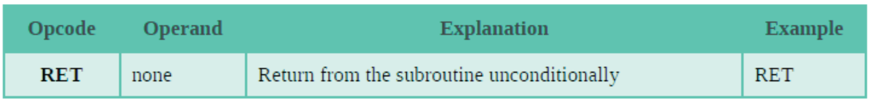
The return instruction transfers the program sequence from the subroutine to the calling program.

Return instructions are 2 types:

Unconditional Return Instructions and Conditional Return Instructions.

Unconditional Return Instruction

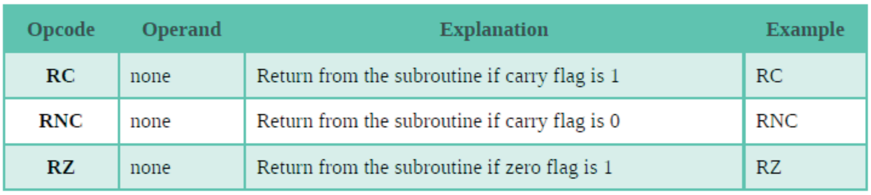
The program sequence is transferred unconditionally from the subroutine to the calling program.



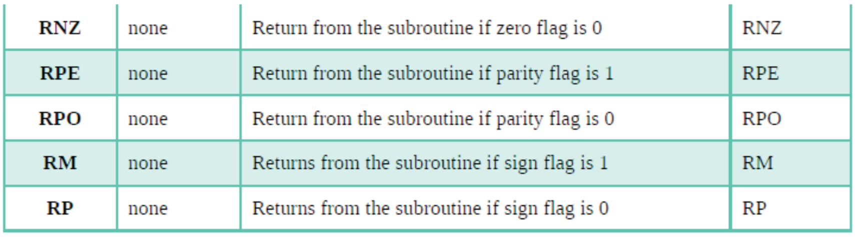
Opcode	Operand	Explanation	Example
RET	none	Return from the subroutine unconditionally	RET

Conditional Return Instruction

The program sequence is transferred conditionally from the subroutine to the calling program only if the condition is satisfied.



Opcode	Operand	Explanation	Example
RC	none	Return from the subroutine if carry flag is 1	RC
RNC	none	Return from the subroutine if carry flag is 0	RNC
RZ	none	Return from the subroutine if zero flag is 1	RZ

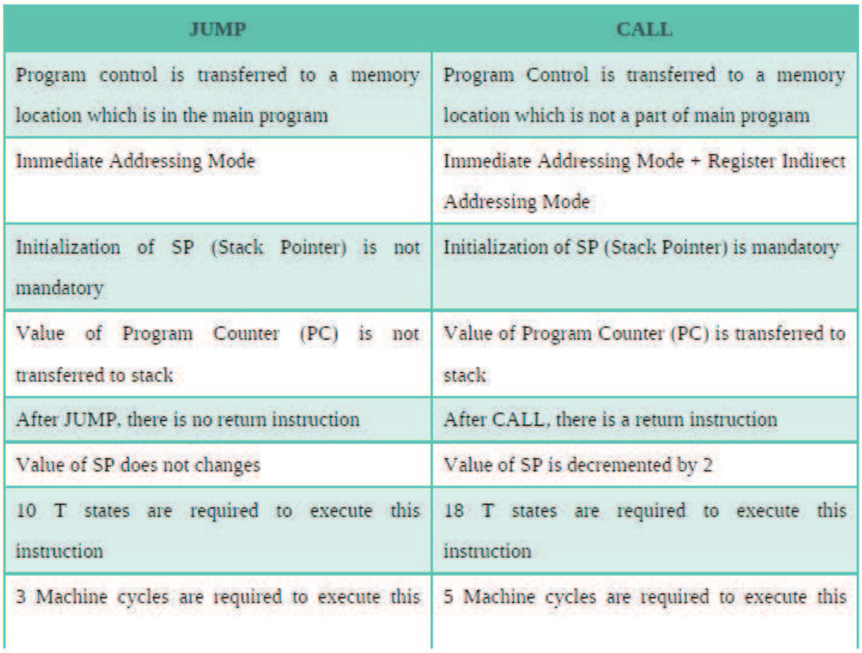


RNZ	none	Return from the subroutine if zero flag is 0	RNZ
RPE	none	Return from the subroutine if parity flag is 1	RPE
RPO	none	Return from the subroutine if parity flag is 0	RPO
RM	none	Returns from the subroutine if sign flag is 1	RM
RP	none	Returns from the subroutine if sign flag is 0	RP

Difference between CALL and JUMP instructions

CALL instruction is used to call a subroutine. Subroutines are often used to perform tasks that need to be performed frequently. The JMP instruction is used to skip some part of program.

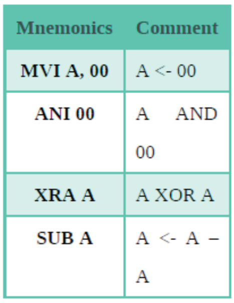
The differences Between CALL and JUMP instructions are:



JUMP	CALL
Program control is transferred to a memory location which is in the main program	Program Control is transferred to a memory location which is not a part of main program
Immediate Addressing Mode	Immediate Addressing Mode + Register Indirect Addressing Mode
Initialization of SP (Stack Pointer) is not mandatory	Initialization of SP (Stack Pointer) is mandatory
Value of Program Counter (PC) is not transferred to stack	Value of Program Counter (PC) is transferred to stack
After JUMP, there is no return instruction	After CALL, there is a return instruction
Value of SP does not changes	Value of SP is decremented by 2
10 T states are required to execute this instruction	18 T states are required to execute this instruction
3 Machine cycles are required to execute this	5 Machine cycles are required to execute this

Reset Accumulator

There are 4 instructions to reset the accumulator in 8085. These instructions are:



Mnemonics	Comment
MVI A, 00	A <- 00
ANI 00	A AND 00
XRA A	A XOR A
SUB A	A <- A – A

MVI A, 00: instruction copies 00 to A.

ANI 00: instruction performs bit by bit AND operation of source operand (i.e. 00) to the destination operand (i.e. the accumulator A) and store the result in accumulator A.

XRA A:instruction performs XOR operation between source operand and destination operand and store the result in the accumulator. Here, source and destination operand both are same i.e. A. Therefore, the result after performing XOR operation, stored in the accumulator is 00.

SUB A:operation subtracts the contents of source operand (here, source register is A) from the contents of accumulator and store the result in the accumulator itself. Since, the source and destination operand are same. Therefore, accumulator A = 00.