Unit-3
Interrupts and Assembly Language Programming

3.1 Define Interrupt

Interrupt: The process of **interrupting** the microprocessor to stop normal program execution to carry out a specific task/work is referred to as **interrupt.**

(or)

The event or signal which diverts the normal program execution, in order to carry out a specific task/work is called **interrupt.**

The processor can be interrupted in the following ways.

i. By an external signal generated by a peripheral.
ii. By an internal signal generated by a special instruction in the program.
iii. By an internal signal generated due to an exceptional condition which occurs while executing an instruction.

For Eg: Divide by zero is exceptional condition which initiates type-0 interrupt

When a microprocessor receives an interrupt signal it stops executing current normal program, save the status of various registers (IP, CS and flag registers) in stack and then processor executes a subroutine in order to perform the specific task requested by the interrupt.

ISR (Interrupt Service Routine): The procedure that is executed in response to an interrupt is called Interrupt Service Routine (ISR).At the end of ISR, the stored status of registers in stack are restored to respective registers, and then resumes the normal program execution from the point where it was interrupted.

3.2. Need of Interrupts:

- The external interrupts are used to implement interrupt driven data transfer scheme.
- Software interrupts are used to implement system calls.
- The interrupts generated by exceptional conditions are used to implement error conditions in the system
- The speed of the processor will be improved

Interrupt driven data transfer scheme:

- The interrupts are useful for efficient data transfer between processor and peripheral;
- When a peripheral is ready for data transfer, it interrupts the processor by sending an appropriate signal. Upon receiving an interrupt signal, the processor suspends the current program execution, save the status in stack and executes an ISR to perform the data transfer between the peripheral and the processor.
- At the end of ISR, the stored status of registers in stack are restored to respective registers, and then resumes or continues the normal program execution from the point where it was interrupted.
- This type of data transfer scheme is called **Interrupt driven data transfer scheme**. There are two methods are available for data transfer between processor and peripheral devices, they are:

  1) Polling technique.

  2) Interrupt method.

Polling technique:-

- The processor has **to periodically poll(or)check the status/readiness** of the devices.
- In polling technique the **processor time is wasted** ,because the processor has to suspend its work and check the status of the device in predefined intervals

Interrupt method:-

- If the device interrupt the processor to **initiate a data transfer whenever it is ready** then processor time is effectively utilized.
- processor need not suspend it work and need not to check the status of the devices.

3.3 Classify the Interrupts

In general the interrupts can be classified in the following 3- ways.
1. Hardware and Software interrupts.
2. Vectored and non-vectored interrupts.
3. Maskable and non-maskable interrupts.

1. Hardware and Software Interrupts

The interrupts initiated by external hardware by sending an appropriate signal to the interrupt pin of the processor is called hardware interrupt. The 8086 processor has 2-pins INTR and NMI to accept the hardware interrupts.

The Software interrupts are program instructions. These instructions are inserted at desired locations in a program. While running a program, if software interrupt instruction is encountered then the processor initiates an interrupt. The software interrupts instructions are INT n, where n is the type number ranging from 0 to 255.

2. Vectored and Non-Vectored Interrupts:

When an interrupt signal is accepted by the processor, if the program control automatically branches to a specific address called vector address, then the interrupt is called **vectored interrupt**. The automatic branching to vector address is predefined by the manufacturer of processor. All the 8086 interrupts are vectored interrupts.

In **non-vectored interrupts** the interrupting device should supply the address of the ISR to be executed in response to the interrupt.

3. Maskable and Non-Maskable Interrupts:

The interrupts whose request can be either **accepted or rejected by the processor** are called **Maskable interrupts.** We can make all the hardware interrupts initiated through INTR pin Maskable by clearing IF (Interrupt Flag).

The interrupts whose request **has to be definitely accepted or cannot be rejected by the processor** are called **Non-Maskable interrupts**. The interrupts initiated through NMI pin and all software interrupts are non-Maskable.

3.4 Understand the Interrupts of 8086.

The 8086 has 256 types of interrupts. INTEL has assigned a type number to each interrupt. The type numbers are in the range of 0 to 255. The interrupts can be initiated either by executing "INT n" instruction, where n is the interrupt type number or the interrupt can be initiated by sending an appropriate signal to INTR input pin of the processor.

For the interrupts initiated by software instruction "INT n", the type number is specified by the instruction only. When the interrupt is initiated by INTR pin, then the processor runs an interrupt acknowledge cycle to get the interrupt type number.
In this 256 interrupts, INTEL has defined the functions of first five interrupts, i.e., the interrupts type-0 to type-4 are dedicated for specific functions by INTEL and they are called INTEL predefined interrupts.
The next 27 interrupts, i.e., from type-5 to type-31 are reserved by INTEL for use in future microprocessors or for system calls/services. The remaining upper 224 interrupts, i.e., from type-32 to type-255 are available for the user as hardware or software interrupts.

INTEL predefined or Dedicated interrupts:
The INTEL has defined the functions of the first five types of interrupts:

- Division by zero (Type-0 interrupt).
- Single step (Type-1 interrupt).
- Non-Maskable interrupts, NMI (Type-2 interrupt).
- Breakpoint Interrupt (Type-3 interrupt).
- Interrupt on overflow (Type-4 interrupt).

These predefined interrupts are only defined by INTEL; Intel has not provided any subroutine/procedure to be executed for these interrupts. The system designer has to write interrupt service routines (ISR) for each interrupt and store them in memory and the corresponding address has to be stored in Vector table.

1. Division by zero (Type-0 interrupt): This interrupt is implemented as part of execution of division instruction. The 8086 will automatically perform a Type-0 interrupt if the result of division operation is too large to fit in destination (i.e. infinity) and this interrupt is non-maskable. So we have to write an ISR which takes the desired action or indicate an error condition when an invalid division occurs. The ISR should be stored in memory and the address of ISR is stored in interrupt vector table.

2. Single Step (Type-1 interrupt): When the trap flag (TF) is set to 1, the processor will automatically generate a type-1 interrupt after execution of each instruction. The user can write an ISR for type-1 interrupt to halt the processor temporarily and return the control to the user after execution of each instruction. Execution of one instruction by one instruction is known as single step, and this feature is used to debug the program.

3.NMI: Non-Maskable Interrupt (Type-2 interrupt): The 8086 processor automatically generate a Type-2 interrupt when it receives a high signal on its NMI pin, this interrupt cannot be disabled or masked. Usually the type-2 interrupt is used to save program data or processor status in case of system AC failure. When power fails, the voltage supplied to the computer may remain for 50ms due to capacitors provided in the circuit. This time is sufficient to save the program and data. The program along with data is restored when power returns and can be executed again from the point where it was interrupted.

4. Break Point Interrupt (Type-3 Interrupt): Type 3 interrupt is a breakpoint interrupt. The type-3 is used to implement a breakpoint function that executes a program partly or up to the desired point and then returns the control to the user. The breakpoint interrupt is initiated by execution of INT 3 instruction.

5. Overflow Interrupt (Type-4 interrupt): In the 8086 processor the overflow flag will be set if the signed arithmetic operation generates a result whose size is larger than the size of the destination register. During such conditions the type-4 interrupt can be used to indicate an error condition. The type-4 interrupt is initiated by "INTO" instruction. If OF is not set the INTO instruction simply performs no operation. If OF is set to 1, the INTO instruction performs type 4 interrupt.

Software Interrupts of 8086: The interrupts that are raised by "INT n" instruction are called **software interrupts**. The "INT n" will initiate type-n interrupt and the value of n is in the range of 0 to 255. Therefore all the 256 types including the INTEL predefined interrupts can be initiated through INT n instruction. The software interrupts are non-maskable and have higher priority than hardware interrupts.
Hardware interrupts of 8086: The interrupts generated by applying appropriate signals to INTR and NMI pins of 8086 are called **hardware interrupts**. The hardware interrupts initiated through INTR are Maskable by clearing Interrupt flag (IF=0). These interrupts have low priority than software interrupts. The hardware interrupts NMI is non Maskable and has higher priority than interrupts initiated through INTR. The NMI is initiated by Low to high transition of the signal applied to NMI pin of the processor.

Priorities of interrupts of 8086: The priorities of the interrupts of 8086 are shown in the table. The 8086 processor checks for internal interrupts before it checks for any hardware interrupts. Therefore software interrupts has higher-priority than hardware interrupts. But the processor can accept the NMI interrupt request and execute a procedure for higher priority interrupt.

Fig.Interrupt priority

Interrupt	Priority
Divide Error, INT(n),INTO	Highest
NMI	↓
INTR	↓
Single Step	Lowest

3.2 Explain the interrupt handling process in 8086:

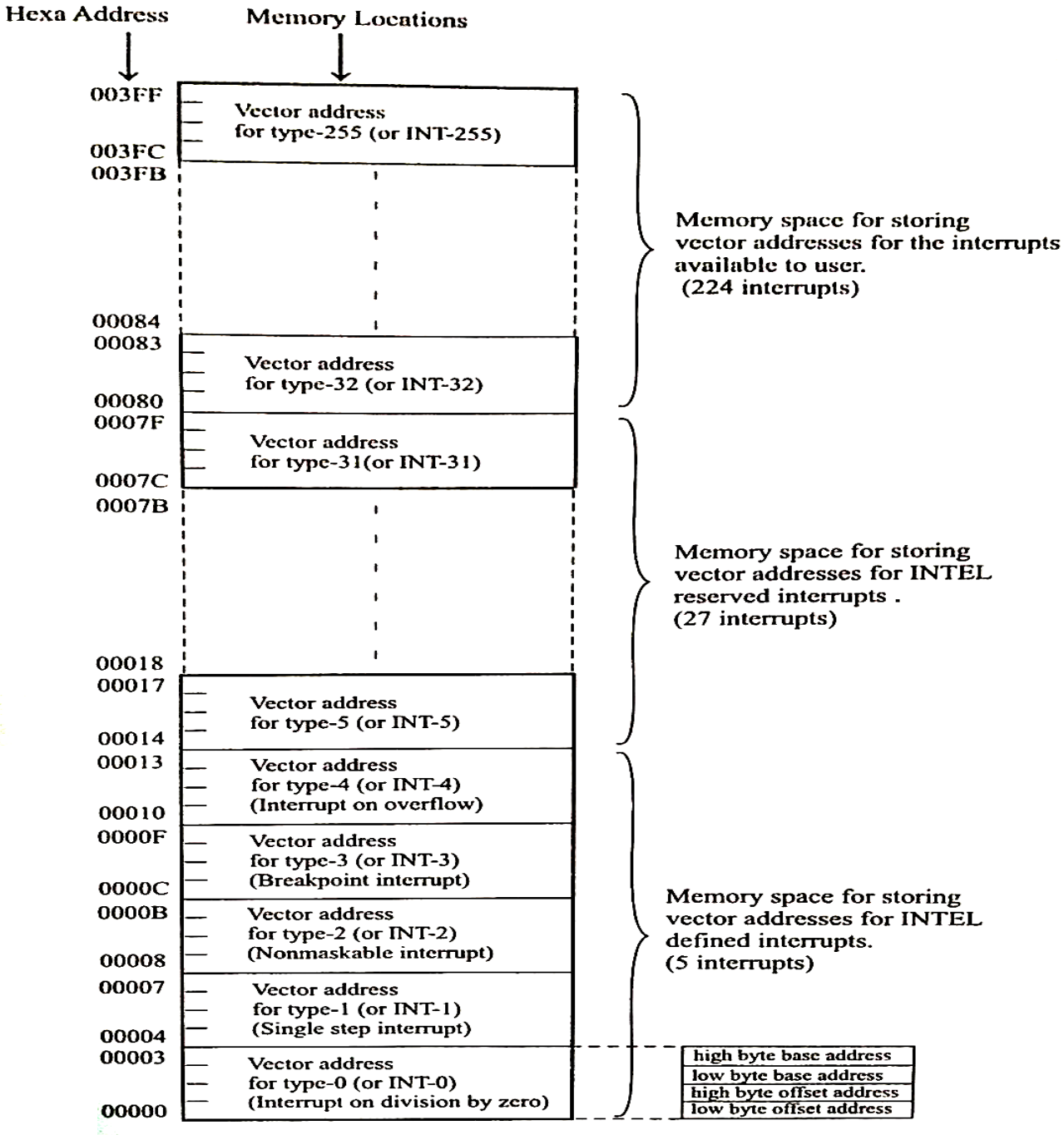
The 8086 has 256 types of interrupts and these interrupts can be implemented either as hardware or software interrupts. The number of interrupts to be implemented and used in a system depends on system designer and also the application for which the system is designed. The choices of implementing the INTEL predefined interrupts are also depend on system designer.

Interrupt Vector Table (IVT):

For each and every interrupt decided to implement in the system, the system designer has to write an interrupt service routine (ISR) and store them in memory. Then the system designer has to create an interrupt vector table in the first 1KB of memory space (i.e. in the memory space with address range of 00000H to 003FFH) of the 8086 system.

In this vector table the 16-bit offset address and 16-bit segment base address of each ISR are stored in four consecutive memory locations. The address stored in this table is called vector addresses. For storing the vector addresses of all the 256 interrupt type, the vector table requires (256*4=1KB) IKB memory space.

The vector address of an interrupt is stored in four consecutive memory locations starting from this 20-bit address. The first two locations are used to store the low and high byte of offset address, and next two locations are used to store low byte and higher byte of segment base address of ISR to be executed for an interrupt. The organization of interrupt table of 8086 based system is given below;



Servicing an Interrupt by 8086:

The 8086 processor checks for an interrupt request at the end of each instruction cycle. If an interrupt request is detected, then the processor responds to the interrupt by performing the following operations.

1. The SP is decremented by two and the content of flag register is pushed to the stack memory.
   [SP← SP – 2 ; [SP] ← Flag register]
2. The interrupt system is disabled by clearing interrupt flag (IF).
   [IF← 0]
3. The single-step execution is disabled by clearing trap flag (TF).
   [TF← 0]
4. The stack pointer is decremented by two and the content of CS register is pushed onto the stack memory.
   [SP← SP – 2 ; [SP] ← CS]
5. Again, the stack pointer is decremented by two and the content of IP register is pushed to stack memory.
   [SP← SP – 2 ; [SP] ← IP]
6. In case of hardware interrupt through INTR, the processor runs an interrupt acknowledge cycle to get the interrupt type number. For software interrupt, the type number is specified in the instruction itself. For NMI and other exceptions the type number is defined by INTEL.

7. The processor generates a 20-bit memory by multiplying the type number by four and sign extending to 20-bit. This memory address is the address of interrupt vector table, where the vector addresses of interrupt service routine (ISR) is stored by the user/System designer.

8. The first word pointed by vector table address is loaded in IP and the next word is loaded in CS register. Now the content of IP is the offset address and the content of CS register is the segment base address of the ISR to be executed.
   [IP ← Offset of ISR From IVT]
   [CS ← Base Address of ISR From IVT]

9. The 20-bit physical memory address of ISR is calculated by multiplying the content of CS register by 10 and adding to the content of IP.

10. The processor executes the ISR to service the interrupt.

11. The ISR will be terminated by IRET instruction. When this instruction is executed, the top of stack is popped to IP, CS and flag register one word by one word. After every pop operation, the SP is incremented by two.
    [SP← SP + 2 ; IP ← [SP]]

    [SP← SP + 2 ; CS ← [SP]]

12. Thus, at the end of ISR, the previous status of the processor is restored and so the processor will continue execution of normal program from the instruction where it was suspended.

(Unit 3 second part)

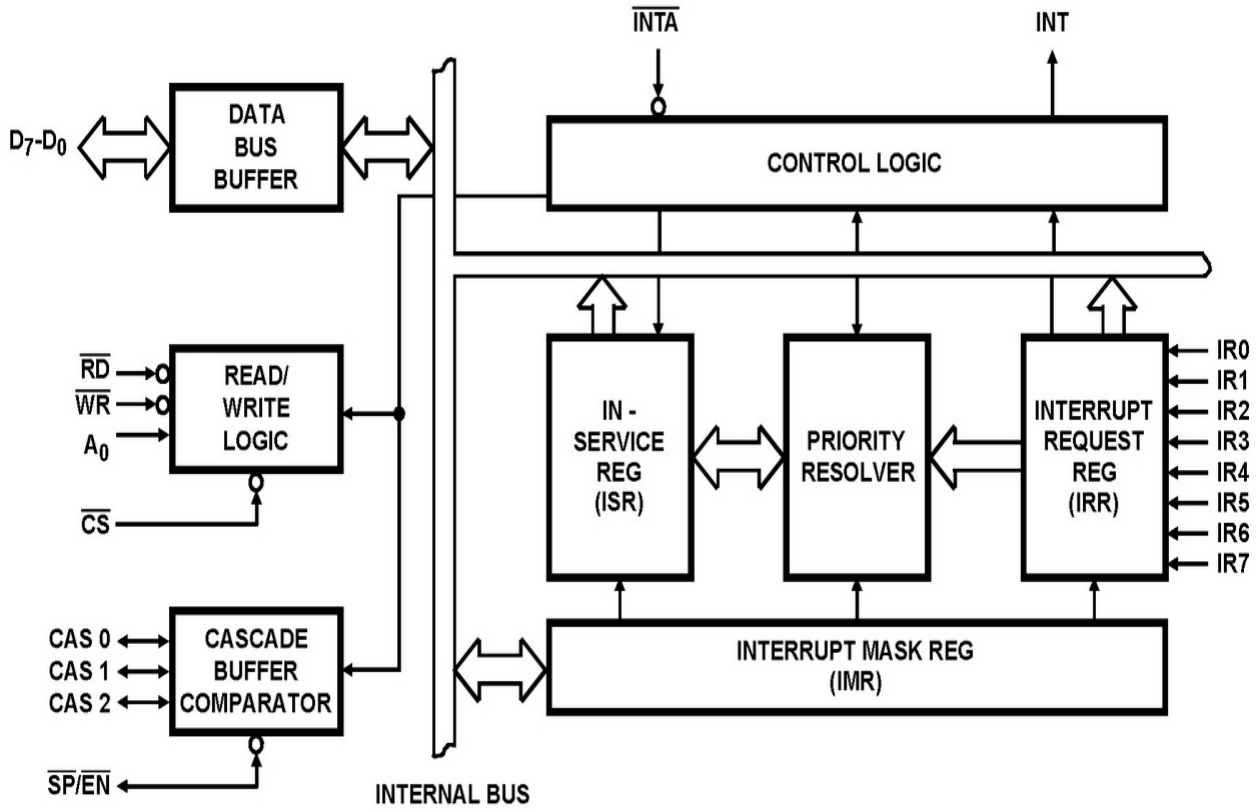
Features of 8259:

The 8259 is a programmable interrupt controller used to expand the interrupts of the 8085 or 8086 processor.

- It is programmed to work either 8085 or 8086.
- It manages 8 interrupt requests according to the instructions written in control registers.
- In 8086 processor based system it supplies type number and type number is programmable.
- In 8085 processor based system it supplies vector address.
- The interrupts can be masked or unmasked individually.
- The priorities of the interrupts are programmable.

Functional Block Diagram of 8259

The functional block diagram of 8259 is as shown below. It has 8 functional blocks. They are Control Logic, Read/Write logic, Data bus buffer, Interrupt Request Register (IRR), In-Service Register (ISR), Interrupt Mask Register (IMR), Priority Resolver (PR), and Cascade Buffer.



1. **Interrupt Request Register (IRR):** IRR stores all the interrupt request inputs in it, in order to serve them one by one on the priority basis.

2. **In-Service Register (ISR):** This stores all the interrupt requests those are being served, i.e. ISR keeps a track of the requests being served.

3. **Priority Resolver:** priority resolver determines the priorities of the interrupt requests.IR0 have the highest priority while the IR7 has the lowest one. The highest priority is selected and stored.

4. **Interrupt Mask Register (IMR):** This register stores the bits required to mask the interrupt inputs. IMR operates on IRR at the direction of the Priority Resolver.

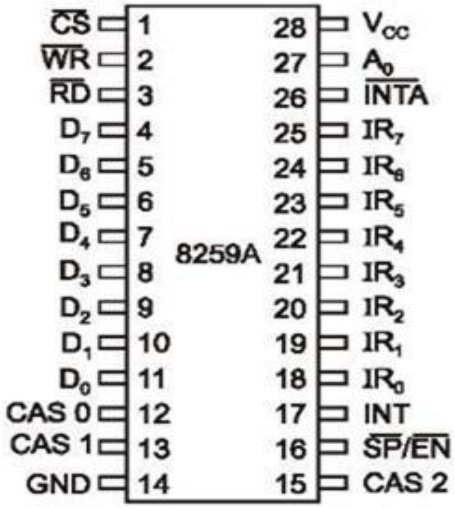
5. **Interrupt Control Logic:** This block manages the interrupt and interrupt acknowledge signals to be sent to the CPU for serving one of the eight interrupt requests. This also accepts the interrupt acknowledge (INTA) signal from CPU.

6. **Data Bus Buffer:** The data bus buffer interfaces internal 8259A bus to the microprocessor system data bus. Control words, status and vector information pass through data buffer during read or write operations.

7. **Read/Write Control Logic:** This circuit accepts and decodes commands from the CPU. This block also allows the status of the 8259A to be transferred on to the data bus.

8. **Cascade Buffer/Comparator:** the cascade buffer /comparator is used to connect 8259 in cascade mode, the 8259 which will be directly interrupting 8086 is called master. The 8259s which interrupting the master 8259 are called slaves. Each interrupt request is passed to INTR pin of processor through master only. Multiple 8259 chips can be connected in cascaded mode by using CAS0-CAS2 pins.

Pin Diagram (8259): The programmable interrupt controller(8259) is a 28 pin Dual in-line packaged IC fabricated with NMOS technology.



1. $\overline{CS}$**:** A low on this input enables 8259A to be selected. No reading or writing occurs unless the device is selected.

2**.** $\overline{WR}$**:** This pin is an active-low write enable input to 8259A. This enables to write command words (ICWS &OCWS) to 8259 chip.

3**.** $\overline{RD}$**:** This is an active-low read enable input to 8259A. A low on this pin enables 8259A to read status on the data bus of CPU.

4. **D0-D7:** These pins from a bidirectional data bus that carries 8-bit. This also carries interrupt vector information.

5. **CAS0 – CAS2**: **Cascade Lines:**
A signal 8259A provides eight vectored interrupts. If more interrupts are required, the 8259A is used in cascade mode. In cascade mode, a master 8259A along with eight slaves 8259A can provide up to 64 vectored interrupt lines. These three lines act as select lines for addressing the slave 8259A.

6. $\overline{SP}/\overline{EN}$ **(slave program/enable):** This pin is a dual purpose pin. When the chip is used in buffered mode, it can be used as buffered enable to control trans-receivers. If this is not used in buffered mode then the pin is used as input to designate whether the chip is used as a master ($\overline{SP}$=1) or slave ($\overline{SP}$=0).

7. **INT:** This pin goes high whenever a valid interrupt request is asserted. This is used to interrupt the CPU and is connected to the interrupt input of CPU.

8. **IR0 – IR7 (Interrupt requests)**: These pins acts as inputs to accept interrupt request from the peripheral devices. The 8259 will able to accept interrupt requests in either level triggered mode or positive edge triggered mode.

9.$\overline{INTA}$ **(Interrupt acknowledge):** This pin is an input used to strobe-in 8259A interrupt vector data on to the data bus. In conjunction with CS, WR and RD pins, this selects the different operations like, writing command words, reading status word, etc.,

Interfacing of 8259 with 8086:

Circuit initialization:

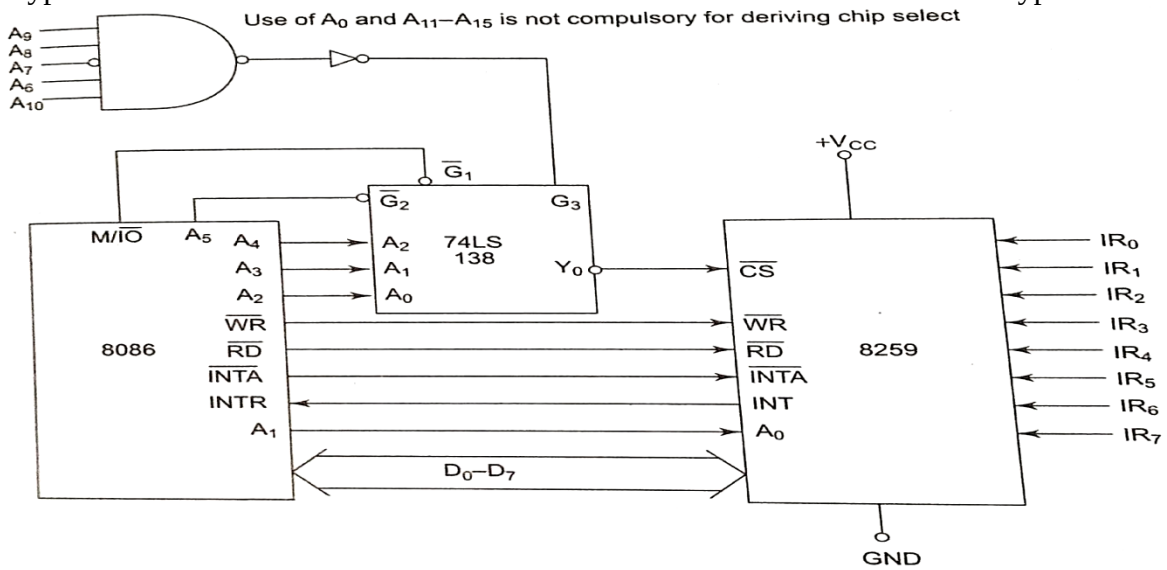
1. The lower order data lines D0-D7 of 8086 are connected to D0-D7 of 8259A
2. The chip select signal for 8259A is generated by using 3 to 8 decoder. Where the address lines A4, A3, A2 are used as input to decoder.
4. The control signal M/$\overline{IO}$ &A5 are used as enable for decoder, connected to $\overline{G1}$ and $\overline{G2}$ respectively.
3. The address line $\mathbf{A_1}$ of the 8086 processor is connected to $\mathbf{A_0}$ of 8259 to provide the internal address.
4. The $\overline{\text{INTA}}$ pin of 8086 is connected to $\overline{\text{INTA}}$ pin of 8259A.
5. The $\overline{\text{RD}}$, $\overline{\text{WR}}$ signals of 8086 are connected to $\overline{\text{RD}}$ and $\overline{\text{WR}}$ pins of 8259A.
6. The INTR pin of 8086 is connected to INT pin of 8259A.

Working:

1. The 8259 should be programmed by sending initialization command word (ICW) and operational command word (OCW).
2. Once 8259 is programmed it is ready for accepting interrupt signals.
3. When it receives an interrupt through any one of the interrupt lines IR0-IR7, it checks for priority and also checks whether it is masked or not.
4. If the previous interrupt is completed and if the current request has highest priority and unmasked, then it is serviced.

Processing of interrupts by 8259:

1. For servicing the interrupt the 8259 will send INT signal to INTR pin of 8086, in response to it, the 8086 sends the acknowledge signal on $\overline{\text{INTA}}$ pin to the $\overline{\text{INTA}}$ pin of the 8259.
2. When the processor accepts the interrupt, it sends two $\overline{\text{INTA}}$ pulses one by one.
3. The first INTA is send to 8259, to inform the acceptance of interrupt and to prepare 8259 for supplying the type number. The second INTA is send to 8259 to read the type number from



8086

4. Once the 8086 receives the type number, it starts processing the interrupt. The 8086 processor multiply the type number by four to generate vector table address.
5. From this vector table, the vector addresses of the interrupt type requested are read and loaded in IP and CS register.
6. Then the processor executes the ISR stored in this address.

Programming of 8259A

Initialization Command Words (ICW):

Before it starts functioning, the 8259A must be initialized by writing **two** to **four** command words into the respective command word registers. These are called as **initialization command words**. The initialization sequence of 8259A is described in the form of a flow chart in below fig:

The command words of 8259A are classified in two groups

1. ***Initialization command words (ICW)*** and
2. ***Operation command words (OCW)***.

- **The ICW'S are used to initialise the 8259:**
- Specify whether 8259 operates the interrupts in Edge/Level Triggered mode.
- Specify whether the 8259 is connected in Single/Cascade mode.
- Specify Call Address Interval in case of 8085.
- Specify whether the processor is 8086 or 8085.
- Vector address or type number
- **The OCW'S are used read status of interrupts & program the following features**
- Masking or unmask the interrupts.
- Assigning Priority.
- Specific or non specific end of interrupt.

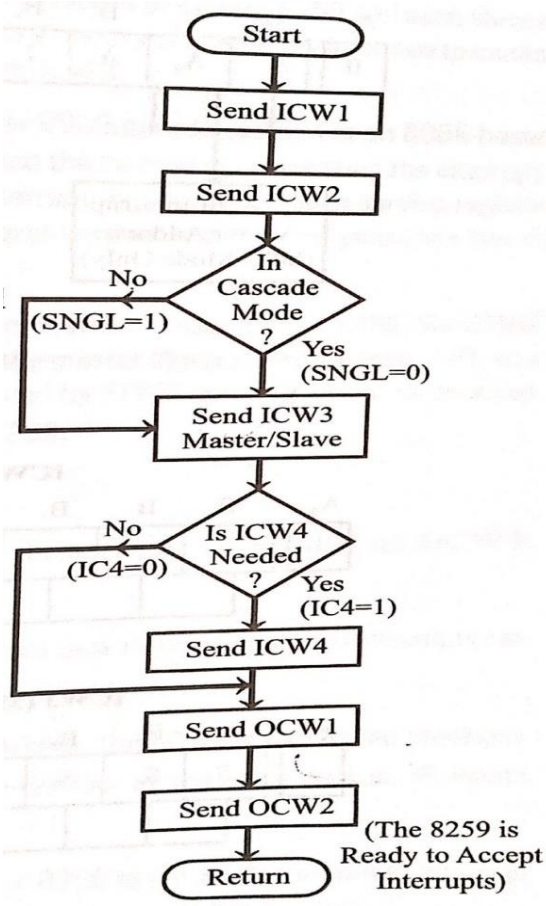
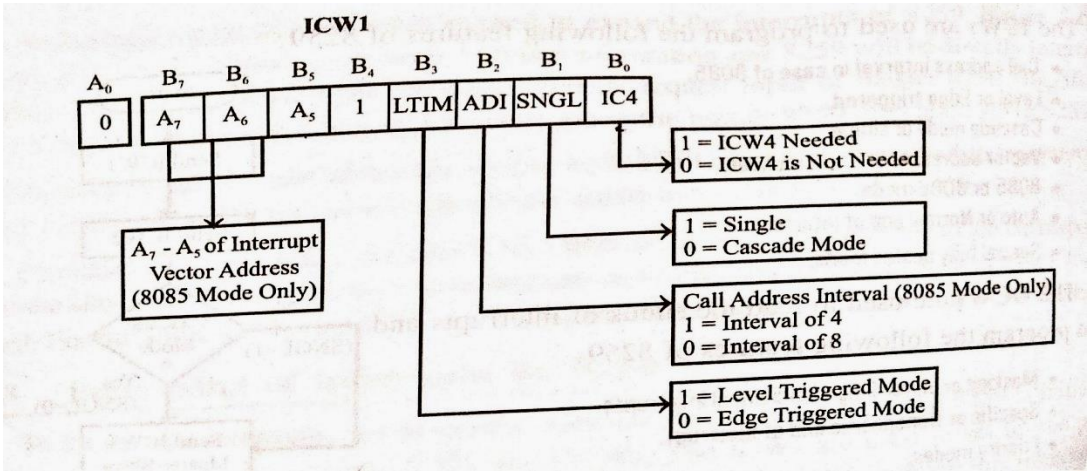


Figure : flowchart for initialising 8259A

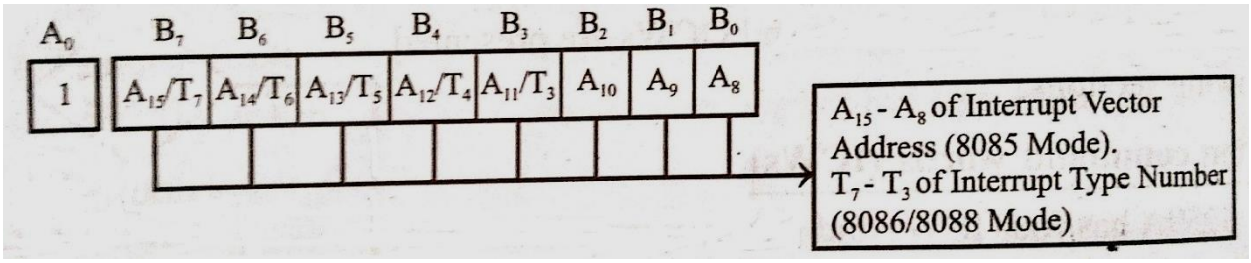
There are four types of **ICW**'s are there they are **ICW1, ICW2, ICW3** and **ICW4.**

ICW1: ICW1 must given to the 8259. The ICW1 programs the basic operations. When ever the A0 pin and B4 bits are'1' the command word is treated as a ICW1. The address bits A_7 to A_5 bits are uesd as a vector address for 8085 microprocessor, and for 8086 those bits are dontcare's.



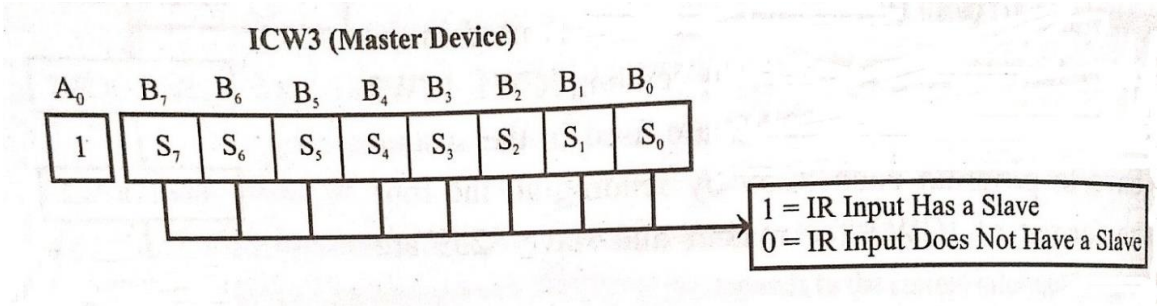
ICW2:

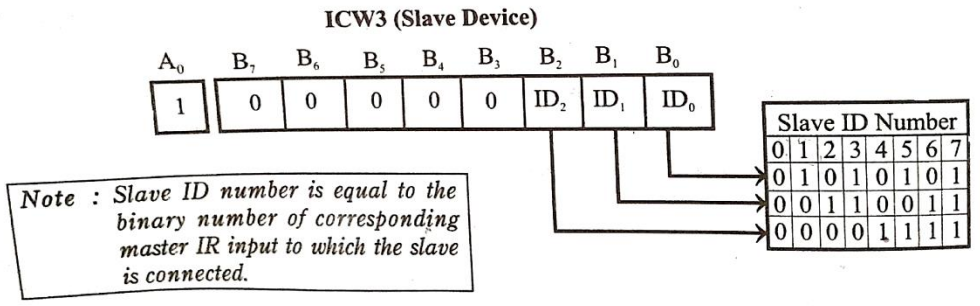
If **A_0** =1, the control word is recognized as ICW2. The ICW2 stores details regarding interrupt vector addresses. This control word stores the all interrupt related information, and the format is given below.



3. **ICW3:(Master)**

The ICW3 should be sent to 8259s in cascade mode. Separate formats are provided for master and slave 8259s. In cascade mode, slave 8259s are connected to master through IR pins of master.



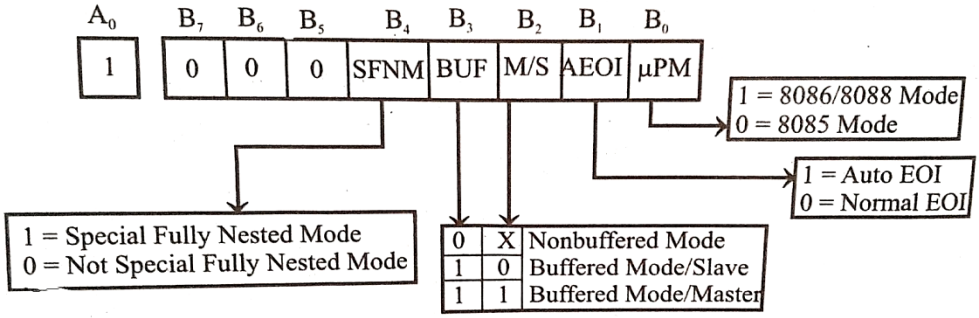


3. ICW4:

The ICW4 is used to inform 8259 whether it is connected to 8085 or 8086 based system.

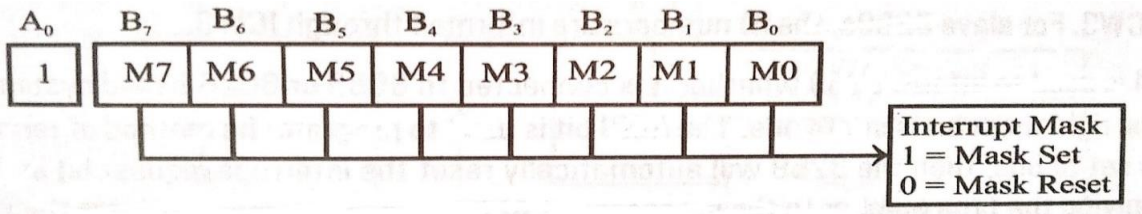
- The bit µPM used to define the processor
  1= 8086/8088
  0= 8085
- The bit SFNM defines, whether the 8259 operates in Special fully nested mode or not
  1= special fully nested mode
  0= not special fully nested mode
- The bits BUF and M/S used to define

BUF	M/S	
0	x	non buffered mode
1	0	buffered mode/slave
1	1	buffered mode/master



Operation command words (OCWS):
The 8259 has 3 operational command words.OCW1, OCW2&OCW3.

1. **OCW1:**
The OCW_1 is send to 8259 to mask or unmask the IR inputs of 8259.



2. OCW2:
The OCW_2 is send to 8259A only when the AEOI(automatic end of interrupt) mode in ICW4 is not selected. The OCW2 is send by the processor to decide on the type of end of interrupt EOI. The different methods of EOI are discussed here

1. Non-specific end of interrupt: This command is send by processor to 8259 to terminate the current interrupt being served by 8259.this resets the corresponding bit in in-service register of 8259 and allow the next higher priority interrupt.

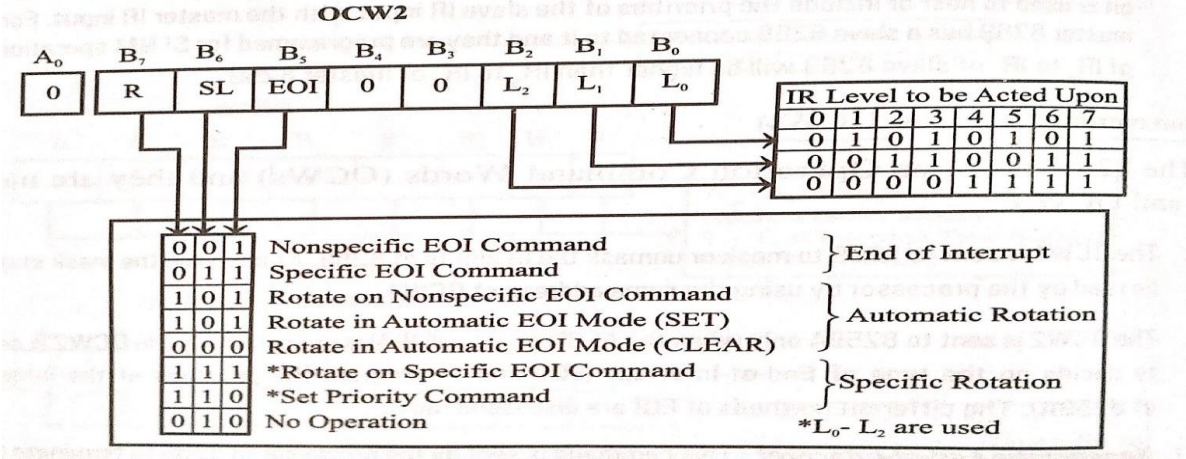
2. Specific end of interrupt: This command is send by the processor to reset or terminate a specific interrupt request decided by the lower three bits of ocw2

3. Rotate on non-specific EOI: This command will take action same as that of nonspecific EOI except that it rotates priorities after resetting bit in in-service register.

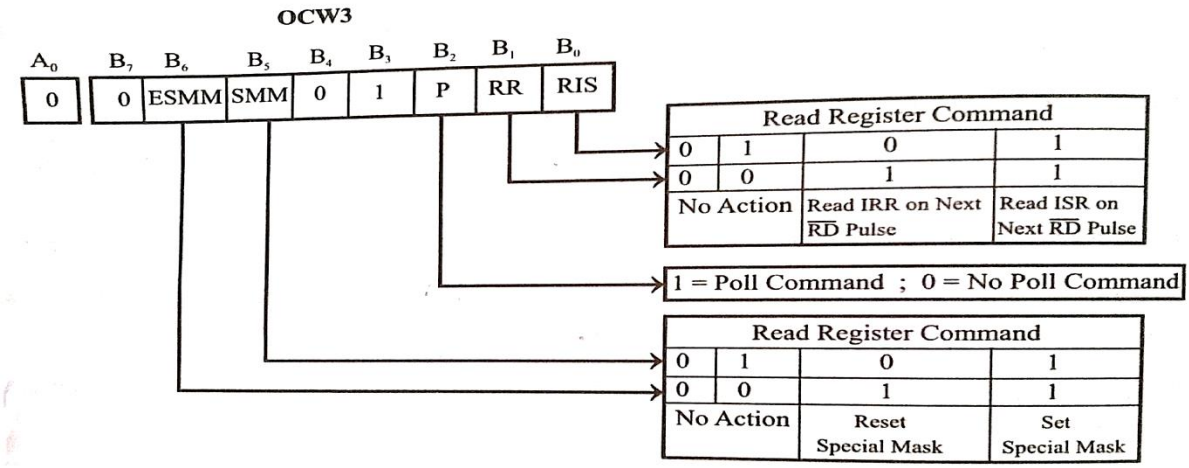
4. Rotate on automatic EOI: This command is send to 8259 to select automatic EOI with rotating priority

5. Rotate on specific EOI: This command will take action similar to that of specific EOI except that it rotates the priorities of interrupts after they are serviced

6. Set priority: The command is send to set priority of interrupt specified by lower three bits of OCW2.



3. **OCW3:**



3.7 Understand the significance of Assembly Language Programming:

Programming means writing set of instructions needed to perform a particular task. Basically there are three types of programming are there, they are

1. Machine level programming
2. Assembly level programming
3. High level programming.

➢ **Machine level programming:** In this programming the instructions are written by using the binary codes which ueses only 0's and 1's.the language in which the instructions are represented in binary codes are called **'machine language'.** The microprocessor can understand and execute the machine language programs directly.

➢ **Assembly language programming:** In assembly language programming the instructions are written using the mnemonics. A mnemonic represents the which operation to be performed by the processor. The language in which the instructins are represented by mnemonics is called **'assembly language'**.

➢ **High level language:** In high level programming the instructions will be in the form of statements written using symbols, english words. Some of the examples are C,C++.....

Advantages of assembly language program:

- It takes **less computation time** and memory
- It needs **less processing power** of cpu
- It **runs faster** than high-level language
- In machine language **thousands of instruction codes for processor that difficult any error prone task is eliminated by writing programs in assembly language.**
- **Speed** is very high.

3.8 Describe the procedure for executing an assembly language program with assembler.

We have two software's for writing the assembly language program.

MASM (Microsoft Assembler) and

TASM (Turbo Assembler)

Before starting the process, ensure that all the files likeEDIT.ASM (editor), MASM.EXE (Assembler), LINK.EXE (Linker), and DEBUG.EXE (debugger) are available in the same directory in which we are working.

For executing the assembly language program, we have to follow the steps given below.

i. Entering or editing a program:

The first step in the process is to write an assembly language program using the following command.

D :/> EDIT FILENAME.ASM

NOTE: for every assembly language program, the extension .asm must be there.

ii. Assembling a Program:

An assembler reads and translates a source file that was created using editor into machine language such as binary code. The assembler reads the source file of our program from the disk where we saved it after editing.

An assembler creates 2-files and they are:

1. **Object file**:it contains the binary code for the instructions and information about the address of the instructions.

2.**List file**: it contains the total information of the source file including labels, offset addresses, Opcodes, memory allotment for different labels and directives.

The command for performing this operation is:

D:\MASM>MASM FILENAME.ASM (or) D:\TASM>TASM FILENAME.ASM

iii. Linking Process: A linker is a program used to join together several object files into one large object file. When writing large programs, it is usually much more efficient to divide the large programs into smaller subprograms. Each subprogram can be individually written, tested and debugged.

The linker produces a link file which contains the binary codes for all the combined subprograms. The linker also produces a link map file which contains the address information about the link files.

The command for performing this operation is:

D:\MASM>LINK FILENAME.OBJ (or) D:\TASM>TLINK FILENAME.OBJ

The linker now **converts the .obj file into .exe file**.

iv. Debugging Process:

Debugging is the process which allows us to load our object code program into system memory, execute the program and debug it.

The command for performing this operation is:

D:\MASM>DEBUG FILENAME.EXE (or) D:\TASM>TD FILENAME.EXE

In this way, the errors are identified and the program is finally executed and the outputs are obtained**.**

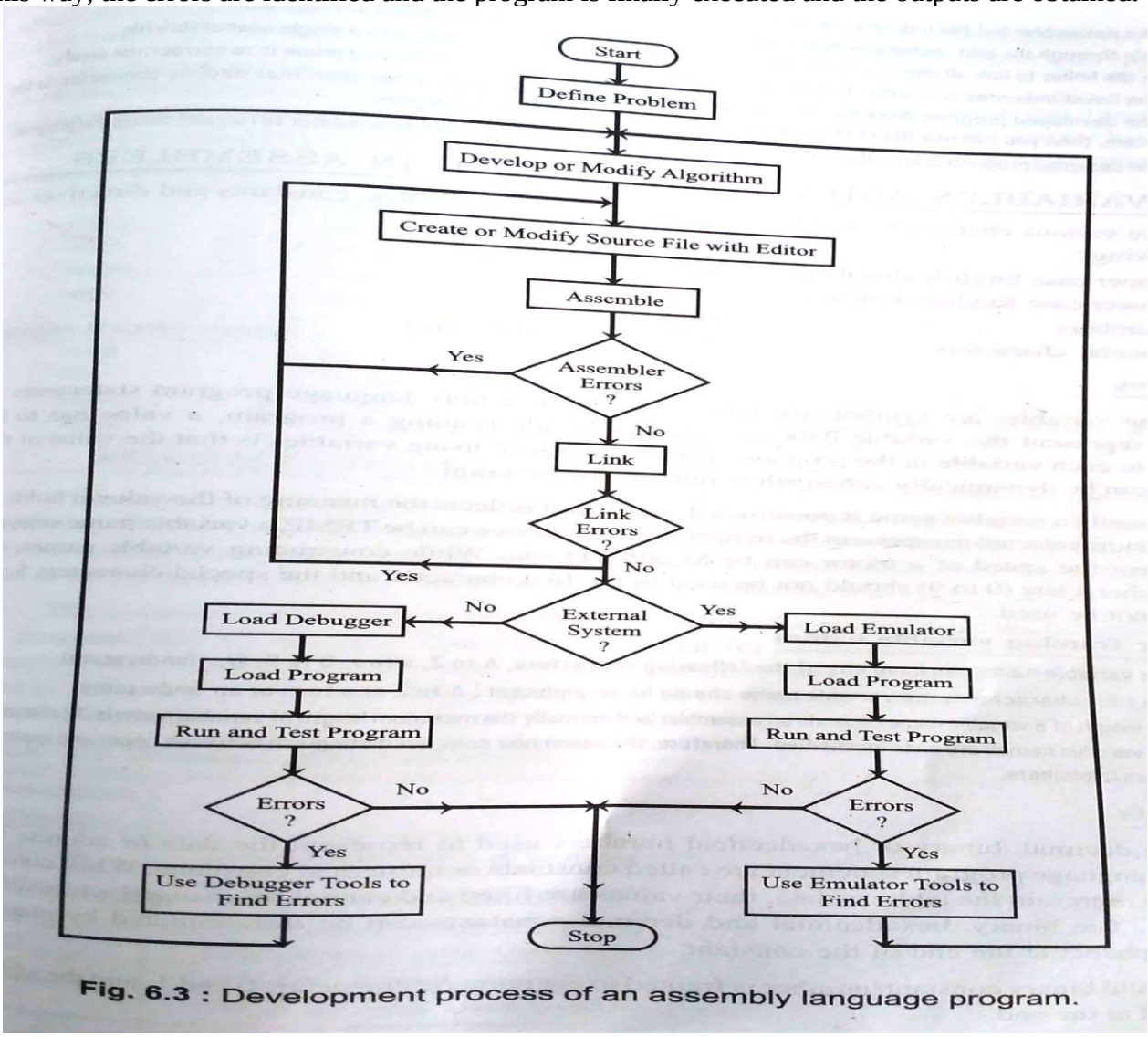


Fig. 6.3 : Development process of an assembly language program.

3.9 Explain Conditional and Loop Statements

Assembler supports the use of conditional statements and loop statements in the assembly language program. These statements control the flow of the program execution. Let us see conditional and loop statements in the assembly language programs.

Conditional Statements

1. IF - THEN

IF *condition* **THEN**
action

This structure says that IF the stated condition is found to be true, the series of actions following THEN will be executed. If the conditions is false, execution will skip the over the actions after THEN and proceed with the next mainline instruction.

Consider the following programs by implementing IF THEN statements. Generally this statement is implemented with a conditional jump instruction.

NOTE: only the basic part of the program will be specified in the following programs.

1. Write a program to compare two 16 bit quantities stored in AX&BX registers. If both contents are equal, then increment SI register.

Program

```
CMP AX, BX
JE GO
NOP
GO: INC SI
INT 03
```

2. Write a program to add two 16 bit numbers stored in AX&BX register. If no carry exists after addition increment SI register

Program

```
ADD AX, BX
JNC GO
NOP
GO: INC SI
INT 03
```

2. IF – THEN – ELSE

IF – THEN – ELSE structure is used to indicate a choice between two alternative actions.

General Format

IF *condition*
THEN *action*
ELSE *action*

1. Write a program to compare two 8 bit quantities, if both contents are not equal, transfer 04h into CL register otherwise increment SI register.

Algorithm

1. Read values in AH&BH
2. If AH is not equal to BH then put CL=04h
3. Otherwise increment SI register
4. End the program

Program

```
CMP AH, BH
JNZ GO
INC SI
NOP
GO: MOV CL, 04
INT 03
```

3. Multiple IF – THEN – ELSE

In different programming applications, we have to choose one of several alternative actions based on the values. For this purpose, we use multiple IF THEN ELSE statement

General Format

IF condition
THEN action
ELSE IF condition
THEN action
ELSE action

Program:

```
ADD AH, BH
CMP AH, 10H
JC L1
CMP AH, 40H
JC L2
MOV CL, 00H
NOP
L1: INC SI
L2: MOV CL, 04H
INT 03H
```

Loop statements:

Do while

Format:

WHILE condition is present
DO action

Algorithm:

1. Read the value in AL
2. While CL<1 do
   Mul CL

Program:

```
MOV AL, DATA
MOV CL, AL
DEC CL
L1: MUL CL
DEC CL   ; perform multiplication until CL=0
JNZ L1
INT 03H
```

3.11 Explain Procedural Programming in 8086.

Procedure (or) subprogram (subroutine):

Procedure or subprogram is a part of program within a larger program that performs a specific task often when writing programs we will find that we need a particular sequence of instructions at different points in a program to avoid writing the sequence of instructions in the program each time, we can write sequence as separate sub program called procedure. Whenever instructions in the procedure are required to be executed we use **CALL** instruction. The call instruction calls the procedure and **RET** instruction returns control from procedure to the main program.

The processor uses the stack to keep track of current pogram. when a procedure is called current value of IP is pushed on stack. When procedure returns value of stack is popped back into IP**. PROC** is an assembler directive used to indicate start of the subprogram or procedure. ENDP is used to indicate end of subprogram. In 8086 there are 2 types of subprogram

1. Near procedure 2. Far procedure

1. NEAR PROCEDURE:

If main program and subprogram are present in same code memory segment then the procedure is called a near procedure.

Program:

```
ASSUME CS: CODE
CODE SEGMENT
        :
CALL MUL
        :
INT 03
MUL PROC NEAR
        :
RET
MUL ENDP
CODE ENDS
END
```

2. **FAR PROCEDURE:**

If main program and subprogram are present in different code segment then the procedure is called far procedure.

Program:

```
ASSUME CS: CODE1
CODE1 SEGMENT
        :
CALL FAR MUL
        :
INT 03
CODE1 ENDS
ASSUME CS: CODE2
CODE2 SEGMENT
        :
MUL PROC
        :
RET
MUL ENDP
CODE2 ENDS
END
```

3.12 Illustrate CALL, RETURN statements and parameter passing:

1. CALL instruction

CALL instruction is used to enter into the procedure .The CALL instruction is used to transfer execution to sub program or procedure. There are two basic types of CALLs, ***near*** and ***far.***

General format:

CALL procedure name

Where procedure name is the name of procedure defined by user.

NEAR CALL:A **near call** is a call to a procedure which is in the same code segment. When the 8086 executes a near CALL instruction it decrements the stack pointer by two& pushes the offset of next instruction after the CALL on the stack (PUSH IP). It loads IP with the offset of the first instruction of the procedure in the same segment.

FAR CALL: A **far call** is call to a procedure which is in a different segment. When the 8086 executes a far CALL, it decrements the stack pointer by two and pushes the contents of the CS register to the stack. It then decrement the stack pointer by two again and pushes the IP value on to the stack.

2. RET instruction : The RET instruction will return the execution conrol from a procedure to the the main program where it was suspended. Up on execution of this RET instruction the stack pointer is incremented by 2 and the IP,CS, flag register's are reloaded with the stack contents respectively. There are two types of returns

Near return: The near return is used to transfer the program execution control from the near procedure to the main program. Up on execution of this instruction the stack pointer is incremented by 2 and the contents of the stack are moved into the IP.

Far return: The far return is used to transfer the program execution control from the far procedure to the main program. Up on execution of this instruction the stack pointer is incremented by 2 and the contents of the stack are moved into the IP, CS and flag register respectively.

Parameter Passing:

Parameter passing is a mechanism for communication of data and information between calling procedure to the called procedure. Procedures are called from main program. They usually get parameters (may be data values or address) from main program, process it and send back the output parameters to main program. There are four ways to pass parameters to and from the procedures. They are

- Using registers
- Using general memory
- Using pointers
- Using stack

1. Program for subtraction of two numbers using parameter passing through *registers.*

Program:

```
ASSUME  CS: CODE, DS: DATA
 DATA SEGMENT
A DB 05H
B  DB 02H
C DB ?
DATA ENDS
CODE SEGMENT
MOV  AL, A
MOV  BL, B
 CALL  SUBTRACTION
MOV  C,AL
INT  03H
SUBTRACTION  PROC
SUB  AL, BL
RET
SUBTRACTION  ENDP
CODE  ENDS
END
```

2. Program for subtraction of two numbers using parameters passing through *memory locations*

Program:

```
ASSUME  CS: CODE, DS: DATA
DATA SEGMENT
A  DB 33H
B  DB 55H
C  DB ?
DATA ENDS
CODE SEGMENT
MOV  AL, A
MOV  [8000H], B
CALL  SUBTRACTION
 MOV C,AL
INT 03H
SUBTRACTION  PROC
SUB  AX, [8000]
RET
SUBTRACTION  ENDP
CODE ENDS
END
```