# Lecture Seven

## Instruction Execution and Addressing

## COMPUTER INSTRUCTION

A binary code used for specifying micro-operations for the computer.

In computer science, an **instruction set architecture** (**ISA**) is an abstract model of a computer. It is also referred to as **architecture** or **computer architecture**. A realization of an ISA, such as a central processing unit (CPU), is called an *implementation*.

In general, an ISA defines the supported data types, the registers, the hardware support for managing main memory, fundamental features (such as the memory consistency, addressing modes, virtual memory), and the input/output model of a family of implementations of the ISA.

## NUMBER OF OPERANDS

The number of operands of an operator is called its arity. Based on arity, operators are classified as nullary (no operands), unary (1 operand), binary (2 operands), ternary (3 operands), etc.

Operands specify the value an instruction is to operate on, and where the result is to be stored. Instruction sets are classified by the number of operands used. An instruction may have no, one, two, or three operands.

### 1. THREE-OPERAND INSTRUCTION:

In instruction that have three operands, one of the operand specifies the destination as an address where the result is to be saved. The other two operands specify the source either as addresses of memory location or constants.

**ADD destination, source1, source2**

EX: A=B+C

ADD A,B,C

EX: Y=(X+D)* (N+1)

```
        ADD T1, X, D
        ADD T2, N, 1
        Mul Y, T1, T2
```

## 2. TWO OPERAND INSTRUCTION

In this type both operands specify sources. The first operand also specifies the destination address after the result is to be saved. The first operand must be an address in memory, but the second may be an address or a constant.

**ADD destination, source**

EX: A=B+C

```
MOV AX, BX
ADD AX, CX
```

EX: Y=(X+D)* (N+1)

```
        MOV AX, X
        ADD AX, D
        MOV BX, N
        ADD BX, 1
        MUL BX
        MOV Y,AX
```

## 3. ONE OPERAND INSTRUCTION

Some computer have only one general purpose register, usually called on Acc. It is implied as one of the source operands and the destination operand in memory instruction the other source operand is specified in the instruction as location in memory.

**ADD source**

LDA source; copy value from memory to ACC.
STA destination; copy value from Acc into memory.

EX: A=B+C

```
LDA B
ADD C
STA A
```

EX: Y=(X+D)* (N+1)

```
LDA X
ADD D
STA T1
LDA N
ADD 1
MUL T1
STA Y
```

## 4. ZERO OPERAND INSTRUCTION

Some computers have arithmetic instruction in which all operands are implied, these zero operand instruction use a stack, a stack is a list structure in which all insertion and deletion occur at one end, the element on a stack may be removed only in the reverse of the order in which they were entered. The process of inserting an item is called **Pushing**, removing an item is called **Popping**.

Computers that use Zero operand instruction for arithmetic operations also use one operand **PUSH** and **POP** instruction to copy value between memory and the stack.

| | |
|---|---|
| PUSH source; | Push the value of the memory operand onto the Top Of the stack. |
| POP destination; | POP value from the Top of the stack and copy it into The memory operand. |

```
EX: A=B+C                      EX: Y=(X+D)* (N+1)
PUSH B                         PUSH X
PUSH C                         PUSH D
ADD                            ADD
POP A                          PUSH N
                               PUSH 1
                               ADD
                               MUL
                               POP Y
```

***NOTE IN ADD*** Pop the two value of the stack, add them, and then push the sum back into the stack.

## INSTRUCTION EXECUTION AND ADDRESSING

There are a lot of instructions in assembly but there are only about twenty that you have to know and will use very often. Most instructions are made up of three characters and have an operand then a comma then another operand. For example to put a data into a register you use the MOV instruction.

## 1. DATA TRANSFER INSTRUCTIONS

The microprocessor has a group of data transfer instructions that are provided to move data either between its internal registers or between an internal register and a storage location in memory. Some of these instructions are:

### MOV

MOV use to transfer a byte or a word of data from a source operand to a destination operand. It’s more useful data transfer instruction because it transfers the data from one memory location to another. These operands can be internal registers and storage locations in memory. Notice that the MOV instruction cannot transfer data directly between a source and a destination that both reside in external memory. For instance, flag bits within the microprocessors are not modified by execution of a MOV instruction.

**EXAMPLES:**

| | | | |
|---|---|---|---|
| 1. MOV DX, CS | Where CX=0100H | DX=CS=0100H | CS $\longrightarrow$ DX |
| 2. MOV AX, 05H | Transform the value 05 to AX | | |
| 3. MOV BX, [0ABCD] | Transform the value that saved in location 0ABCD to BX | | |

### XCHG

In MOV instruction the original contents of the source location are preserved and the original contents of the destination are destroyed. But **XCHG** (exchange) instruction can be used to swap data between two general purpose register or between a general purpose register and storage location in memory.

**EXAMPLES:**

| | |
|---|---|
| XCHG AL, DL | Exchanges the contents of AL with DL. |

### Load Effective Address LEA, LDS, LES, LFS, LGS, and LSS

There are several load-effective address instructions in the microprocessor instruction set. The LES instruction loads any 16 bit register with the offset address of the data specified by the operands, as determined by the addressing mode selected for the instruction.

**EXAMPLES:**

LED BX,[DI] MOV BX,[DI]
The LDS, LES, LCS and LSS instructions load any 16 bit or 32 bit register with an offset address and the DS, ES, CS or SS segment register with a segment address.

**EXAMPLES:**
LDS BX,[DI] ,This instruction transfers the 32bit number addressed by DI in the data segment into BX and DS register.

## Push and POP Instruction

It is necessary to save the contents of certain registers or some other main program parameters. These values are saved by pushing them onto the stack. Typically, these data correspond to registers and memory locations that are used by the subroutine. The instruction that is used to save parameters on the stack is the push (PUSH) instruction and that used to retrieve them back is the pop (POP) instruction. Notice a general-purpose register, a segment register (excluding CS), or a storage location in memory as their operand.

Execution of a PUSH instruction causes the data corresponding to the operand to be pushed onto the top of the stack. For instance, if the instruction is PUSH AX the result is as follows:
((SP)-1) (AH)
((SP)-2) (AL)
This shows that the two bytes of the AX are saved in the stack part of memory and the stack pointer is decrement by 2 such that it points to the new top of the stack.
On the other hand, if the instruction is POP AX Its execution results in
(AL) ((SP))
(AH) ((SP) + 1)
The saved contents of AX are restored back into the register.
We also can save the contents of the flag register and if saved we will later have to restore them. These operations can be accomplished with the push flags (PUSHF) and pop flags (POPF) instructions, respectively. Notice the PUSHF save the contents of the flag register on the top of the stack. On the other hand, POPF returns the flags from the top of the stack to the flag register.