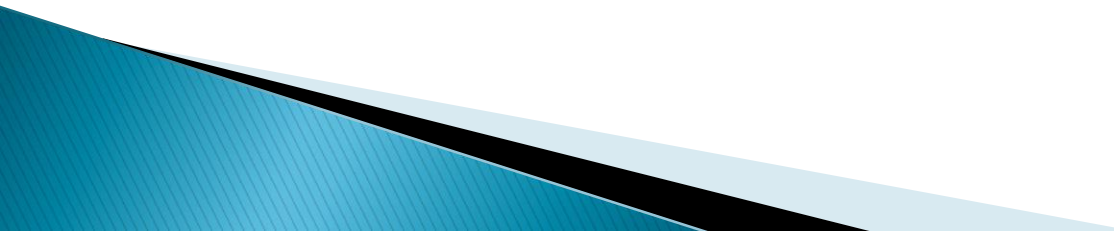


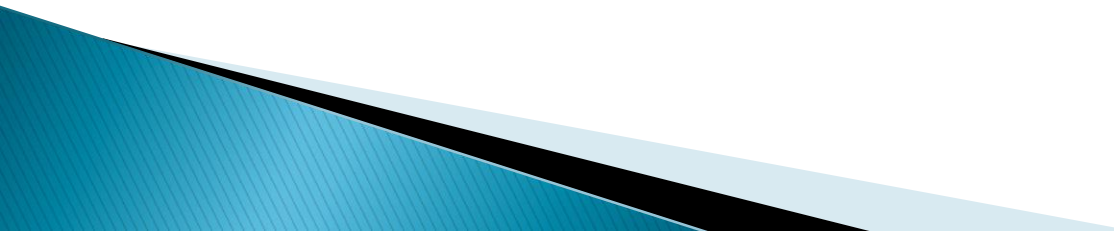
# Introduction to Assembly language Programming

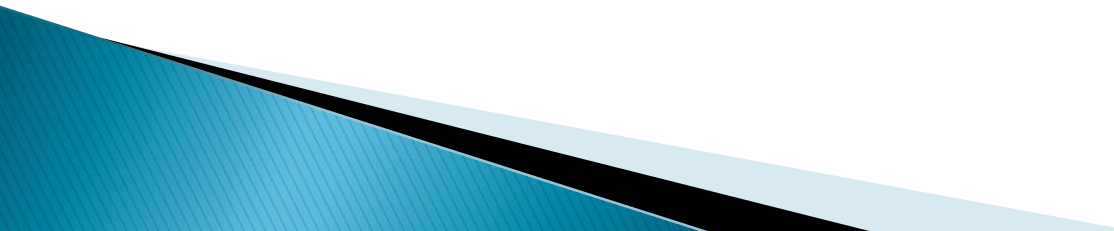
Unit 2

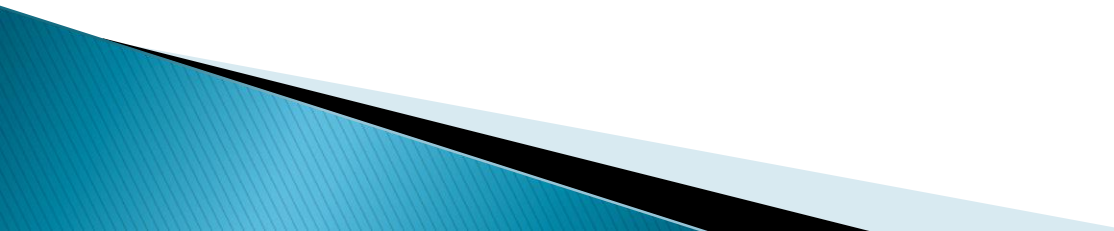
# Mnemonics

- ▶ Mnemonics is a Greek word called mindful(mind aid).
  - ▶ Manufacturer of microprocessor has devised a symbolic code for each instruction called mnemonics.
  - ▶ E.g 0011 1100 in 8085 is INR A.
- 

# Assembly language and High level languages

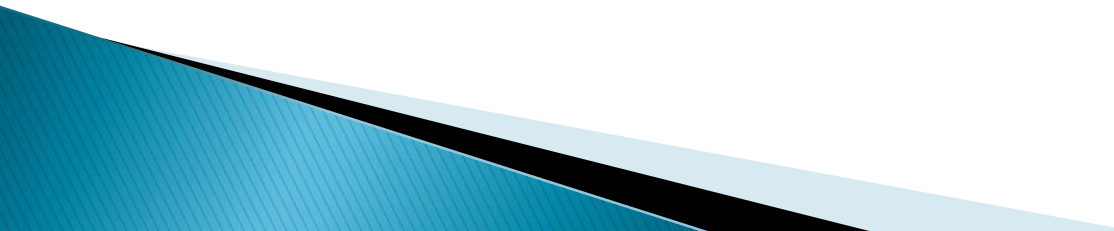
- ▶ Each device has its own **set of instruction** depending on design of CPU.
  - ▶ To communicate with computer one must give instructions in **binary form**.
  - ▶ It is difficult for most people to write programs in sets of 0's and 1's.
  - ▶ So, microprocessor manufacturer introduce **English like words** to represents the binary instruction of machine.
- 

- ▶ Programmer can write programs called **assembly language** programs using **mnemonics**.
  - ▶ Assembly language program are **non transferrable** from one machine to other.
  - ▶ Programming languages are of two types–
    - **low level language**
      - Machine language
      - Assembly language
    - **High level language** (machine independent like BASIC,FORTAN)
- 

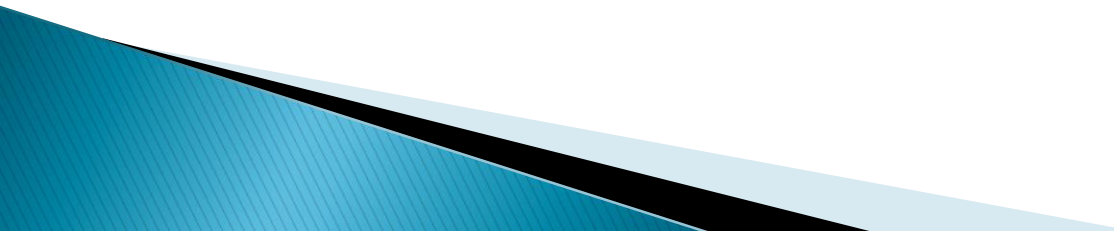
- ▶ **High level languages are machine independent and are transferrable.**
  - ▶ **Machine and assembly languages are microprocessor specific**
  - ▶ **Since a program can be written in high level or assembly language but microprocessor only understand binary so it must be translated to binary code or machine code**
  - ▶ **The program written in assembly languages are translated into a binary code by using a program called an assembler.**
- 

- ▶ Programming with Intel 8085 microprocessor

# Introduction to assembly language programming

- ▶ Assembly language use two three or four letter mnemonics to represent each instruction type.
  - ▶ Low level assembly language is designed for a specific family of processor
  - ▶ The symbolic instruction of assembly language are assembled into machine language
  - ▶ Assembly language makes writing program easier than machine code.
  - ▶ Later it is translated into machine language and can be loaded into memory and run.
- 

# Advantage of assembly language

- ▶ Requires less memory and execution time than high level language.
  - ▶ Gives programmer the ability to perform highly technical tasks.
  - ▶ Provides more control over handling particular hardware requirement.
  - ▶ Generates smaller and compact executable modules.
- 



# Example of machine-language

Here's what a program-fragment looks like:

```
10100001 10111100 10010011 00000100
00001000 00000011 00000101 11000000
10010011 00000100 00001000 10100011
11000000 10010100 00000100 00001000
```

It means:             $z = x + y;$

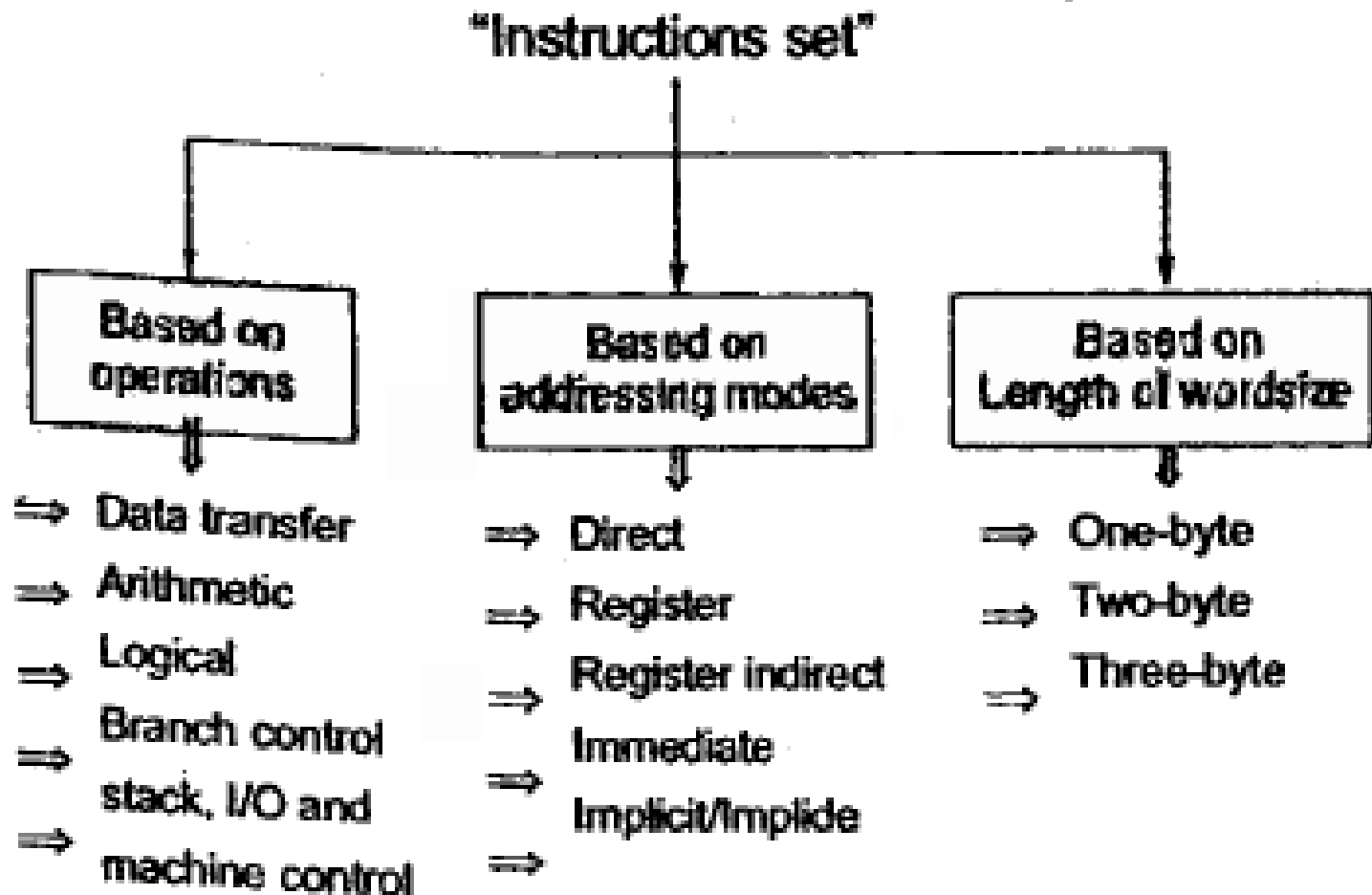
# Typical Format of Assembly Language Instruction

Label	Opcode Field	Operand Field	Comments
Next:	ADD	AL,07H	;Add correction factor

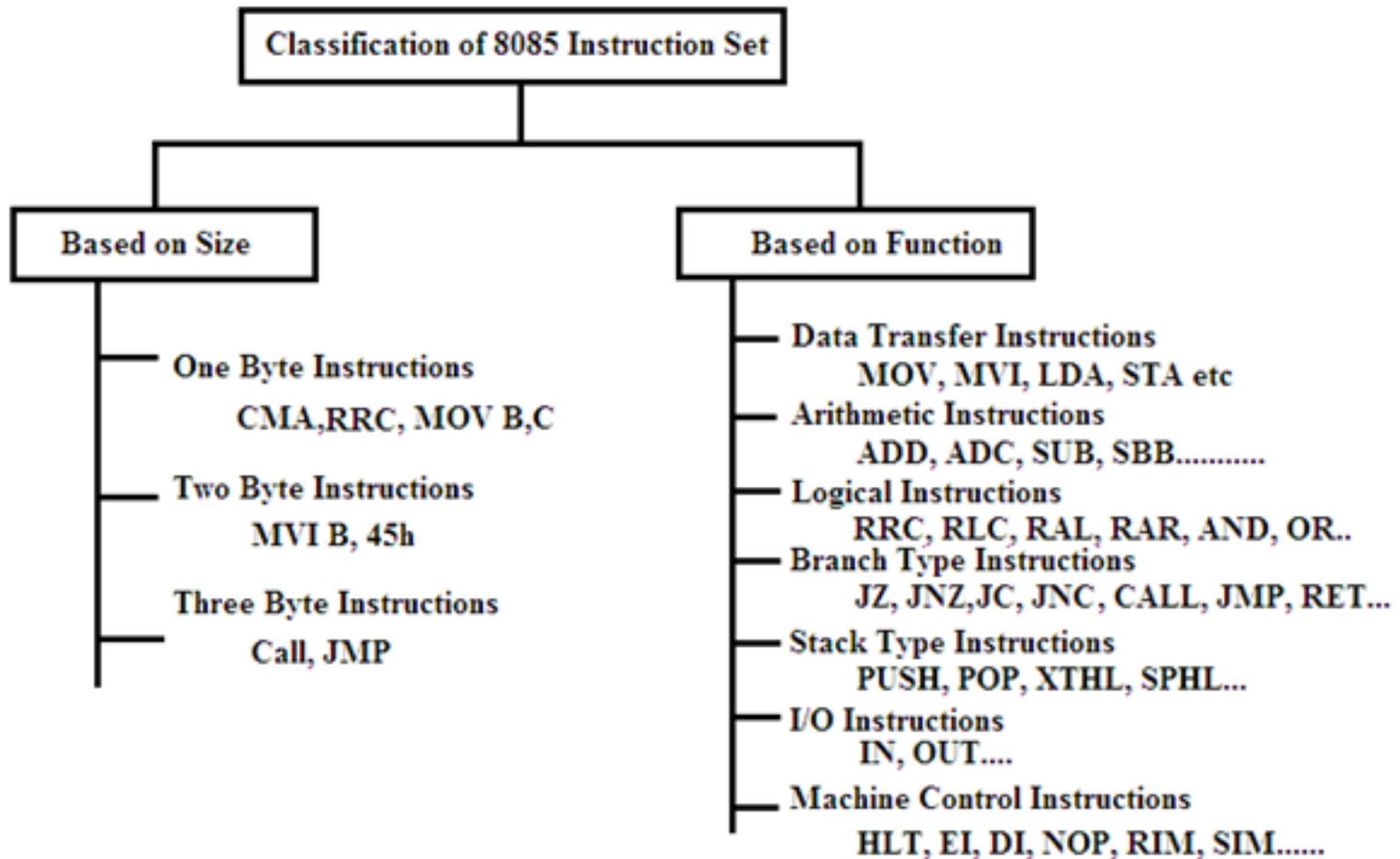
ADD R1, R0

- Here R0 is source Register & R1 is destination register
- Instruction adds contents of R0 with the contents of R1 and stores result in R1 register.
- 8085 Can handle maximum of 256 instructions.

# Classification of Instruction sets



# Classification of 8085 Instruction sets



# Instruction format

- ▶ Depending on the number of address specified, the Instruction format can be classified into three categories:
  - One Byte Instruction:
    - Here one byte will be operand.
    - E.g MOV A,B
  - Two Byte Instruction:
    - First byte will be opcode and second byte will be operand/data.
    - E.g MVI A, data
  - Three Byte Instruction:
    - First byte opcode and second , third byte will be operand/data.
    - E.g LXI B,4050H

# Instruction Set of 8085

- An instruction is a binary pattern designed inside a microprocessor to perform a specific function.
- The entire group of instructions that a microprocessor supports is called *Instruction Set*.
- 8085 has **246** instructions.
- Each instruction is represented by an 8-bit binary value.
- These 8-bits of binary value is called *Op-Code* or *Instruction Byte*.





Intel 8085 uses the following addressing modes:

1. Direct Addressing Mode
2. Register Addressing Mode
3. Register Indirect Addressing Mode
4. Immediate Addressing Mode
5. Implicit Addressing Mode

# 1. Direct Addressing Mode



In this mode, the address of the operand is given in the instruction itself.

<b>LDA 2500 H</b>	<b>Load the contents of memory location 2500 H in accumulator.</b>
-------------------	--

- LDA is the operation.
- 2500 H is the address of source.
- Accumulator is the destination.



## 2. Register Addressing Mode



In this mode, the operand is in general purpose register.

**MOV A, B**

**Move the contents of register B to A.**

- MOV is the operation.
- B is the source of data.
- A is the destination.

### 3. Register Indirect Addressing Mode



In this mode, the address of operand is specified by a register pair.

<b>MOV A, M</b>	<b>Move data from memory location specified by H-L pair to accumulator.</b>
-----------------	---

- MOV is the operation.
- M is the memory location specified by H-L register pair.
- A is the destination.

## 4. Immediate Addressing Mode

In this mode, the operand is specified within the instruction itself.

<b>MVI A, 05 H</b>	<b>Move 05 H in accumulator.</b>
--------------------	----------------------------------

- MVI is the operation.
- 05 H is the immediate data (source).
- A is the destination.



## 5. Implicit Addressing Mode



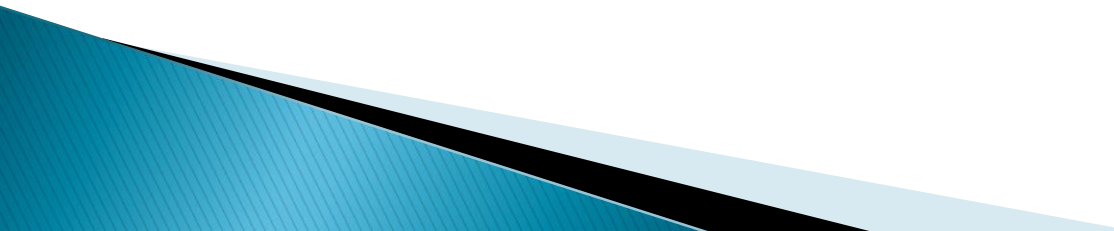
If address of source of data as well as address of destination of result is fixed, then there is no need to give any operand along with the instruction.

**CMA**


**Complement accumulator.**

- CMA is the operation.
- A is the source.
- A is the destination.


# Writing or Assembling a program

- ▶ Step 1: Read problem carefully
  - ▶ Step 2: Break it down into small steps
  - ▶ Step 3: Represent these small steps in a possible sequence with a flow chart
  - ▶ Step 4: Translate the block of flowchart into appropriate mnemonic instructions
  - ▶ Step 5: Translate mnemonic into machine code
  - ▶ Step 6: Enter the machine code in memory and execute
  - ▶ Step 7: Start troubleshooting(debugging a program)
- 

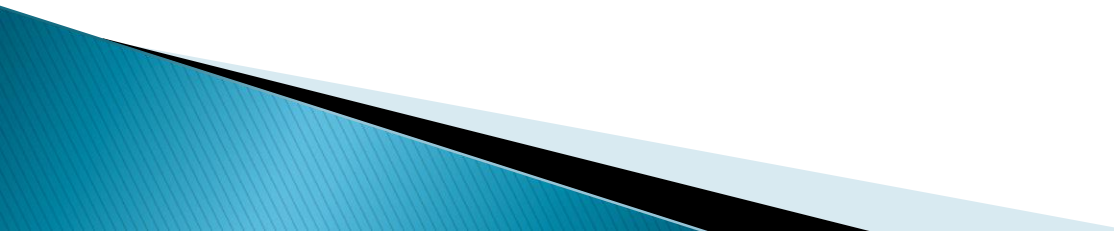
# Executing a program

- ▶ The machine code can be loaded into read write memory starting with some memory address
  - ▶ The execution of program can be done in two ways
    - 1. The first is to execute entire code by pressing execute key
    - 2. Second is to use the single step key. It executes one instruction at time. By using examine register key we can observe the contents of registers and flags after each instruction being executed.
- 

# Debugging a program


- ▶ Debugging binary code is much more difficult and cumbersome.
  - ▶ Essentially we search carefully for the errors in the program logic, machine codes and execution
  - ▶ The debugging process can be divided into two parts,
    - Static Debugging: Visual Inspection of a flowchart and machine code
    - Dynamic Debugging: observing output or register contents following a execution of each instruction (Single step technique) or Group of Instructions (Breakpoint Technique)
- 

# Developing Counters and Time Delay Routines

- ▶ Designing counter is a frequent programming application.
  - ▶ Counters are used to keep track of event.
  - ▶ Time delays are important to set up reasonably accurate timing between two events.
  - ▶ Process of designing counters and time delay using software instructions is more flexible and less time consuming than design process using hardware.
- 

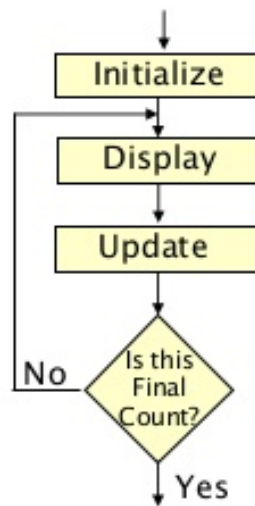


# Counter

- ▶ A counter is designed simply by loading an appropriate number into one register and using INR(increment by 1) and DCR (decrement by 1) instructions.
  - ▶ A loop is established to update a the count and each count is checked whether it has reached final number. If not, loop is repeated.
  - ▶ Drawback of such counter is counting perform at very high speed that only last count is observed.
  - ▶ To observe counting time delay between count must be used.
- 

# Counter

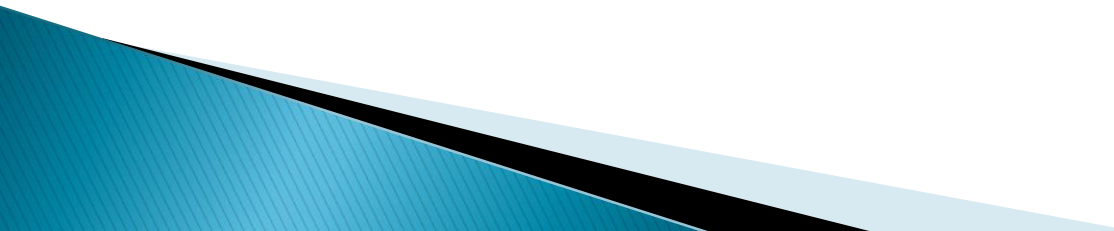
## COUNTERS



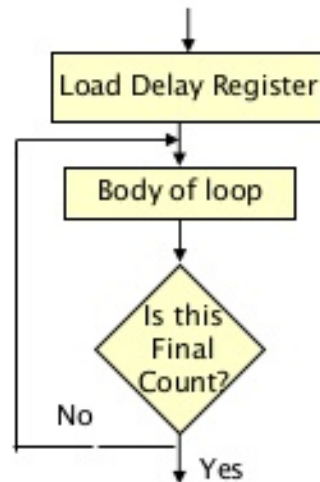
Flowchart of a Counter

A Counter is designed simply by loading an appropriate number into one of the registers and using the INR (Increment by one) or the DCR (Decrement by one) instructions. A loop is established to update the count, and each count is checked to determine whether it has reached the final number; if not, the loop is repeated.

# Time delay

- ▶ Designing Time delay is similar to that used to set up counter.
  - ▶ The register is loaded with number depending on time delay required.
  - ▶ And register is decremented until it reaches zero by setting up loop with conditional jump instruction.
  - ▶ The loop causes delay depending on clock period of system.
- 

# TIME DELAYS



Flowchart of a Time Delay

The procedure used to design a specific delay is similar to that used to set up a counter. A register is loaded with a number, depending on the time delay required, and then the register is decremented until it reaches zero by setting up a loop with a conditional jump instruction. The loop causes the delay, depending upon the clock period of the system.















































