ROPROCESSOR

- A Microprocessor is a multipurpose, Programmable clock-driven, register based electronic device that read binary instruction from a storage device called memory, accepts binary data as input and processes data according to those instructions and provides results as outputs.

- A Microprocessor is a clock driven semiconductor device consisting of electronic circuits manufactured by using either a LSI or VLSI technique.

- A typical programmable machine can be represented with three components : MPU,Memory and I/O as shown in Figure

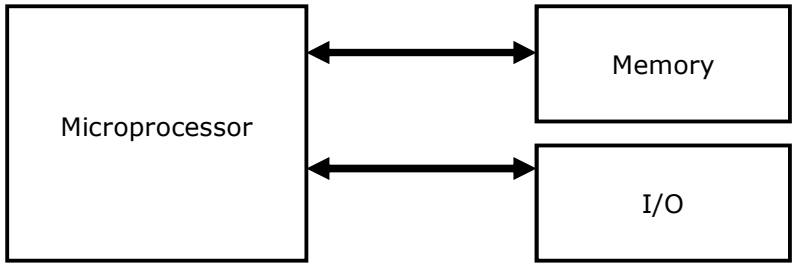


Figure: A Programmable Machine

- These three components work together or interact with each other to perform a given task; thus they comprise a system

- The machine (system) represented in above figure can be programmed to turn traffic lights on and off, compute mathematical functions, or keep trace of guidance system.

- This system may be simple or sophisticated, depending on its applications.

- The MPU applications are classified primarily in two categories : reprogrammable systems and embedded systems

- In reprogrammable systems, such as Microcomputers, the MPU is used for computing and data processing.

- In embedded systems, the microprocessor is a part of a final product and is not available for reprogramming to end user.

MICROCOMPUTER

- As the name implies, Microcomputers are small computers

- They range from small controllers that work directly with 4-bit words to larger units that work directly with 32-bit words

- Some of the more powerful Microcomputers have all or most of the features of earlier minicomputers.

- Examples of Microcomputers are Intel 8051 controller-a single board computer, IBM PC and Apple Macintosh computer.

©Achyuta Nand Mishra

MICRO CONTROLLER

- Single-chip Microcomputers are also known as Microcontrollers.
- They are used primarily to perform dedicated functions.
- They are used primarily to perform dedicated functions or as slaves in distributed processing.
- Generally they include all the essential elements of a computer on a single chip: MPU,R/W memory, ROM and I/O lines.
- Typical examples of the single-chip microcomputers are the Intel 8051, AT89C51, AT89C52 and Zilog Z8.
- Most of the micro controllers have an 8-bit word size, at least 64 bytes of R/W memory, and 1K byte of ROM
- I/O lines varies from 16 to 40

Typical Example: AT89C51 Microcontroller

- It is low power, high performance CMOS 8 bit microcomputer with 4K bytes of Flash programmable and erasable Read Only Memory.
- 128 bytes of Internal RAM
- 32 I/O pins arranged as 4 ports (PO-P3)
- A full duplex serial port
- 6 Hardware Interrupts
- 16 bit PC and Data Pointers.
- 8 bit Program Status Word
- Two 16 bits timers/counter TO and T1

©Achyuta Nand Mishra

GENERAL ARCHITECTURE OF MICROCOMPUTER SYSTEM

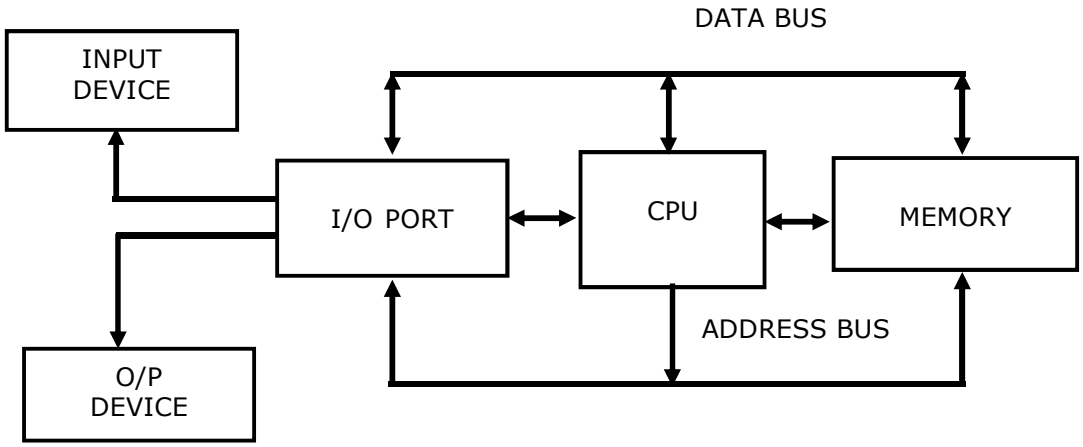


Figure: Block Diagram of a simple Microcomputer

- Figure shows a block diagram for a simple Microcomputer.

`

- The major parts are the CPU, Memory and I/O.
- Connecting these parts are three sets of parallel lines called buses.
- The three buses are address bus, data bus and the control bus.

MEMORY

- It consists of RAM and ROM.
- The First Purpose of memory is to store binary codes for the sequences of instructions you want the computer to carry out.
- The second purpose of the memory is to store the binary-coded data with which the computer is going to be working

INPUT/OUTPUT

- The input/output or I/O Section allows the computer to take in data from the outside world or send data to the outside world.
- Peripherals such as keyboards, video display terminals, printers are connected to I/O Port.

CPU

- The CPU controls the operation of the computer.
- In a microcomputer CPU is a microprocessor.
- The fetches binary coded instructions from memory, decodes the instructions into a series of simple actions and carries out these actions in a sequence of steps.
- The CPU also contains an address counter or instruction pointer register, which holds the address of the next instruction or data item to be fetched from memory.

©Achyuta Nand Mishra

ADDRESS BUS

- The address bus consists of 16, 20, 24 or 32 parallel signal lines.
- On these lines the CPU sends out the address of the memory location that is to be written to or read from.
- The no of memory location that the CPU can address is determined by the number of address lines.
- If the CPU has N address lines, then it can directly address 2^N memory locations i.e. CPU with 16 address lines can address 2^{16} or 65536 memory locations.

DATA BUS

- The data bus consists of 8, 16 or 32 parallel signal lines.
- The data bus lines are bi-directional.
- This means that the CPU can read data in from memory or it can send data out to memory

CONTROL BUS

- The control bus consists of 4 to 10 parallel signal lines.
- The CPU sends out signals on the control bus to enable the output of addressed memory devices or port devices.
- Typical control bus signals are Memory Read, Memory Write, I/O Read and I/O Write.

©Achyuta Nand Mishra

COMPONENTS OF CPU

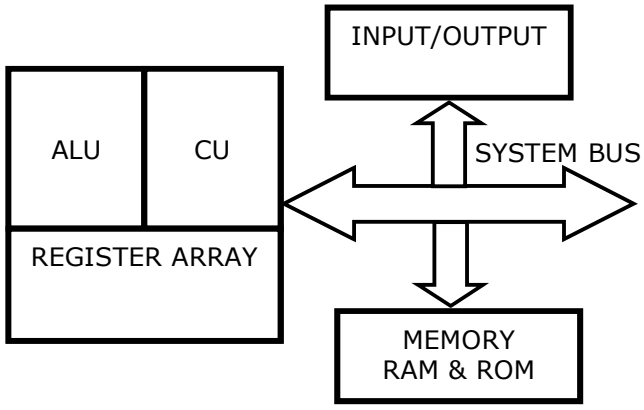


Figure: Microprocessor Based System with Bus Architecture.

- The Microprocessor is divided into three segments: ALU, Register array and Control Unit.

ARITHMETIC LOGIC UNIT

- This is the area of Microprocessor where various computing functions are performed on data.
- The ALU performs operations such as addition, subtraction and logic operations such as AND, OR and exclusive OR.

REGISTER ARRAY

- These are storage devices to store data temporarily.
- There are different types of registers depending upon the Microprocessors.
- These registers are primarily used to store data temporarily during the execution of a program and are accessible to the user through the instructions.
- General purpose Registers of 8086 includes
  AL, AH, BL, BH, CL, CH, DL, DH.

CONTROL UNIT

- The Control Unit Provides the necessary timing and control signals to all the operations in the Microcomputer
- It controls the flow of data between the Microprocessor and Memory and Peripherals.
- The Control unit performs 2 basic tasks
  - Sequencing
  - Execution

1. SEQUENCING

- The control unit causes the processor to step through a series of micro-operations in the proper sequence, based on the program being executed.

2. EXECUTION

- The control unit causes each micro operation to be performed.

©Achyuta Nand Mishra

CONTROL SIGNALS

- For the control unit to perform its function it must have inputs that allow it to determine the state of the system and outputs that allow it to control the behavior of the system.

- Inputs : Clock , Instruction Register, Flags

- Outputs :
    - Control signals to Memory
    - Control signals to I/O
    - Control Signals within the Processor.

©Achyuta Nand Mishra

EVOLUTION OF MICROPROCESSOR: INTEL SERIES

4004

- The first commercially available Microprocessor was the Intel 4004 produced in 1971.
- It contained 2300 PMOS transistors.
- The 4004 was a 4 bit device intended to be used with some other devices in making a calculator.
- In 1972 Intel came out with the 8008,which was capable of working with 8 bit words.

8008

- The 8008, however required 20 or more additional devices to form a functional CPU.

8080

- In 1974 Intel announced the 8080, which had a much larger instruction set than the 8008 and required only two additional devices to form a functional CPU.
- The 8080 used NMOS transistor, so it operated much faster than the 8008
- The 8080 is referred as a Second generation Microprocessor.
- It requires +5V,-5V and +12V supply.

8085

- In 1977, Intel Produced 8085, an upgrade of 8080 that required only a +5V supply
- It was a 8 bit Microprocessor.

8088

- Intel Produced 8088, which was the first Microprocessor used in Personal computer by IBM.
- It has 16 bit registers and an 8 bit data bus and can address up to 1 million bytes of internal memory.

8086

- In 1978 Intel came out with the 8086 which is a full 16 bit Microprocessor.
- It has a 16 bit data bus and runs faster.
- It can address 2^{20} or 1048576 memory locations.

80286

- Runs faster than the preceding processors, has additional capabilities and can address up to 16 million bytes.
- This processor can operate in real mode or in protected mode, which enables an operating system like windows to perform multitasking and to protect them from each other.

©Achyuta Nand Mishra

80386

- Has 32 bit registers and 32bit data bus.
- It can address up to 4 billion bytes of memory.
- The processor supports virtual mode, whereby it can swap portions of memory onto disk

80486

- Has 32 bit registers and 32 bit data bus.
- High speed cache memory connected to the processor bus enables the processor to store copies of the most recently used instructions and data.
- The processor can operate faster when using the cache directly without having to access the slower memory.

PENTIUM

- It has 32 bit registers, a 64 bit data bus and separate caches for data and for memory.
- The Pentium has a 5 Stage pipelined structure and the Pentium II has a 12 stage super pipelined structure.
- This feature enables them to run many operations in parallel.

MEMORY CALCULATIONS

- 2^{10} =1K(Kilo)
- 2^{20}=1M(Mega)
- 2^{30}=1G(Giga)
- 2^{40}=1T (Tera)
- Specify Data and Address Bus size and calculate the size of Memory.

APPLICATIONS OF MICROPROCESSOR

- Microcomputers
- Industrial Control
- Robotics
- Traffic Lights
- Washing Machines
- Microwave Oven
- Security Systems
- On Board Systems

EXAMPLE: A SYSTEM DESIGN WITH MPU

- SMART Fan
- Access Control System
- Automated Water Tank

©Achyuta Nand Mishra

CHAPTER –III

8086/8088 CPU ARCHITECTURE AND INSTRUCTION SET

©Achyuta Nand Mishra

8086 CPU ARCHITECURE AND INSTRUCTION SET

THE 8086 MICROPROCESSOR OVERVIEW

- The Intel 8086 is a 16 bit Microprocessor that is intended to be used as the CPU in a Microcomputer.
- The term 16 bit means that it's ALU, its internal registers, and most of its instructions are designed to work with 16 bit binary words.
- The 8086 has a 16 bit data bus, so it can read data from or write data to memory and ports either 16 bits or 8 bits at a time.
- The 8086 has a 20 bit address bus, so it can address 2^{20} or 1,048576 memory locations.
- Sixteen bit words will be stored in two consecutive memory locations.
- If the first byte of a word is at an even address, the 8086 can read the entire word in one operation.
- If the first byte of the word is at an odd address, the 8086 will read the first byte with one bus operation and the second byte with another bus operation
- The main point here is that if the first byte of a 16 bit word is at an even address, the 8086 can read the entire word in one operation

8086 AND 8088

- The Intel 8088 has the same arithmetic logic unit, the same registers and the same instruction set as the 8086.
- The 8088 also has a 20 bit address bus, so it can address any one of 1,048,576 bytes in memory.
- The 8088 has an 8 bit data bus, so it can only read data from or write data to memory and ports 8 bits at a time.
- The 8086 can read or write either 8 or 16 bits at a time.
- To read a 16 bit word from two successive memory locations, the 8088 will always have to do two read operations.
- The Intel 80186 is an improved version of 8086,and 80188 is an improved version of 8088
- In addition to 16 bit CPU, the 80186 and 80188 each have programmer peripheral devices integrated in the same package.

©Achyuta Nand Mishra

8086 INTERNAL ARCHITECTURE

INTERNAL ARCHITECTURE OF 8086

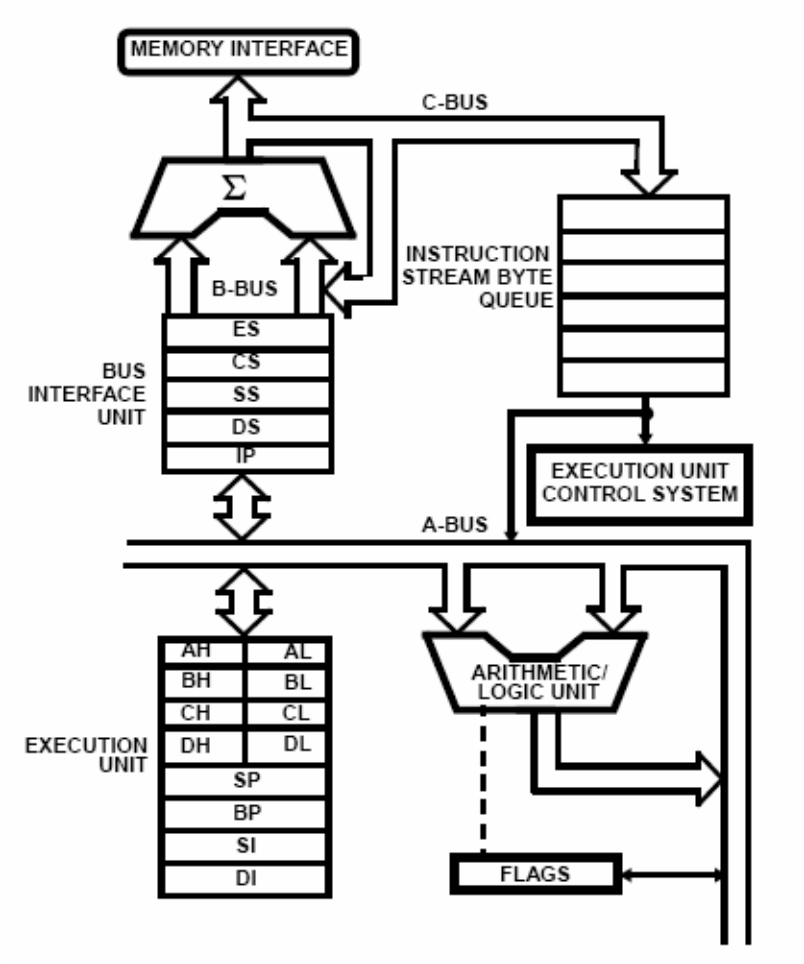


Figure: Internal Architecture of 8086

- The 8086 CPU is divided into two independent functional parts : BIU (Bus Interface Unit) and EU (Execution Unit)
- Dividing the work between these units speeds up the processing.
- The BIU send out address, fetches instructions from memory, reads data from ports and memory, and writes data to ports and memory.
- In other words BIU handles all transfers of data and addresses on the buses for the execution unit.
- The EU of the 8086 tells the BIU where to fetch the instructions and data from, decodes instructions and executes instructions.

©Achyuta Nand Mishra

A.THE EXECUTION UNIT

- The EU contains control circuitry which directs internal operations.
- A decoder in EU translates instructions fetched from memory into a series of actions which the EU carries out.
- The EU has a 16 bit ALU which can add subtract, ND, OR, increment, decrement, complement or shift binary numbers.

1. GENERAL PURPOSE REGISTERS

- The EU has eight general purpose registers, labeled AH, AL, BH, BL, CH, CL, DH and DL.
- These registers can be used individually for temporary storage of 8 bit data.
- The AL register is also called accumulator
- It has some features that the other general purpose registers do not have.
- Certain pairs of these general purpose registers can be used together to store 16 bit words.
- The acceptable register pairs are AH and AL,BH and BL,CH and CL,DH and DL
- The AH-AL pair is referred to as the AX register, the BH-BL pair is referred to as the BX register, the CH-CL pair is referred to as the CX register, and the DH-DL pair is referred to as the DX register.

  AX = Accumulator Register

  BX = Base Register

  CX = Count Register

  DX = Data Register

2. FLAG REGISTER

- A Flag is a flip-flop which indicates some condition produced by the execution of an instruction or controls certain operations of the EU.
- A 16 bit flag register in the EU contains 9 active flags.
- Figure below show shows the location of the nine flags in the flag register.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
U	**U**	**U**	**U**	**OF**	**DF**	**IF**	**TF**	**SF**	**ZF**	**U**	**AF**	**U**	**PF**	**U**	**CF**

Figure: 8086 Flag Register Format

U = UNDEFINED

©Achyuta Nand Mishra

CONDITIONAL FLAGS
CF = CARRY FLAG [Set by Carry out of MSB]
PF = PARITY FLAG [Set if Result has even parity]
AF= AUXILIARY CARRY FLAG FOR BCD
ZF = ZERO FLAG [Set if Result is 0]
SF = SIGN FLAG [MSB of Result]

CONTROL FLAG
TF = SINGLE STEP TRAP FLAG
IF = INTERRUPT ENABLE FLAG
DF = STRING DIRECTION FLAG
OF = OVERFLOW FLAG

- The six conditional flags in this group are the CF,PF,AF,ZF,SF and OF
- The three remaining flags in the Flag Register are used to control certain operations of the processor.
- The six conditional flags are set or reset by the EU on the basis of the result of some arithmetic or logic operation.
- The Control Flags are deliberately set or reset with specific instructions you put in your program.
- The three control flags are the TF,IF and DF.
- Trap Flag is used for single stepping through a program.
- The Interrupt Flag is used to allow or prohibit the interruption of a program.
- The Direction Flag is used with string instructions.

3. POINTER REGISTERS
- The 16 bit Pointer Registers are IP,SP and BP respectively
- SP and BP are located in EU whereas IP is located in BIU

3.1 STACK POINTER (SP)
- The 16 bit SP Register provides an offset value, which when associated with the SS register (SS:SP)

3.2 BASE POINTER (BP)
- The 16 bit BP facilitates referencing parameters, which are data and addresses that a program passes via the stack.
- The processor combines the addresses in SS with the offset in BP.
- BP can also be combined with DI and SI as a base register for special addressing.

4. INDEX REGISTERS
- The 16 bit Index Registers are SI and DI

4.1 SOURCE INDEX (SI) REGISTER
- The 16 bit Source Index Register is required for some string handling operations
- SI is associated with the DS Register.

©Achyuta Nand Mishra

4.2 DESTINATION INDEX (DI) REGISTER

- The 16 bit Destination Index Register is also required for some string operations.
- In this context, DI is associated with the ES register.

B. THE BUS INTERFACE UNIT

1. SEGMENT REGISTERS

1.1 CS REGISTER

- It contains the starting address of a program's code segment.
- This segment address plus an offset value in the IP register indicates the address of an instruction to be fetched for execution
- For normal programming purpose, you need not directly reference this register.

1.2 DS REGISTER

- It contains the starting address of a program's data segment
- Instruction uses this address to locate data.
- This address plus an offset value in an instruction causes a reference to a specific byte location in the data segment.

1.3 SS REGISTER

- Permits the implementation of a stack in memory
- It stores the starting address of a program's stack segment the SS register.
- This segment address plus an offset value in the Stack Pointer (SP) register indicates the current word in the stack being addressed.

1.4 ES REGISTER

- It is used by some string operations to handle memory addressing.
- ES Register is associated with the DI Register.

2. INSTRUCTION POINTER (IP)

- The 16 bit IP Register contains the offset address of the next instruction that is to execute.
- IP is associated with CS register as (CS:IP)
- For each instruction that executes, the processor changes the offset value in IP so that IP in effect directs each step of execution.

3. THE QUEUE

- While the EU is decoding an instruction or executing an instruction which does not require use of the buses, the BIU fetches up to six instructions bytes for the following instructions.
- The BIU Stores prefetched bytes in First in First out register set called a queue.
- When the EU is ready for its next instruction, it simply reads the instruction bytes for the instruction from the queue in the BIU.
- This is much faster than sending out an address to the system memory and waiting for memory to send back the next instruction bytes or bytes.
- Fetching the next instruction while the current instruction executes is called pipeling.

©Achyuta Nand Mishra

8086 MEMORY ORGANIZATION

1. INTRODUCTION

- The Intel 8086 is a 16 bit Microprocessor that is intended to be used as the CPU in a Microcomputer.
- The 8086 has a 20 bit address bus so it can address any one of 2^{20} or 1,048,576 memory locations.
- Each of the 1,048,576 memory address of the 8086 represents a byte-wide location.
- 16 bit word will be stored in two consecutive memory locations.
- If the first byte of a word is at an even address, the 8086 can read the entire word in one operation.
- If the first byte of a word is at an odd address, the 8086 will read the first byte with one bus cycle and the second byte with another bus cycle.

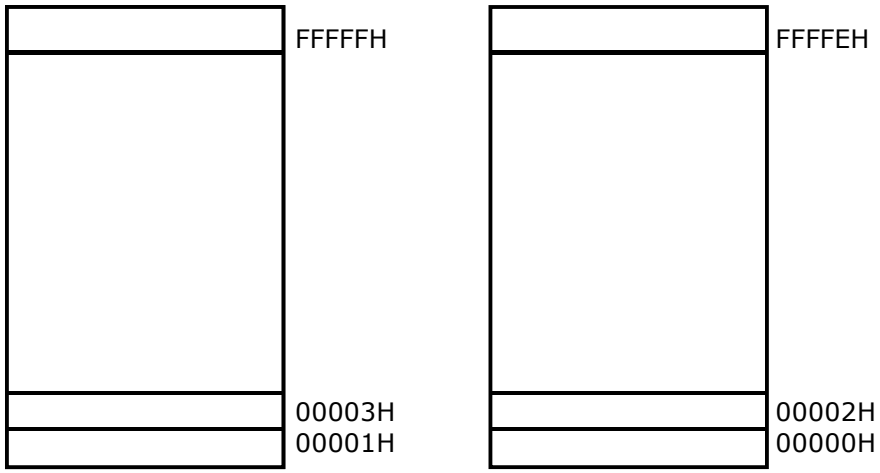


Figure: The Physical Memory System of 8086

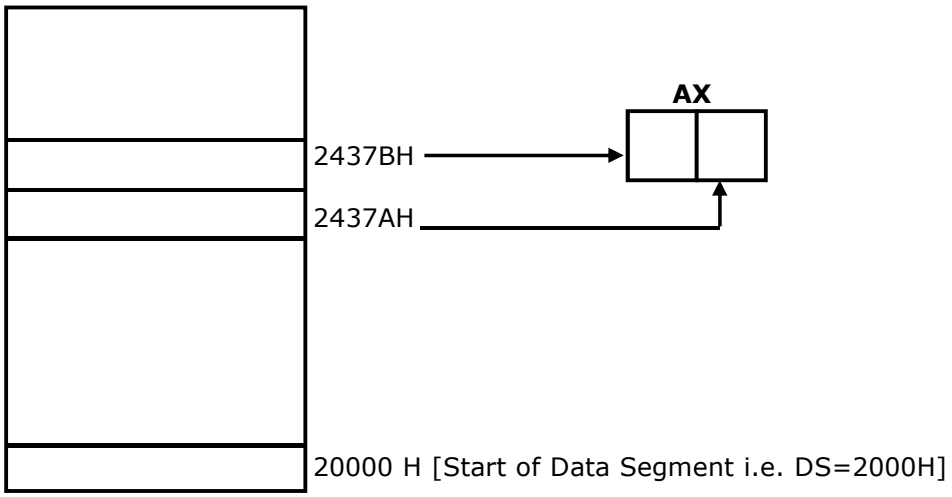
©Achyuta Nand Mishra

2. ACCESSING DATA IN MEMORY

- An important point here is that an 8086 always stores the low byte of word in lower address and stores high byte of word in higher address.
- Low Byte – Low Address : High Byte – High Address
- MOV AX,[437AH]

  Assume DS=2000H
- To Compute the physical address Add 2000**0** H and 437AH

  2000**0** H **+** 437A H = 2437AH
- 2437A H is the physical address.



3. SEGMENTED MEMORY

- The 8086 BIU sends out 20 bit address so it can address any of 2^{20} or 1,048,576 bytes in memory.
- However at any given time the 8086 works with only four 65536 bytes (64 Kbyte) segment within this 1,048,576 byte (1 Mbyte) Range
- Four segments are : Code Segment, Stack Segment, Data Segment and Extra Segment
- Four segment registers in BIU are used to hold the upper 16 bits of the starting address of 4 memory segments that the 8086 is working with at a particular time.
- The 4 segment registers are code segment register (CS), stack segment register (SS), data segment register (DS) and the extra segment register (ES).
- For small programs which do not need all 64 Kbytes in each segment can overlap.

©Achyuta Nand Mishra

- For example, the code segment holds the upper 16 bits of the starting address for the segment from which the BIU is currently fetching instruction code bytes.
- The BIU always inserts zero for the lowest 4 bits of the 20 bit starting address.
- If the code segment register contains 348A H then the code segment will start at address 348A**0** H
- A 64 Kbytes segment can be located anywhere within the 1 Mbyte address space, but the segment will always start at an address with zeros in the lowest 4 bits.

ADVANTAGES OF SEGMENTATION [SEGMENT: OFFSET SCHEME]

Intel designed the 8086 family devices to access memory using the segment: offset approach rather than accessing memory directly with 20 bit. The advantages are listed below.

- The segment: offset scheme requires only a 16 bit number to represent the base address for a segment and only a 16 bit offset to access any location in a segment. This means that 8086 has to manipulate and store only 16 bit quantities instead of 20 bit quantities.
- This makes easier interface with 8 and 16 bit wide memory boards and with 16 bit registers in the 8086.
- It allows programs to be relocated in memory system. A relocatable program is one that can be placed in any area of memory and executed without change.
- It allows programs written to function in the real mode to operate in protected mode.
- Segmentation also makes easy to keep user's program and data separate from one another and segmentation makes it easy to switch from one user's program to another user's program.

DISADVANTAGE OF SEGMENT: OFFSET APPROACH

- The segment: offset scheme introduces complexity in hardware and software design.

DIFFERENT SEGMENT OFFSET COMBINATION

SEGMENT	OFFSET	SPECIAL PURPOSE
CS [CODE SEGMENT]	IP [INSTRUCTION POINTER]	INSTRUCTION ADDRESS
SS [STACK SEGMENT]	SP [STACK POINTER] BP [BASE POINTER]	STACK ADDRESS
DS [DATA SEGMENT]	BX [BASE REGISTER] DI [DESTINATION INDEX] SI [SOURCE INDEX] 8 BIT NUMBER 16 BIT NUMBER	DATA ADDRESS
ES [EXTRA SEGMENT]	DI [DESTINATION INDEX]	STRING DESTINATION ADDRESS

©Achyuta Nand Mishra

INTERRUPT FUNCTION

Dos Service INT 21H

This service has many function but few of them are given below. The function no. of the service is to be loaded in the register AH and the other registers are loaded with the data as required before the interrupt call.

Function No.	Description
00H	Terminate the current program: INT 21H, function 4CH is used instead
01H	Console input with echo: wait for a character from the standard input device. The character is returned in AL and echoed. Respond to CTRL+BREAK
02H	Character output: Send the character in DL to parallel output device
05H	Printer output: Send the character in DL to parallel printer port.
06H	Direct console input-output: Reads the character in AL if DL=0FFh else display character at DL to the standard output device. Control character are not filtered.
07H	Console input: wait for a character from standard input device. The character is returned in AL. but not echoed. Does not respond to CTRL + BREAK
08H	Console input without echo: Wait for a character from standard input device. The character is returned in Al, but not echoed. Respond CTRL+BREAK
09H	String output: Send a string of characters to the standard output device until $ character is reached. DX contains the offset address of the string.
0AH	Read string: Read characters from the standard input device. DX points a location whose first byte gives the max characters allowed to enter, the next byte reserved to store the actual no of characters entered and the rest space to store entered characters.
0BH	Check Keyboard status: Return FFH and AL if an input character is available in the keyboard buffer else returns 00H in Al.
0CH	Clear key board buffer and invoke input functions: The input functions are stored in AL and other register should hold the values as required.

©Achyuta Nand Mishra

ADDRESSING MODES

- The different ways in which a processor can access data are referred to as its addressing modes.
- In assembly language statements, the addressing mode is indicated in the instruction itself.
- The various addressing modes are

1. Register Addressing Mode
2. Immediate Addressing Mode
3. Direct Addressing Mode
4. Register Indirect Addressing Mode
5. Base plus Index Addressing Mode
6. Register Relative Addressing Mode
7. Base Relative Plus Index Addressing Mode

1. REGISTER ADDRESSING MODE

- It is the most common form of data addressing.
- Transfers a copy of a byte/word from source register to destination register.

INSTRUCTION	SOURCE	DESTINATION
MOV AX,BX	REGISTER BX	REGISTER AX

- It is carried out with 8 bit registers AH,AL,BH,BL,CH,CL,DH & DL or with 16 bit registers AX,BX,CX,DX,SP,BP,SI and DI.
- It is important to use registers of same size.
- Never mix an 8 bit register with a 16 bit register i.e. MOV AX,BL

EXAMPLES

```
MOV AL,BL        : Copys BL into AL
MOV ES,DS        : Copys DS into ES
MOV AX,CX        : Copys CX into AX
```

2. IMMEDIATE ADDRESSING MODE

- The term immediate implies that the data immediately follow the hexadecimal opcode in the memory.
- Note that immediate data are constant data.
- It transfers the source immediate byte/word of data in destination register or memory location.

INSTRUCTION	SOURCE	DESTINATION
MOV CH,3AH	DATA 3AH	REGISTER AX

EXAMPLES

```
MOV AL,90          : Copys 90 into AL
MOV AX,1234H       : Copys 1234H into AX
MOV CL,10000001B   : Copys 100000001 binary value into CL
```


©Achyuta Nand Mishra

3. DIRECT ADDRESSING MODE

- In this scheme, the address of the data is defined in the instruction itself.
- When a memory location is to be referenced, its offset address must be specified

INSTRUCTION	SOURCE	DESTINATION
MOV AL,[1234H]	ASSUME DS=1000H 1000**0** H + 1234 H 11234H MEMORY LOCATION 11234 H	REGISTER AL

EXAMPLES

MOV AL,[1234H] : Copys the byte content of data segment memory location 11234H into AL.

MOV AL, NUMBER : Copys the byte content of data segment memory location NUMBER into AL.

4. REGISTER INDIRECT ADDRESSING MODE

- Register Indirect Addressing allows data to be addressed at any memory location through an offset address held in any of the following registers: BP, BX, DI and SI.
- The Index and Base registers are used to specify the address of data.
- It transfers byte/word between a register and a memory location addressed by an index or base registers.
- The symbol [ ] denote indirect addressing.
- The data segment is used by default with register indirect addressing or any other addressing mode that uses BX,DI or SI to address memory. If BP register addresses memory, the stack segment is used by default.

INSTRUCTION	SOURCE	DESTINATION
MOV CL,[BX]	ASSUME DS=1000H ASSUME BX=0300H 1000**0** H + 0300 H 10300H MEMORY LOCATION 10300H	REGISTER CL

EXAMPLES

MOV CX,[BX] : Copys the word contents of the data segment memory location addressed by BX into CX.

MOV [DI],BH : Copys BH into the data segment memory location addressed by DI.

MOV [DI],[BX] : Memory to Memory moves are not allowed except with string instructions.

©Achyuta Nand Mishra

5. BASE PLUS INDEX ADDRESSING MODE

- Base plus index addressing is similar to indirect addressing because it indirectly addresses memory data
- This type of addressing uses one base register (BP or BX) and one Index Register (DI or SI) to indirectly address memory.

INSTRUCTION	SOURCE	DESTINATION
MOV [BX+SI],CL	REGISTER CL	ASSUME DS=1000H ASSUME BX=0300H ASSUME SI =0200H 1000**0**H + 0300H + 0200H 10500H MEMORY LOCATION 10500H

EXAMPLES

MOV CX,[BX+DI] : Copys the word contents of the data segment memory location addressed by BX plus DI into CX.

MOV CH,[BP+SI] : Copys the byte contents of the stack segment memory location addressed by BP plus SI into CH.

6. REGISTER RELATIVE ADDRESSING MODE

- In this case, the data in a segment of memory are addressed by adding the displacement to the content of base or an index register (BP,BX ,DI or SI).
- Transfers a byte/word between a register and the memory location addressed by an index or base register plus a displacement.

INSTRUCTION	SOURCE	DESTINATION
MOV [BX+4],CL	REGISTER CL	ASSUME DS=1000H ASSUME BX=0300H 1000**0**H + 0300H + 4H 10304H MEMORY LOCATION 10304H

EXAMPLES

MOV ARRAY[SI],BL : Copys BL into the data segment memory location addressed by ARRAY plus SI.

MOV LIST[SI+2],CL : Copys CL into the data segment memory location addressed by sum of LIST, SI and 2.

©Achyuta Nand Mishra

7. BASE RELATIVE PLUS INDEX ADDRESSING MODE

- The base relative plus index addressing mode is similar to the base plus index addressing mode but it adds a displacement to form a memory address.

- Transfers a byte or word between a register and the memory location addressed by a base and an index register plus a displacement.

INSTRUCTION	SOURCE	DESTINATION
MOV [BX+SI+05],CL	REGISTER CL	ASSUME DS=1000H ASSUME BX=0300H ASSUME SI =0200H 1000**0**H + 0300H + 0200H +05H 10505H MEMORY LOCATION 10505H

EXAMPLES

MOV LIST[BP+DI],CL : Copys CL into the stack segment memory location addressed by the sum of LIST, BP and DI

MOV DH,[BX+DI+20H] : Copys the byte contents of the data segment memory location addressed by the sum of BX,DI and 20H into DH

©Achyuta Nand Mishra

INSTRUCTION SETS

1. DATA TRANSFER INSTRUCTIONS

1.1 GENERAL PURPOSE BYTE OR WORD TRANSFER INSTRUCTIONS

INSTRUCTIONS	COMMENTS

MOV
 	
MOV Destination,Source
MOV CX,04H
	Copy byte or word from specified source to specified destination.

PUSH PUSH Source PUSH BX	Copy specified word to top of stack.
POP POP Destination POP AX	Copy word from top to stack to specified location.
XCHG XCHG Destination,Source XCHG AX,BX	Exchange word or byte.
XLAT	Translate a byte in AL using a table in memory. It first adds AL + BX to form memory address. It then copys the content into AL

1.2 SIMPLE INPUT AND OUTPUT PORT TRANSFER INSTRUCTIONS

INSTRUCTIONS	COMMENTS
IN IN AX,Port_Addr IN AX,34H	Copy a byte or word from specified port to accumulator.
OUT OUT Port_Addr,AX OUT 2CH,AX	Copy a byte or word from accumulator to specified port.

1.3 SPECIAL ADDRESS TRANSFER INSTRUCTIONS

INSTRUCTIONS	COMMENTS
LEA LEA Register,Source LEA BX,PRICE	Load effective address of operand into specified register.

LDS
 	
LDS Register,Source
LDS BX,[4326H]
	Load DS register and other specified register from memory.

LES	Load ES register and other specified register from memory.

©Achyuta Nand Mishra

1.4 FLAG TRANSFER INSTRUCTIONS

INSTRUCTIONS	COMMENTS
LAHF	Copy to AH with the low byte of the flag register.
SAHF	Stores AH register to low byte of flag register.
PUSHF	Copy flag register to top of stack.
POPF	Copy word at top of stack to flag register.

©Achyuta Nand Mishra

2. ARITHMETIC INSTRUCTIONS

2.1 ADDITION INSTRUCTIONS

INSTRUCTIONS	COMMENTS

ADD
 	
ADD Destination,Source
ADD AL,74H
	Add specified byte to byte or word to word.

ADC ADC Destination,Source ADC CL,BL	Add byte + byte + carry flag Add word+word + carry flag
INC INC Register INC CX	Increment specified byte or word by 1.
AAA	ASCII adjust after addition.
DAA	Decimal adjust after addition.

2.2 SUBTRACTION INSTRUCTIONS

INSTRUCTIONS	COMMENTS

SUB
 	
SUB Destination,Source
SUB CX,BX
	Subtract byte from byte or word from word.

SBB SBB Destination,Source SBB CH,AL	Subtract byte and carry flag from byte. Subtract word and carry flag from word.
DEC DEC Register DEC CX	Decrement specified byte or word by 1.
NEG NEG Register NEG AL	Form 2's complement.

CMP
 	
CMP Destination,Source
CMP CX,BX
 	
CF ZF SF
CX = BX 0 1 0
CX > BX 0 0 0
CX < BX 1 0 1
	Compare two specified bytes or words.

AAS	ASCII adjust after subtraction.
DAS	Decimal adjust after subtraction.

©Achyuta Nand Mishra

2.3 MULTIPLICATION INSTRUCTIONS

INSTRUCTIONS	COMMENTS
MUL MUL Source MUL CX	Multiply unsigned byte by byte or unsigned word by word. When a byte is multiplied by the content of AL, the result is kept into AX. When a word is multiplied by the content of AX, MS Byte in DX and LS Byte in AX.
IMUL IMUL Source IMUL CX	Multiply signed byte by byte or signed word by word.
AAM	ASCII adjust after multiplication. It converts packed BCD to unpacked BCD.

2.4 DIVISION INSTRUCTIONS

INSTRUCTIONS	COMMENTS
DIV DIV Source DIV BL DIV CX	Divide unsigned word by byte Divide unsigned word double word by byte. When a word is divided by byte, the word must be in AX register and the divisor can be in a register or a memory location. After division AL (quotient) AH (remainder) When a double word is divided by word, the double word must be in DX:AX pair and the divisor can be in a register or a memory location. After division AX (quotient) DX (remainder)
AAD	ASCII adjust before division BCD to binary convert before division.
CBW	Fill upper byte of word with copies of sign bit of lower byte.
CWD	Fill upper word of double word with sign bit of lower word.

©Achyuta Nand Mishra

3. BIT MANIPULATION INSTRUCTIONS

3.1 LOGICAL INSTRUCTIONS

INSTRUCTIONS	COMMENTS
NOT NOT Destination NOT BX	Invert each bit of a byte or word.
AND AND Destination,Source AND BH,CL	AND each bit in a byte/word with the corresponding bit in another byte or word.
OR OR Destination,Source OR AH,CL	OR each bit in a byte or word with the corresponding bit in another byte or word.
XOR XOR Destination,Source XOR CL,BH	XOR each bit in a byte or word with the corresponding bit in another byte or word.
TEST TEST Destination,Source TEST AL,BH	AND operands to update flags, but don't change the operands.

3.2 SHIFT INSTRUCTIONS

INSTRUCTIONS	COMMENTS
SHL/SAL SAL Destination,Count SHL Destination,Count CF←MSB←LSB←0	Shift Bits of Word or Byte Left, Put Zero(s) in LSB.
SHR SHR Destination,Count 0→MSB→LSB→CF	Shift Bits of Word or Byte Right, Put Zero(s) in MSB.
SAR SAR Destination,Count MSB→MSB→LSB→CF	Shift Bits of Word or Byte Right, Copy Old MSB into New MSB.

3.3 ROTATE INSTRUCTIONS

INSTRUCTIONS	COMMENTS
ROL	Rotate Bits of Byte or Word Left,MSB to LS and to CF.
ROR	Rotate Bits of Byte or Word Right,LSB to MSB and to CF.
RCL	Rotate Bits of Byte or Word Left,MSB to CF and CF to LSB.
RCR	Rotate Bits of Byte or Word Right,LSB TO CF and CF TO MSB.

©Achyuta Nand Mishra

4. PROGRAM EXECUTION TRANSFER INSTRUCTIONS

4.1 UNCONDITIONAL TRANSFER INSTRUCTION

INSTRUCTIONS	COMMENTS
CALL	Call a Subprogram/Procedure.
RET	Return From Procedure to Calling Program.
JMP	Goto Specified Address to Get Next Instruction (Unconditional Jump to Specified Destination).

4.2 CONDITIONAL TRANSFER INSTRUCTION

INSTRUCTIONS	COMMENTS
JA/JNBE	Jump if Above/Jump if Not Below or Equal.
JAE/JNB	Jump if Above or Equal/Jump if Not Below.
JB/JNAE	Jump if Below/Jump if Not Above or Equal.
JBE/JNA	Jump if Below or Equal/Jump if Not Above.
JC	Jump if Carry Flag CF=1.
JE/JZ	Jump if Equal/Jump if Zero Flag (ZF=1).
JG/JNLE	Jump if Greater/Jump if Not Less than or Equal.
JGE/JNL	Jump if Greater than or Equal/Jump if Not Less than.
JL/JNGE	Jump if Less than/Jump if Not Greater than or Equal.
JLE/JNG	Jump if Less than or Equal/Jump if Not Greater than.
JNC	Jump if No Carry i.e. CF=0
JNE/JNZ	Jump if Not Equal/Jump if Not Zero(ZF=0)
JNO	Jump if No Overflow.
JNP/JPO	Jump if Not Parity/Jump if Parity Odd.
JNS	Jump if Not Sign(SF=0)
JP/JPE	Jump if Parity/Jump if Parity Even (PF=1)
JS	Jump if Sign (SF=1)

©Achyuta Nand Mishra

4.3 ITERATION CONTROL INSTRUCTIONS

INSTRUCTIONS	COMMENTS
LOOP	Loop Through a Sequence of Instructions Until CX=0.
LOOPE/LOOPZ	Loop Through a Sequence of Instructions While ZF=1 and CX!=0.
LOOPNE/LOOPNZ	Loop Through a Sequence of Instruction While ZF=0 & CX!=0.
JCXZ	Jump to Specified Address if CX=0.

4.4 INTERRUPT INSTRUCTIONS

INSTRUCTIONS	COMMENTS
INT	
INT0	Interrupt Program Execution if OF=1
IRET	Return From Interrupt Service Procedure to Main Program.

5 PROCESSOR CONTROL INSTRUCTIONS

5.1 FLAG SET/CLEAR INSTRUCTION

INSTRUCTIONS	COMMENTS
STC	Set Carry Flag CF to 1.
CLC	Clear Carry Flag to 0.
CMC	Complement the State of CF.
STD	Set Direction Flag to 1.
CLD	Clear Direction Flag to 0.
STI	Set Interrupt Flag to 1. (Enable INTR Input).
CLI	Clear Interrupt Enable to 0

5.2 NO OPERATION INSTRUCTION

INSTRUCTIONS	COMMENTS
NOP	No Action Except Fetch and Decode.

©Achyuta Nand Mishra

5.3 EXTERNAL HARDWARE SYNCHRONIZATION INST.

INSTRUCTIONS	COMMENTS
HLT	Halt (Do Nothing) Until Interrupt or Reset.
WAIT	Wait Until Signal On the TEST Pin is Low.
ESC	Escape to External Coprocessor Such as 8087 or 8089.
LOCK	Prevents Another Processor From Taking the Bus While the Adjacent Instruction Executes.

6 STRING INSTRUCTIONS

INSTRUCTIONS	COMMENTS
REP	Repeat Instruction Until CX=0.
REPE/REPZ	Repeat if Equal/Repeat if Zero
REPNE/REPNZ	Repeat if Not Equal/Repeat if Not Zero.
MOVS/MOVSB/MOVSW	Move Byte or Word From One String to Another.
COMPS/COMPSB/COMPSW	Compare Two String Bytes or Two String Words.
SCAS/SCASB/SCASW	Compares a Byte in AL or Word in AX With a Byte or Word Pointed By DI in ES.

©Achyuta Nand Mishra

ASSEMBLY LANGUAGE PROGRAMMING

INTRODUCTION

- Assembly Language uses two, three or 4 letter mnemonics to represent each instruction type.
- Low level Assembly Language is designed for a specific family of Processors : the symbolic instruction directly relate to Machine Language instructions one for one and are assembled into machine language
- To make programming easier, many programmers write programs in assembly language
- They then translate Assembly Language program to machine language so that it can be loaded into memory and run.

ADVANTAGES OF ASSEMBLY LANGUAGE

- A Program written in Assembly Language requires considerably less Memory and execution time than that of High Level Language.
- Assembly Language gives a programmer the ability to perform highly technical tasks.
- Resident Programs (that resides in memory while other programs execute) and Interrupt Service Routine (that handles I/P and O/P) are almost always developed in Assembly Language.
- Provides more control over handling particular H/W requirements.
- Generates smaller and compact executable modules.
- Results in faster execution.

TYPICAL FORMAT OF AN ASSEMBLY LANGUAGE INSTRUCTION

LABEL	OPCODE FIELD	OPERAND FIELD	COMMENTS
NEXT:	ADD	AL,07H	; Add correction factor

- Assembly language statements are usually written in a standard form that has 4 fields.
- A label is a symbol used to represent an address. They are followed by colon
- Labels are only inserted when they are needed so it is an optional field.
- The opcode field of the instruction contains the mnemonics for the instruction to be performed
- The instruction mnemonics are sometimes called as operation codes.
- The operand field of the statement contains the data, the memory address, the port address or the name of the register on which the instruction is to be performed.
- The final field in an assembly language statement is the comment field which starts with semicolon. It forms a well documented program.

©Achyuta Nand Mishra

ASSEMBLY LANGUAGE PROGRAM DEVELOPMENT TOOLS

1. EDITOR

- An Editor is a Program which allows you to create a file containing the Assembly Language statements for your Program.

2. ASSEMBLER

- An Assembler Program is used to translate the assembly language mnemonics for instruction to the corresponding binary codes.

3. LINKER

- A Linker is a Program used to join several files into one large .obj file. It produces .exe file so that the program becomes executable.

4. LOCATOR

- A Locator is a program used to assign the specific address of where the segment of object code are to be loaded into memory.
- It usually converts .exe file to .bin file.
- A Locator program EXE2BIN converts .exe file to .bin file.

5. DEBUGGER

- A Debugger is a program which allows you to load your .obj code program into system memory, execute program and troubleshoot.
- It allows you to look at the content of registers and memory locations after your program runs.
- It allows to set the breakpoint.

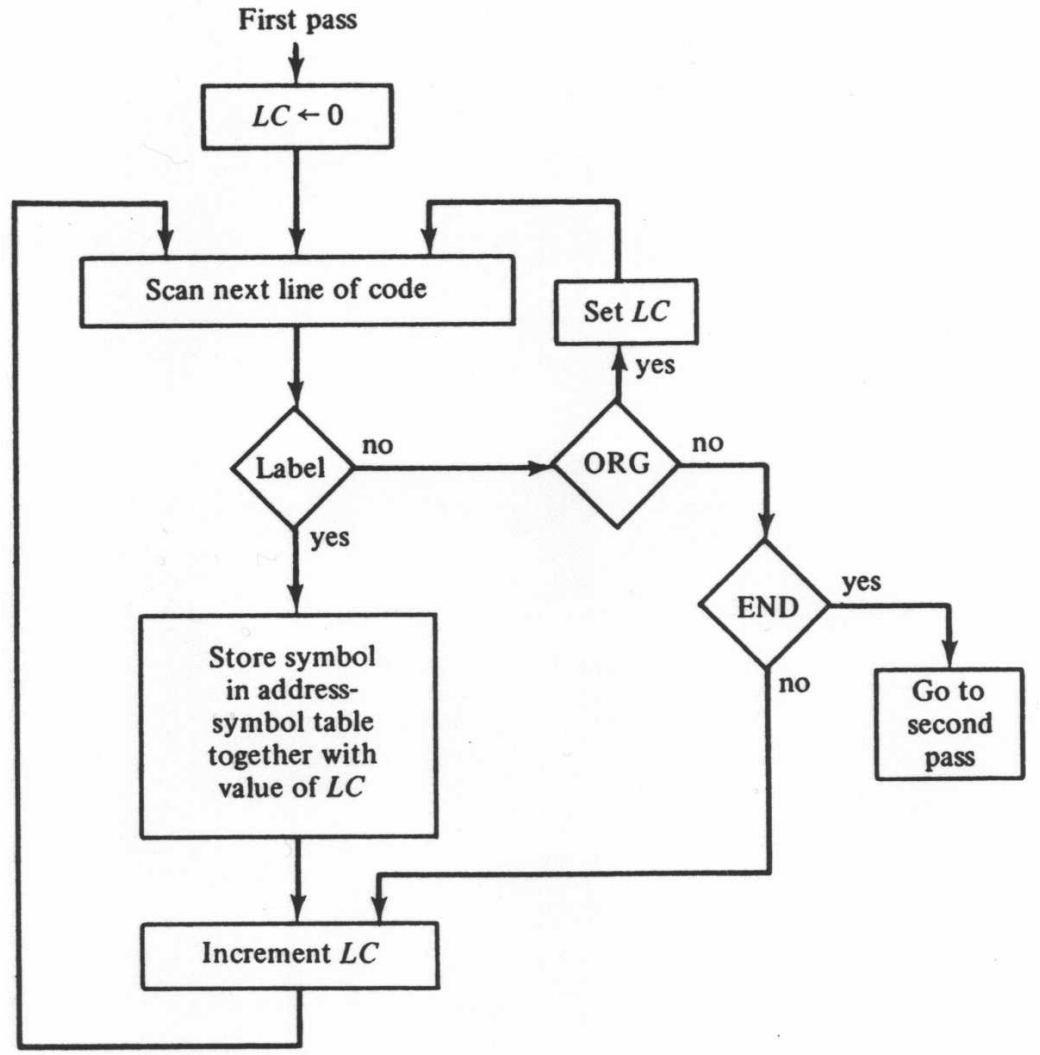
6. EMULATOR

- An Emulator is a mixture of hardware and software.
- It is used to test and debug the hardware and software of an external system such as the prototype of a Microprocessor based system.

©Achyuta Nand Mishra

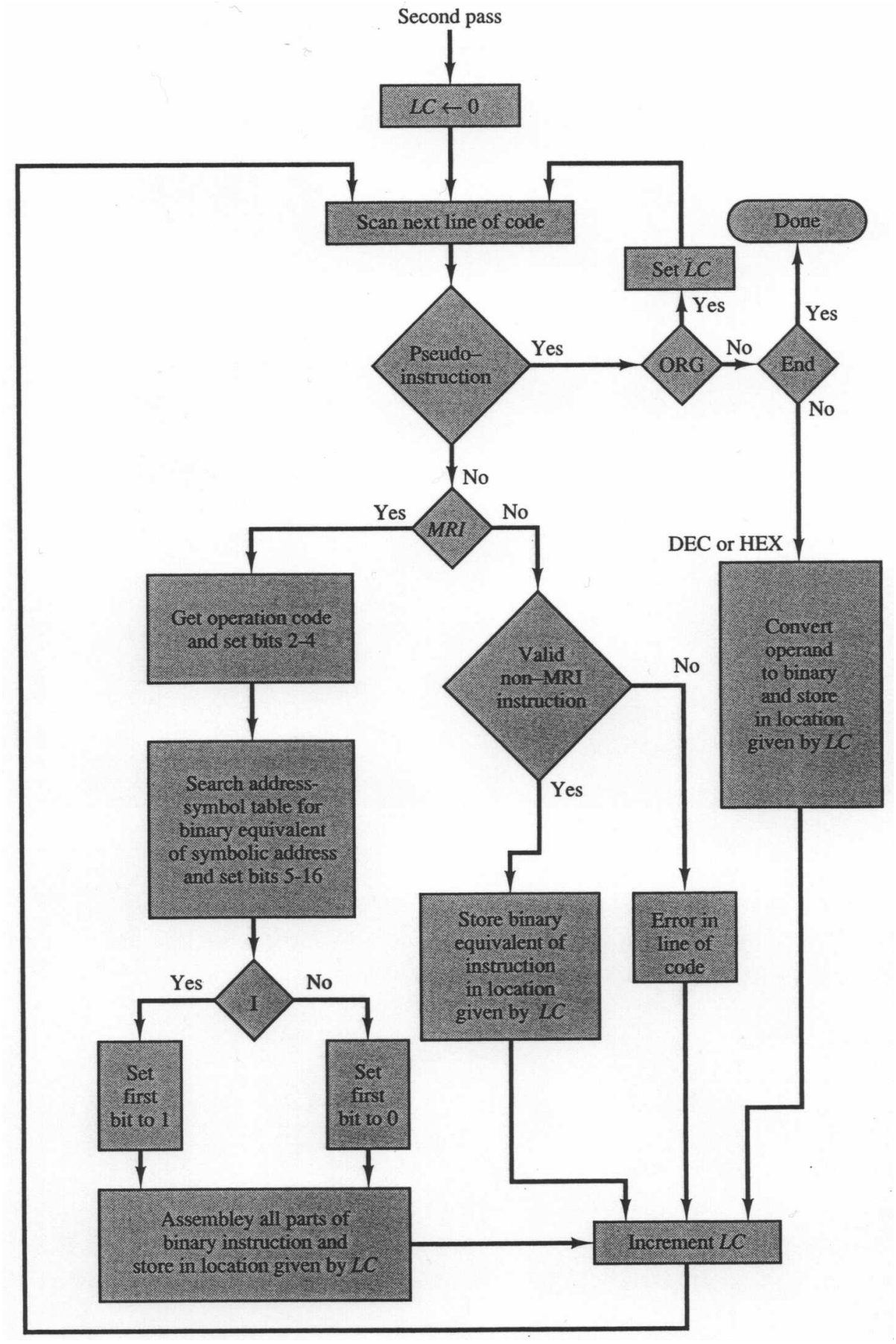
TWO Pass Assembler

- **First pass**
  - **Assign first hex location via ORG**
  - **Assign sequential locations for each line of code containing instruction**
  - **Create symbol table**



- **Second pass**
  - **Resolve symbolic memory addresses via symbol table**
  - **Translate instruction symbols along with optional "I" into hex code**

©Achyuta Nand Mishra



©Achyuta Nand Mishra

ASSEMBLY LANGUAGE PROGRAM FEATURES

#PROGRAM COMMENTS

- The use of Comments throughout a program can improve its clarity, especially in Assembly Language.

- A Comment begins with Semicolon.

EXAMPLE

```
MOV AX, BX ; Adds the Content of BX with AX
```

#RESERVED WORDS

- Instructions : MOV, ADD
- Directives : END,SEGMENT
- Operators : FAR,OFFSET
- Predefined Symbols : @DATA

#IDENTIFIERS

- An Identifier (or symbol) is a name that you apply to an item in your program that you expect to reference. The two types of identifiers are NAME and LABEL.

- NAME : Refers to the Address of a data item
  COUNTER DB 0

- LABEL: Refer to the Address of an instruction, procedure, or segment.

```
MAIN PROC
A20: MOV AL,BL
```

#STATEMENTS

- An Assembly Program consists of a set of statements. The two types of statements are:

1. INSTRUCTION

- Instructions such as MOV & ADD which the Assembler translates to Object Code.

2. DIRECTIVES

- Directives tell the Assembler to perform a specific action, such as define a data item etc.

ASSEMBLY LANGUAGE PROGRAMMING USING MASM

GENERAL PATTERN FOR WRITING ALP IN MASM

```
[PAGE DIRECTIVE]
[TITLE DIRECTIVE]
[MEMORY MODEL DEFINITION]
[SEGMENT DIRECTIVES]
[PROC DIRECTIVES]
```


©Achyuta Nand Mishra

……

……

……

……

[END DIRECTIVES]

BASIC FORMAT OF ALP BASED UPON THE GENERAL PATTERN

```
PAGE 60,80
TITLE "ALP TO PRINT FACTORIAL NO"

.MODEL [MODEL NAME]
.STACK
.DATA
 ……  ; INITIALIZE DATA VARIALBLES
.CODE

MAIN PROC

……
……..
……..
…… ; INSTRUCTION SETS
……..
……
……..
……

MAIN ENDP
END MAIN
```


©Achyuta Nand Mishra

DIRECTIVES

- Assembly Language supports a number of statements that enable to control the way in which a source program assembles and lists. These Statements are called Directives.
- They act only during the assembly of a program and generate no machine executable code.
- The most common Directives are PAGE, TITLE, PROC, and END.

PAGE DIRECTIVE

- The PAGE Directive helps to control the format of a listing of an assembled program.
- It is optional Directive.
- At the start of program, the PAGE Directive designates the maximum number of lines to list on a page and the maximum number of characters on a line.
- Its format is
  PAGE [LENGTH],[WIDTH]
- Omission of a PAGE Directive causes the assembler to set the default value to PAGE 50,80

TITLE DIRECTIVE

- The TITLE Directive is used to define the title of a program to print on line 2 of each page of the program listing.
- It is also optional Directive.
- Its format is
  TITLE [TEXT]

  TITLE "PROGRAM TO PRINT FACTORIAL NO"

SEGMENT DIRECTIVE

- The SEGMENT Directive defines the start of a segment.
- A Stack Segment defines stack storage, a data segment defines data items and a code segment provides executable code.
- MASM provides simplified Segment Directive.
- The format (including the leading dot) for the directives that defines the stack, data and code segment are

```
.STACK [SIZE]
.DATA
     ……. Initialize Data Variables
.CODE
```

- The Default Stack size is 1024 bytes.
- To use them as above, Memory Model initialization should be carried out.

©Achyuta Nand Mishra

MEMORY MODEL DEFINTION

- The different models tell the assembler how to use segments to provide space and ensure optimum execution speed.
- The format of Memory Model Definition is

  .MODEL [MODEL NAME]

- The Memory Model may be TINY, SMALL, MEDIUM, COMPACT, LARGE AND HUGE.

MODEL TYPE	DESCRIPTION
TINY	All DATA, CODE & STACK Segment must fit in one Segment of Size <=64K.
SMALL	One Code Segment of Size <=64K. One Data Segment of Size <=64 K.
MEDIUM	One Data Segment of Size <=64K. Any Number of Code Segments.
COMPACT	One Code Segment of Size < =64K. Any Number of Data Segments.
LARGE	Any Number of Code and Data Segments.
HUGE	Any Number of Code and Data Segments.

#THE PROC DIRECTIVE

- The Code Segment contains the executable code for a program, which consists of one or more procedures, defined initially with the PROC Directive and ended with the ENDP Directive.
- Its Format is given as:

```
PROCEDURE NAME    PROC
                  ……..
                  …….
                  …….
PROCEDURE NAME    ENDP
```

#END DIRECTIVE

- As already mentioned, the ENDP Directive indicates the end of a procedure.
- An END Directive ends the entire Program and appears as the last statement.
- Its Format is

  END [PROCEDURE NAME]

©Achyuta Nand Mishra

#PROCESSOR DIRECTIVE

- Most Assemblers assume that the source program is to run on a basic 8086 level.
- As a result, when you use instructions or features introduced by later processors, you have to notify the assemblers be means of a processor directive as .286,.386,.486 or .586
- This directive may appear before the Code Segment.

#THE EQU DIRECTIVE

- It is used for redefining symbolic names

EXAMPLE
DATAX DB 25
DATA EQU DATAX

#THE .STARTUP AND .EXIT DIRECTIVE

- MASM 6.0 introduced the .STARTUP and .EXIT Directive to simplify program initialization and Termination.
- .STARTUP generates the instruction to initialize the Segment Registers.
- .EXIT generates the INT 21H function 4ch instruction for exiting the Program.

©Achyuta Nand Mishra

DEFINING TYPES OF DATA

- The Format of Data Definition is given as
  [NAME] DN [EXPRESSION]

EXAMPLES

```
STRING DB 'HELLO WORLD'
NUM1 DB 10
NUM2 DB 90
```

DEFINITION	DIRECTIVE
BYTE	DB
WORD	DW
DOUBLE WORD	DD
FAR WORD	DF
QUAD WORD	DQ
TEN BYTES	DT

- Duplication of Constants in a Statement is also possible and is given by

[NAME] DN [REPEAT-COUNT DUP (EXPRESSION)]

EXAMPLES

```
DATAX DB 5 DUP(12)          ; 5 Bytes containing hex 0c0c0c0c0c
DATA DB 10 DUP(?)           ; 10 Words Uninitialized

DATAZ DB 3 DUP(5 DUP(4))    ; 44444 44444 44444
```

1. CHARACTER STRINGS

- Character Strings are used for descriptive data.
- Consequently DB is the conventional format for defining character data of any length
- An Example is
  DB 'Computer City'
  DB "Hello World"
  DB "Bio Medical College"

2. NUMERIC CONSTANTS

```
#BINARY        : VAL1 DB 10101010B

#DECIMAL       : VAL1 DB 230

#HEXADECIMAL   : VAL1 DB 23H
```


©Achyuta Nand Mishra

ALP SAMPLES USING DOS AND VIDEO BIOS FUNCTIONS

 # SAMPLE 1

```
;Program to Print Hello World in ALP
;Microprocessor Programming
;Bio Medical college
;Compiled By Achyuta Nand Mishra

;-------------------------------------------------------------
.MODEL SMALL
.STACK
.DATA
     STRING DB 'HELLO WORLD $'
.CODE

;-------------------------------------------------------------

MAIN PROC
     MOV AX,@DATA
     MOV DS,AX             ; Initialize the DATA Segment

     MOV DX,OFFSET STRING ; Load the Offset Address into DX
     MOV AH,09H            ; AH=09H For String Display until $
     INT 21H               ; DOS Interrupt Function

     MOV AX,4C00H          ; End Request with AH=4CH
                           ; or AX=4C00H
     INT 21H
     MAIN ENDP             ; End Procedure
END MAIN                   ; End Program


;-------------------------------------------------------------
```


©Achyuta Nand Mishra

#SAMPLE 2

```
;Program to Print the Sum of Two 8 Bit Numbers
;Microprocessor Programming
;Bio Medical College
;Compiled ByAchyuta Nand Mishra
;-------------------------------------------------------------
.MODEL SMALL
.STACK
.DATA
        VAL1 DB 89
        VAL2 DB 10
        MSG DB 'SUM OF 2 NUMBERS: $'
.CODE

;-------------------------------------------------------------

MAIN PROC
    MOV AX,@DATA
    MOV DS,AX

    MOV AX,0
    MOV AL,VAL1
    ADD AL,VAL2

    AAM          ;AAM Converts Binary Value to Unpacked BCD.


    ADD AX,3030H ;Ax is Added with 3030H to Obtain ASCII Value


    PUSH AX
    ;;;;;;;;;;;;;DISPLAY MESSAGE
    LEA DX,MSG
    MOV AH,09H
    INT 21H
    ;;;;;;;;;;;;;END DISPLAY MESSAGE

    POP AX

    MOV DL,AH
    MOV DH,AL
    MOV AH,02H
    INT 21H

    MOV DL,DH
    MOV AH,02H
    INT 21H

    MOV AX,4C00H
    INT 21H
MAIN ENDP
END MAIN

;-------------------------------------------------------------
```


©Achyuta Nand Mishra

#SAMPLE 3

```
;Program To Clear the Screen With Bios Interrupt Function.
;Microprocessor Programming
;Bio Medical college
;Compiled By Achyuta Nand Mishra
;2005

;-------------------------------------------------------------
.MODEL SMALL
.STACK
.DATA
.CODE
;-------------------------------------------------------------

MAIN PROC
        MOV AX,@DATA
        MOV DS,AX

        ;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;INT 10H Function 06H : Scroll Up Screen
        ;AH=Function 06H
        ;AL=Number of lines to scroll,or 00H for full screen
        ;BH=Attribute value(color,blinking)
        ;CX=Strating row:column
        ;DX=Ending row :column
        ;;;;;;;;;;;;;;;;;;;;;;;;;;

        MOV AX,0600H  ;AH=06,AL=00 for Full Screen
        MOV BH,71H    ;White background(7),Blue foreground(1)
        MOV CX,0000H  ;Upper left row:column
        MOV DX,184FH  ;lower right row:column
        INT 10H       ;Bios Interrupt Function

        MOV AX,4C00H
        INT 21H

MAIN ENDP
END MAIN
;-------------------------------------------------------------
```


©Achyuta Nand Mishra

#SAMPLE 4

```
;Progam to Display Character with Attributes
;Microprocessor Programming
;Bio medical college Kathmandu
;Compiled Achyuta Nand Mishra
;2005

;-------------------------------------------------------------
.MODEL SMALL
.STACK
.DATA
.CODE
;-------------------------------------------------------------

MAIN PROC
        MOV AX,@DATA
        MOV DS,AX

        ;;;;;;;;;;;;;;;;;;;;;;
        ;INT 10H Function 09H: Display Character
        ;AH=09H
        ;AL=ASCII Character
        ;BH=Page Number
        ;BL=Attribute or Pixel value.
        ;CX=Count
        ;;;;;;;;;;;;;;;;;;;;;;

        MOV AH,09H  ;Request Display
        MOV AL,01H  ;Happy Face for display
        MOV BH,00H  ;Page Number 0
        MOV BL,0C1H ;Red background,Blue foreground
        MOV CX,79   ;No of repeated characters
        INT 10H     ;Bios Interrupt Function

        MOV AX,4C00H
        INT 21H

MAIN ENDP
END MAIN
;-------------------------------------------------------------
```


©Achyuta Nand Mishra

Chapter IV

Bus structure and Memory Devices

©Achyuta Nand Mishra

BUS TIMING

BASIC BUS OPERATION

- The Three buses of 8086 and 8088 are Address, Data and Control.
- If Data are written to the memory, the MPU outputs the memory address on the address bus, outputs the data to be written into memory on the data bus, and issues a write($\overline{WR}$) to memory.
- If Data are to be read from the memory, the MPU outputs the memory address on the address bus, issues a read ($\overline{RD}$) memory signal and accepts data via data bus.

TIMING IN GENERAL

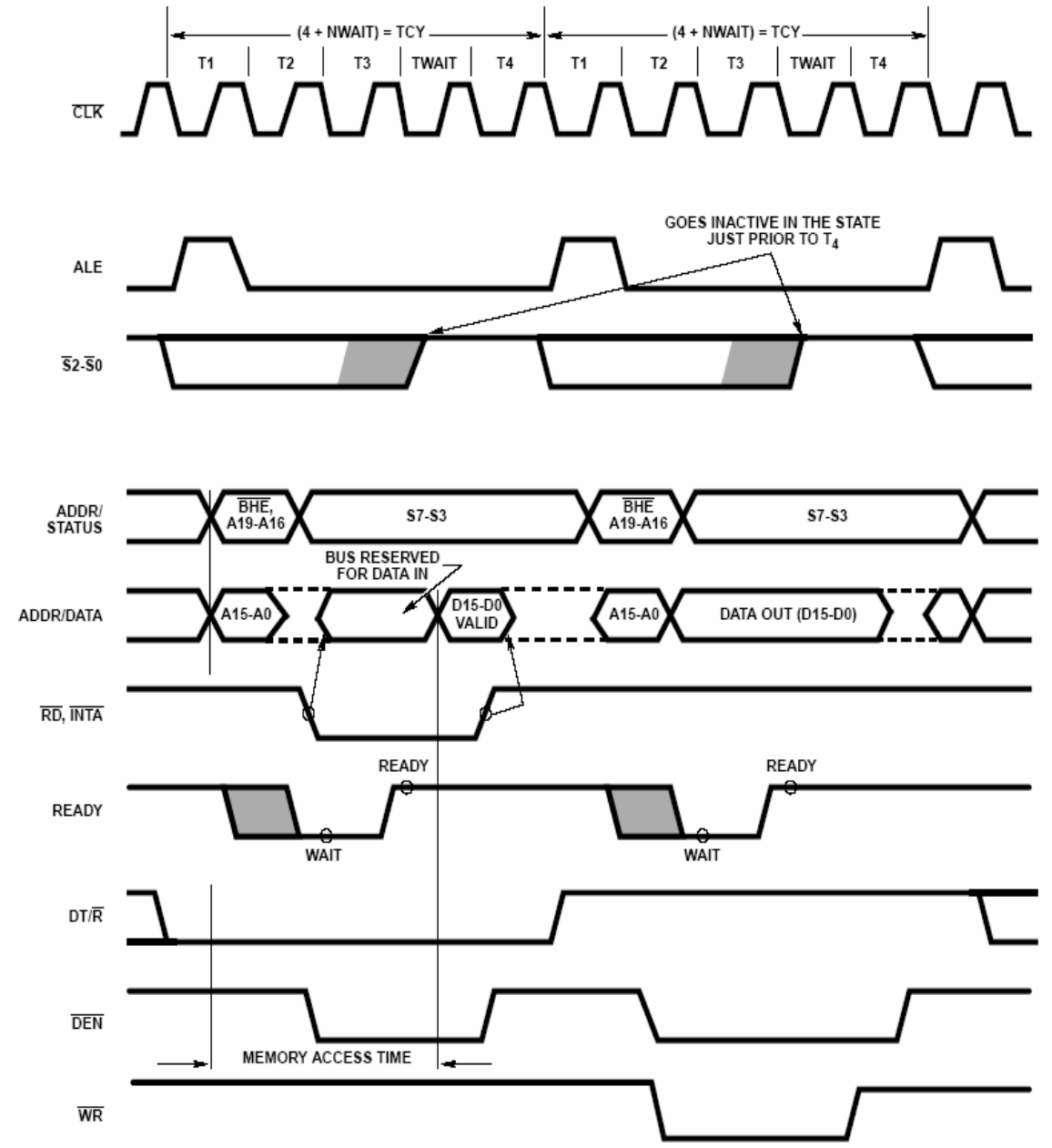
- One clock cycle is called a state.
- For reference, a state is measured from the falling edge of one clock pulse to the falling edge of next clock pulse.
- Each bus cycle equals for system clocking period
- 1 Bus Cycle = 4T States
- If the clock is operated in 5MHZ, one bus cycle is completed in 800 ns (200 ns x 4 = 800 ns)
- This means that MP reads/write data between itself and memory or I/O at a rate of 1.25 million times a second.
- A basic MPU operation such as reading a byte from memory or writing a byte to a port is called a Machine Cycle.
- The time a MPU requires to fetch and execute an entire instruction is referred to as an instruction cycle.
- An instruction consists of one or more Machine cycles.
- To summarize, an Instruction cycle is made up of Machine cycle and a Machine cycle is made up of T states.
- The time for a state is determined by the frequency of clock pulse/signal.

©Achyuta Nand Mishra

SIMPLIFIED 8086/8088 WRITE BUS CYCLE

SIMPLIFIED 8086/8088 READ BUS CYCLE

TIMING DIAGRAM



©Achyuta Nand Mishra

READ BUS CYCLE [EXPLANATION]

- The 8086 asserts M/$\overline{IO}$ high if the read is to be from memory and asserts M/$\overline{IO}$ low if the read is going to be from a port.
- At about the same time it asserts ALE high to enable the external address latches.
- It then sends out $\overline{BHE}$ and the desired address on the AD0-A19 lines.
- When ALE is pulled low, the address information is latched on the outputs of the external latches.
- The Address/Data line is asserted into input mode [floating mode].
- The 8086 then asserts its $\overline{RD}$ [READ] signal low.
- The RD signal going lo turns on the addressed memory or port , which then outputs the desired data on the data bus.
- To complete the cycle the 8086 brings the $\overline{RD}$ line high again.
- This causes the addressed memory or port to float its output on the data bus.
- If the 8086 READY input is made low before or during T2 of Machine cycle, the 8086 will assert WAIT states as long as the READY I/P is low.
- When READY is made high, the 8086 will continue with T4 of the Machine cycle.
- WAIT states can be used to give slow devices additional time to put out valid data.
- The DT/$\overline{R}$ is asserted low.
- The 8086 $\overline{DEN}$ signal will enable the buffer at the appropriate time in Machine cycle.

©Achyuta Nand Mishra

#WRITE BUS CYCLE [EXPLANATION]

- During T1 of a write Machine cycle the 8086 asserts M/$\overline{IO}$ high for Memory Write and M/$\overline{IO}$ low for I/O Write.
- The 8086 raises ALE high to enable address latches.
- The 8086 then outputs $\overline{BHE}$ low and send address on AD0-A19.
- When address is passed to latches ALE is pulled low
- After the address information is latched, the 8086 removes the address information and outputs the desired data on the data bus.
- It then asserts $\overline{WR}$ [WRITE] signal low.
- If the Memory/Port device cannot accept the data word within a normal machine cycle, external H/W can be set up to pulse READY input.
- If the system is large enough to need buffers, during Write cycle, the 8086 asserts DT/$\overline{R}$ high to put the buffers in transmit mode.
- When the 8086 asserts $\overline{DEN}$ low to enable the buffers, data output from the 8086 will pass through the buffers to the addressed port or memory location.

©Achyuta Nand Mishra

ADDRESS DECODING

- In order to attach a memory device to the MPU, it is necessary to decode the address sent from the MPU.
- Decoding makes the memory function at a unique section or partition of the memory map.
- Without an address decoder only one memory device can be connected to a MPU.
- When the 8088 MPU is compared to the 2716 EPROM, a difference in the number of address connection is apparent.
- The EPROM has 11 address connections whereas 8088 MPU has 20 address connections.
- This means that the MPU sends out a 20 bit memory address whenever it reads or writes data.
- Due to the 11 address pins, there is a mismatch and it must be corrected.
- The decoder corrects the mismatch by decoding the address pins that do not connect to the memory components.

ADDRESS DECODING TECHNIQUES

1.Simple NAND gate Decoder

2. The 3 to 8 Line Decoder (74LS138)

3. The PROM Decoder

©Achyuta Nand Mishra

CHAPTER – V

INPUT/OUTPUT INTERFACING

©Achyuta Nand Mishra

I/O INTERFACING

I/O SPACE VS. MEMORY SPACE

- The 80x86 processors use the M/IO signal to separate I/O space from memory space.
- When this signal is high, address output by the processor will be interpreted as memory location.
- However, when this line is low, those same address will be interpreted as I/O locations.
- I/O address are restricted to 16 bits , valid I/O address ranges from 0000H to FFFFH

TYPES OF I/O INSTRUCTIONS

1. DIRECT I/O

- The Direct I/O instructions supply the port address as a part of instructions
- Port address is defined from 00-FFH

EXAMPLES

```
IN AL,27H
MOV AX,BX
OUT 26H,AX
```

2. INDIRECT I/O

- To access the full range of I/O ports from 0000H – FFFFH , the indirect I/O instruction.

EXAMPLES

```
MOV DX,1000H
IN AL,DX

MOV AL,20H
OUT DX,AL
```

MODES OF CHANNEL COMMUNICATION

1.SIMPLEX
Data in a simples channel is always one way. E.g. : Radio Broadcasting.

2.HALF DUPLEX
A half duplex channel can send and receive but not at the same time. E.g: Police Radio

3.FULL DUPLEX
Data travels both directions simultaneously. E.g: Cell Phones, Telephone Lines

©Achyuta Nand Mishra

METHODS OF PARALLEL DATA TRANSFER

1. SIMPLE INPUT AND OUTPUT

- When you need to get digital data from a simple switch, all you have to do is connect the switch to an I/P port line and read the value.
- Likewise, when you need to O/P data to simple LED, all you have to do is connect the LED to an O/P port and send the value.
- The LED is always ready, so you can send data at any time.

2. SIMPLE STROBE I/O

- In many applications, valid data is present on an external device only at a certain time, so it must be read in at that time.
- When a key is pressed, circuitry on the keyboard sends out the ASCII code for the pressed key on eight parallel data line , and then sends out a strobe signal on another line to indicate that valid data is present on the eight data lines.
- For higher speed data transfer this method does not work.
- The sending system might send data bytes faster than the receiving system could read them. To prevent this handshake data transfer is required.

3. SINGLE HANDSHAKE I/O DATA TRANSFER

- The peripheral outputs some parallel data and sends STB signal to MPU.
- The MPU detects STB signal on a polled or interrupt basis and reads data byte.
- Then the MP sends on ACK signal to the peripheral to indicate that the data has been read and the peripheral can send to next byte of data.

4. DOUBLE HANDSHAKE I/O DATA TRANSFER

- For data transfers where coordination is required between sending system and the receiving system, a double handshake is used.
- The sending device asserts its STB low to ask "**Are you ready?"**
- The receiving system raises its ACK line high to say "**I'm ready".**
- The peripheral device then sends the data byte and raises its STB signal high to say **"Here is valid data for you".**
- When the receiving system finishes to read the data, receiving system drop its ACK line low to say **"I have data than you and I await your request to send the next byte of data".**

©Achyuta Nand Mishra

DATA COMMUNICATION

MODES OF COMMUNICATION

1. Serial Communication : E.g Serial Data Communication with RS232

2. Parallel Communication : E.g Parallel Data Communication with parallel port.

1. SYNCHRONOUS SERIAL DATA COMMUNICATION

- A more efficient method of transferring serial data is to synchronize the transmitter and the receiver and then send a large block of data characters one after the other with no time between characters.
- No start or stop bits are needed with individual data characters because the receiver automatically knows that every 8 bits received after synchronization represents a data character.
- To indicate start of transmission, transmitter sends out one or more unique characters called sync. Characters.
- The receiver uses the sync characters or the flag to synchronize its internal clock with that of the receiver.
- ISDN, High Speed Modems and Digital Communication Channel use synchronous transmission.

2. ASYNCHRONOUS SERAIL DATA COMMUNICATION

- For asynchronous transmission, each data character has a bit which identifies its start and 1 or 2 bits which identify end.
- Since each character is individually identified, character can be sent at any time asynchronously.
- The beginning of a data character is identified by the line going low for 1 bit time. This bit is called start bit.
- The data bits are then sent out on the line one after the other.
- Note that LSB is transmitted first.
- After parity bit, the signal line is returned high for at least 1 bit time to identify the end of the character. This high bit is always referred as stop bit.

UART : Universal Asynchronous Receiver Transmitter [IN 8250]

USART: Universal Synchronous Asynchronous Receiver Transmitter [INTEL 8251A]

©Achyuta Nand Mishra

8251 UNIVERSAL SYNCHRONOUS ASYNCHRONOUS RECEIVER TRANSMITTER

The 8251 is a USART (Universal Synchronous Asynchronous Receiver Transmitter) for serial data communication. As a peripheral device of a microcomputer system, the 8251 receives parallel data from the CPU and transmits serial data after conversion. This device also receives serial data from the outside and transmits parallel data to the CPU after conversion.

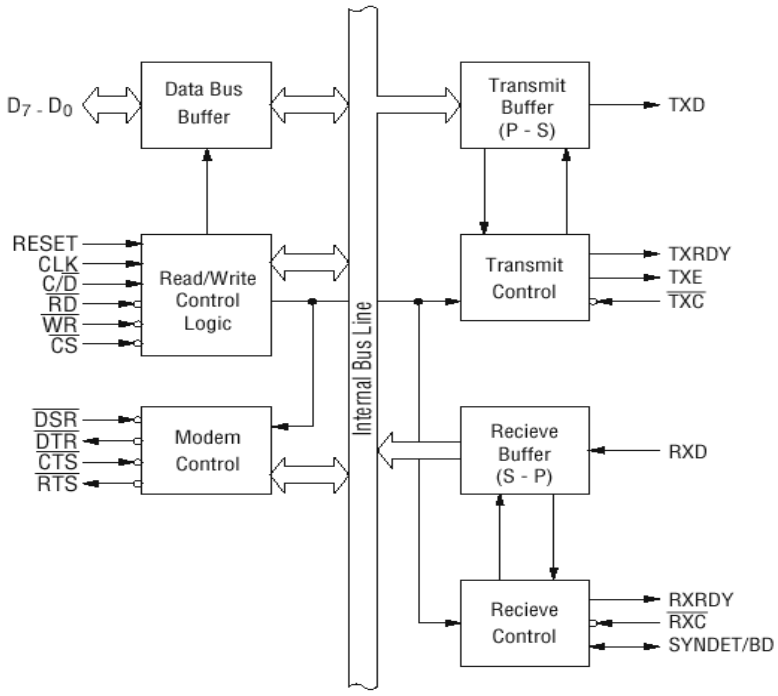


Figure: Block Diagram of the 8251 USART

The 8251 functional configuration is programmed by software. Operation between the 8251 and a CPU is executed by program control. Table 1 shows the operation between a CPU and the device.

$\overline{CS}$	C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	
1	×	×	×	Data Bus 3-State
0	×	1	1	Data Bus 3-State
0	1	0	1	Status → CPU
0	1	1	0	Control Word ← CPU
0	0	0	1	Data → CPU
0	0	1	0	Data ← CPU

©Achyuta Nand Mishra

CONTROL WORDS

1. MODE INSTRUCTION

Mode instruction is used for setting the function of the 8251. Mode instruction will be in "wait for write" at either internal reset or external reset. That is, the writing of a control word after resetting will be recognized as a "mode instruction."

Items set by mode instruction are as follows:

- Synchronous/asynchronous mode
- Stop bit length (asynchronous mode)
- Character length
- Parity bit
- Baud rate factor (asynchronous mode)
- Internal/external synchronization (synchronous mode)
- Number of synchronous characters (Synchronous mode)

The bit configuration of mode instruction is shown in Figures 2 and 3. In the case of synchronous mode, it is necessary to write one-or two byte sync characters. If sync characters were written, a function will be set because the writing of sync characters constitutes part of mode instruction.

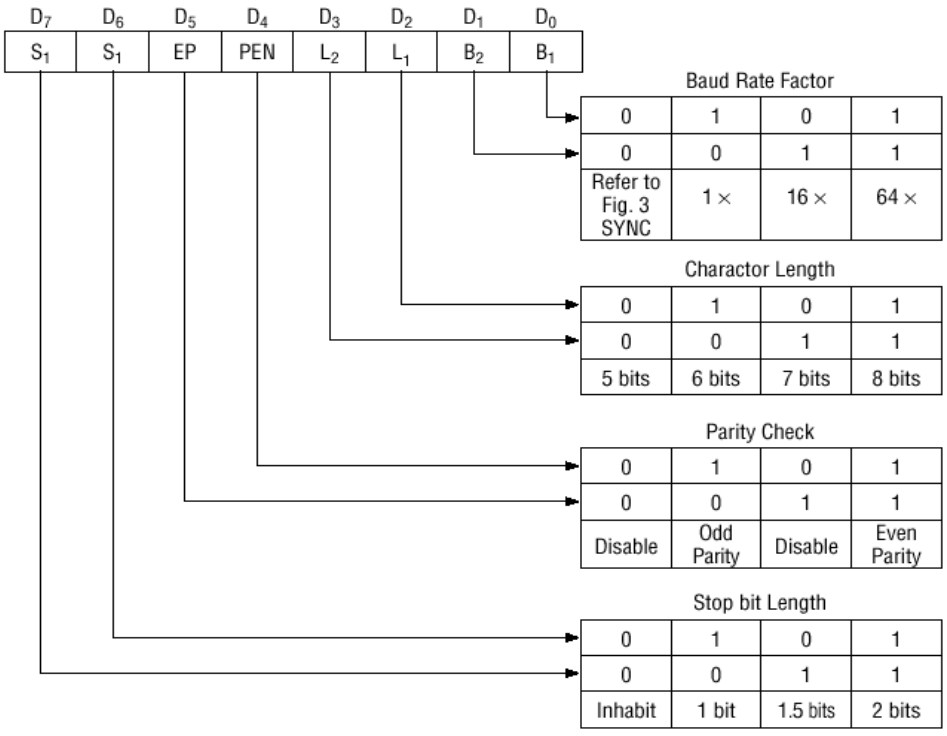


Fig. 2 Bit Configuration of Mode Instruction (Asynchronous)

©Achyuta Nand Mishra

2. COMMANAND WORD

Command is used for setting the operation of the 8251. It is possible to write a command whenever necessary after writing a mode instruction and sync characters.

Items to be set by command are as follows:

- Transmit Enable/Disable
- Receive Enable/Disable
- DTR, RTS Output of data.
- Resetting of error flag.
- Sending to break characters
- Internal resetting
- Hunt mode (synchronous mode)

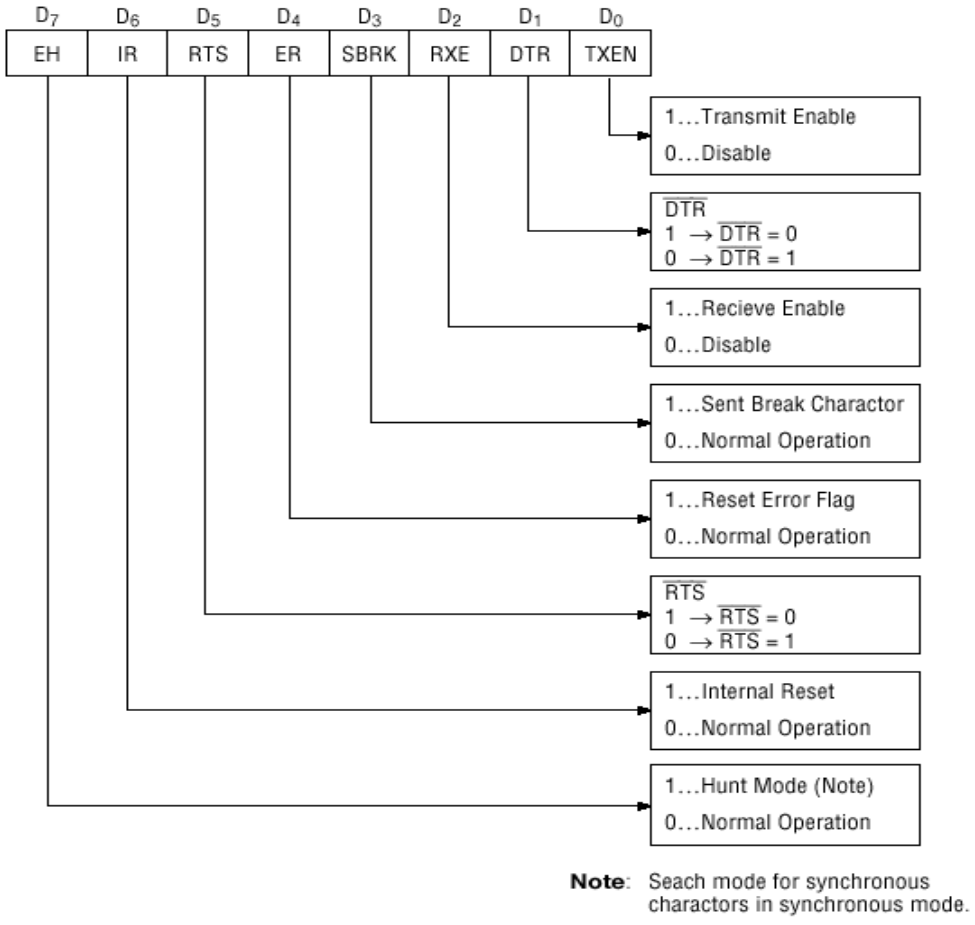


Fig. 4 Bit Configuration of Command

©Achyuta Nand Mishra

3. STATUS WORD

It is possible to see the internal status of the 8251 by reading a status word. The bit configuration of status word is shown in Fig. 5.

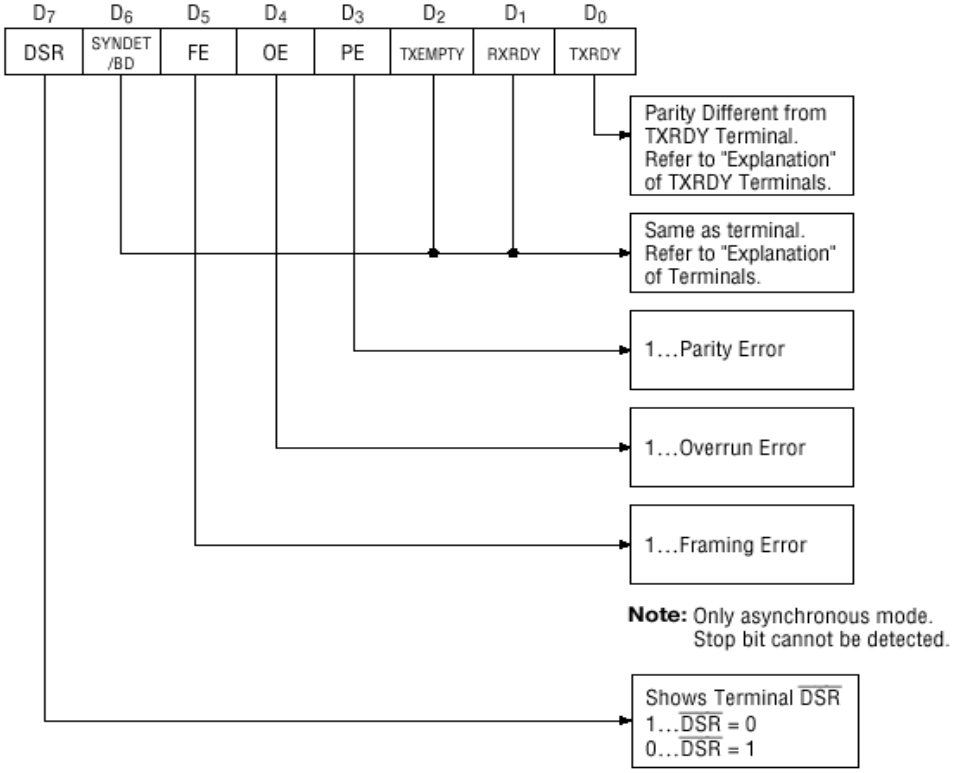


Fig. 5 Bit Configuration of Status Word

©Achyuta Nand Mishra

PIN DESCRIPTION

D0 to D7 [I/O TERMINAL]

This is bidirectional data bus which receive control words and transmits data from the CPU and sends status words and received data to CPU.

RESET [INPUT TERMINAL]

A "High" on this input forces the 8251 into "reset status." The device waits for the writing of "mode instruction." The min. reset width is six clock inputs during the operating status of CLK.

CLK [INPUT TERMINAL]

CLK signal is used to generate internal device timing. CLK signal is independent of RXC or TXC. However, the frequency of CLK must be greater than 30 times the RXC and TXC at Synchronous mode and Asynchronous "x1" mode, and must be greater than 5 times at Asynchronous "x16" and "x64" mode.

WR [INPUT TERMINAL]

This is the "active low" input terminal which receives a signal for writing transmit data and control words from the CPU into the 8251.

RD [INPUT TERMINAL]

This is the "active low" input terminal which receives a signal for reading receive data and status words from the 8251.

C/D [INPUT TERMINAL]

This is an input terminal which receives a signal for selecting data or command words and status words when the 8251 is accessed by the CPU. If C/D = low, data will be accessed. If C/D = high, command word or status word will be accessed.

CS [INPUT TERMINAL]

This is the "active low" input terminal which selects the 8251 at low level when the CPU accesses. Note: The device won't be in "standby status"; only setting CS = High.

TXD [OUTPUT TERMINAL]

This is an output terminal for transmitting data from which serial-converted data is sent out. The device is in "mark status" (high level) after resetting or during a status when transmit is disabled. It is also possible to set the device in "break status" (low level) by a command.

TXRDY [OUTPUT TERMINAL]

This is an output terminal which indicates that the 8251is ready to accept a transmitted data character. But the terminal is always at low level if CTS = high or the device was set in "TX disable status" by a command. Note: TXRDY status word indicates that transmit data character is receivable, regardless of CTS or command.

©Achyuta Nand Mishra

TXEMPTY [OUTPUT TERMINAL]

This is an output terminal which indicates that the 8251 has transmitted all the characters and had no data character. In "synchronous mode," the terminal is at high level, if transmit data characters are no longer remaining and sync characters are automatically transmitted. If the CPU writes a data character, TXEMPTY will be reset by the leading edge of WR signal. Note: As the transmitter is disabled by setting CTS "High" or command, data written before disable will be sent out. Then TXD and TXEMPTY will be "High". Even if a data is written after disable, that data is not sent out and TXE will be "High". After the transmitter is enabled, it sent out.

TXC [INPUT TERMINAL]

This is a clock input signal which determines the transfer speed of transmitted data. In "synchronous mode," the baud rate will be the same as the frequency of TXC. In "asynchronous mode", it is possible to select the baud rate factor by mode instruction. It can be 1, 1/16 or 1/64 the TXC. The falling edge of TXC sifts the serial data out of the 8251.

RXD [INPUT TERMINAL]

This is a terminal which receives serial data.

RXRDY [OUTPUT TERMINAL]

This is a terminal which indicates that the 8251 contains a character that is ready to READ. If the CPU reads a data character, RXRDY will be reset by the leading edge of RD signal. Unless the CPU reads a data character before the next one is received completely, the preceding data will be lost. In such a case, an overrun error flag status word will be set.

RXC [INPUT TERMINAL]

This is a clock input signal which determines the transfer speed of received data. In "synchronous mode," the baud rate is the same as the frequency of RXC. In "asynchronous mode," it is possible to select the baud rate factor by mode instruction. It can be 1, 1/16, 1/64 the RXC.

SYNDET/BD [INPUT OR OUTPUT TERMINAL]

This is a terminal whose function changes according to mode. In "internal synchronous mode." this terminal is at high level, if sync characters are received and synchronized. If a status word is read, the terminal will be reset. In "external synchronous mode, "this is an input terminal. A "High" on this input forces the 8251 to start receiving data characters.

In "asynchronous mode," this is an output terminal which generates "high level" output upon the detection of a "break" character if receiver data contains a "low-level" space between the stop bits of two continuous characters. The terminal will be reset, if RXD is at high level.

DSR [INPUT TERMINAL]

This is an input port for MODEM interface. The input status of the terminal can be recognized by the CPU reading status words.

©Achyuta Nand Mishra

DTR [OUTPUT TERMINAL]

This is an output port for MODEM interface. It is possible to set the status of DTR by a command.

CTS [INPUT TERMINAL]

This is an input terminal for MODEM interface which is used for controlling a transmit circuit. The terminal controls data transmission if the device is set in "TX Enable" status by a command. Data is transmittable if the terminal is at low level.

RTS [OUTPUT TERMINAL]

This is an output port for MODEM interface. It is possible to set the status RTS by a command.

©Achyuta Nand Mishra

PROGRAMMING 8251A

- According to the data sheet, the 8251A requires a worst case recovery time of 16 cycles.
- This means that you have to delay 16 processor clock cycles.
- The initialization sequence of 8251A is hence lengthy.
- The second reason is that, the 8251A does not respond correctly to a hardware reset on power up. Therrefore a series of software command must be sent to the device to make sure it reset properly before the desired mode and command word are sent.
- The 8251A distinguishes a command word from a mode word by the order in which thery are sent to the device.
- Any words sent to the command address after the mode word will be treated as command word until the device is reset.

TECHNIQUES

- A simple way to produce the required delay and a margin of safety is to load CX with 2 and count it down with the loop instruction
- LOOP first -> 17 clock cycles
- LOOP last -> 5 clock cycles

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOV DX,0FFF2H  ;Command Register Address of 8251A
MOV AL,00H
OUT DX,AL
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
MOV CX,2      ;4 Clock Cycles
D0:LOOP DO  ;17+5 Clock Cycles
OUT DX,AL    ;8 Clock Cycles
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
MOV CX,2      ;4 Clock Cycles
D1:LOOP DO  ;17+5 Clock Cycles
OUT DX,AL    ;8 Clock Cycles
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
MOV AL,40   ; Send Internal Reset
OUT DX,AL
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOV CX,2
D2:LOOP D2

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOV AL,11001110 b ;Load Mode Word
OUT DX,AL

MOV CX,2
D2:LOOP D2

MOV AL,00110111 b ; Load Command Word
OUT DX,AL

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```


©Achyuta Nand Mishra

TRASNMITTING SERIAL DATA

1. Point at control register/status register.

2. Read status

3. Check status of DSR and Transmit Ready (Data set Ready)

4. If yes transmit. If no continue to step 1

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOV DX,0FFF2 H              ;Point at control register

TEST1:
        IN AL,DX                ;Read status
        AND AL,10000001 b  ;Check status of DSR and Tx Ready
        CMP AL,10000001 b  ;Is it ready
        JNE TEST1               ;Continue to poll if not equal
        MOV DX,0FFF0 H      ;Point at data address
        MOV AL,DATA_SEND ;Load data to send and sent it
        OUT DX,AL

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

RECEIVING SERIAL DATA

1. Point at control register

2. Read status

3. Check status of RxRdy

4. If yes receive. If no continue step 2

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOV DX,0FFF2 H

TEST2:
        IN AL,DX
        AND AL,00000010 b
        CMP AL,00000010 b
        JNE TEST2
        MOV DX,0FFF0 H
        IN AL,DX

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

 ©Achyuta Nand Mishra

Chapter VI

Interrupt

©Achyuta Nand Mishra

8086 INTERRUPT PROCESSING

INTERRUPT AND INTERRUPT APPLICAITON

INTRODUCTION

- An Interrupt is an event that causes the processor to suspend its present task and transfer the present task and transfer control to new program called ISR.
- Most MPU allow normal program execution to be interrupted by some external signal or by a special instruction in the program.
- In response to an interrupt, the MPU stops executing its current program and calls a procedure which services the interrupt

SOURCES OF INTERRUPTS

There are three sources of an interrupts and they are as follows.

1. **PROCESSOR INTERRUPT**
   These interrupts are generated by the processor itself, usually in response to an error condition. For example a type 0 interrupt occurs when attempt to divide by zero.

2. **SOFTWARE INTERRUPT**
   These are special 80x86 instructions that trigger an interrupt response to processor. The general form of the software interrupt instruction is INT 10H.

3. **HARDWARE INTERRUPT**
   Hardware interrupts are interrupt requests initiated by external hardware. 8086 has 2 pins reserved for this purpose (NMI and INTR)

THE PURPOSE OF INTERRUPTS

- Interrupts are particularly useful when interfacing I/O device that provides or require data relatively low data transfer rates.
- Unlike the polling sequence, interrupt processing allows the MPU to execute other softwares while the keyboard operator is thinking about what key to type next.
- As soon as the key is pressed, the keyboard encoder de-bounces the switch and puts out one pulse that interrupts the MPU.

TYPES OF INTERRUPT

1. MASKABLE INTERRUPT

Those interrupts which can be blocked by user using instructions are called maskable interrupt.
STI-> Enable INTR Input.
CLI-> Disable INTR Input.

2. NON MASKABLE INTERRUPT

Those interrupts which cannot be blocked by instructions are termed as non maskable interrupt.

NMI-> Non Maskable Interrupt

©Achyuta Nand Mishra

8086 INTERRUPTS

- The interrupt of entire Intel family of MPU include 2 hardware pins that request interrupts (INTR and NMI).
- And other one hardware pin INTA acknowledges the interrupt request through INTR.
- An 8086 interrupt can come from any one of 2 sources
- One source is an external signal applied to the Nonmaskable Interrupt (NMI) or to the (INTR) input pin.
- An interrupt caused by a signal applied to one of these inputs (NMI or INTR) is referred as **Hardware Interrupt**.
- A second source of an interrupt is execution of the interrupt instruction,INT. This is referred as **Software Interrupt.**
- The third source of an interrupt is some error condition produced in the 8086 by the execution of an instruction.

INTERRUPT VECTORS

- The Interrupt Vector Table is located in the first 1024 bytes of memory at address 00000H – 0003FFH.
- It contains 256 different 4 byte Interrupt Vectors.
- An Interrupt Vector contains the address (segment and offset) of the Interrupt Service Procedure.
- Each vector is a 4 byte long and contains the starting address of the Interrupt Service Routine.
- The first 2 bytes of the vector contain the offset address and the last two bytes contain the segment address.

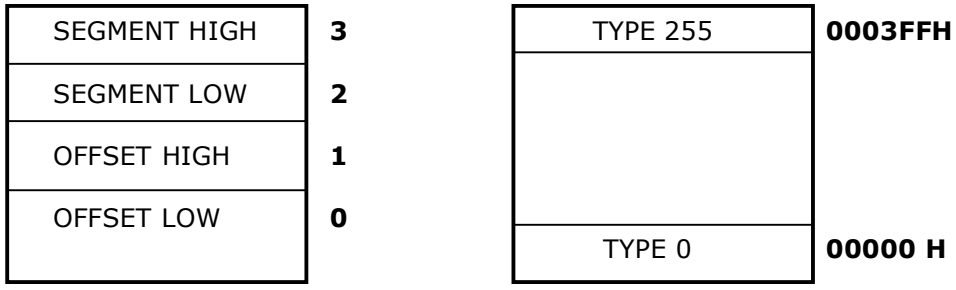


Figure : Interrupt Vector and Interrupt Vector Table

©Achyuta Nand Mishra

8086 INTERRUPT TYPES

1. DIVIDE BY ZERO INTERRRUPT: TYPE 0

- Divide error occurs when the result of division overflow or whenever an attempt is made to divide by zero.
- The 8086 type 0 is automatic and cannot be disabled in any way

2. SINGLE STEP INTERRUPT : TYPE 1

- If the 8086 trap flag is set, the 8086 will automatically do a type 1 interrupt after each instruction executes.
- Upon accepting this interrupt, the TF bit is cleared so that Interrupt Service Procedure executes.
- When the 8086 does a type 1 interrupt, it pushes the flag register on the stack, resets TF and IF and pushes CS and IP value for the next instruction on the stack.
- It gets CS value for the start of type 1 interrupt service procedure from address 00006H and gets IP value for the start of the procedure from the address 00004H.
- When you tell a system to single step, it will execute one instruction and stop.
- You can then examine the contents of registers and memory locations.
- In other words, in single step mode, a system will stop after it executes each instruction and waits for further direction from you.

3. NON MASKABLE INTERRUPT : TYPE 2

- A result of placing logic 1 on the NMI input pin causes type 2 interrupt.
- This input is non-maskable, which means that it cannot be disabled.
- Another common use of type 2 interrupt is to save program data in case of a system power failure or to deal with some other catastrophic failure conditions.
- Some external circuitry detects when the AC power to the system fails and sends an Interrupt signal to the NMI.

4. BREAKPOINT INTERRUPT : TYPE 3

- The INT 3 instruction is often used to store a breakpoint in a program for debugging.

5. OVERFLOW INTERRUPT : TYPE 4

- A special vector used with the INT0 instruction.
- The INT0 instruction interrupts the program if an overflow condition exists as reflected by overflow flag (OF).

©Achyuta Nand Mishra

PRIORITY OF 8086 INTERRUPTS

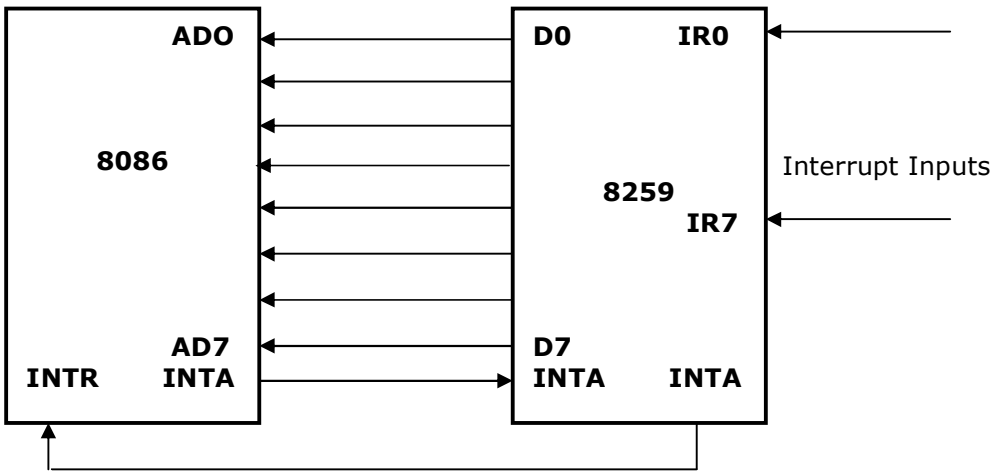
INTERRUPTS	PRIORITY
DIVIDE ERROR,INT n , INT0	HIGHEST
NMI	
INTR	
SINGLE STEP	LOWEST

8259A PRIORITY INTERRUPT CONTROLLER

2. 8086 have only 2 interrupt input NMI and INTR

3. If we save NMI for a power failure (catastrophic condition), only one pin is left for all other applications.

4. We use an external device called a Priority Interrupt Controller to funnel the interrupt signals into a single interrupt input on the processor



8259A PRIORIRTY INTERRUPT WITH 8086

1. 8086 will send out 2 Interrupt Ack pulse on its INTA pin to the INTA pin of the 8259A PIC.The INTA pulse tell the 8259A to send the desired interrupt type to the 8086 on the data bus.

2. Multiply the interrupt type it receives from the 8259A by 4 to produce an address in the Interrupt Vector Table.

3. Push the flags on the stack.

4. Clear IF and TF

5. Push the return address on the stack.

6. Get the starting address for ISR (CS: IP).

7. Execute Interrupt Service Routine.

8. Return From Interrupt Service Routine.

©Achyuta Nand Mishra

COMPUTING THE ISR ADDRESS

5. In Real Mode the address of the ISR is stored in four consecutive memory locations (Double Word) in an IVT.

6. When an Interrupt occurs an 8 bit no is supplied to the processor, which identifies the appropriate entry in this table.

7. The method for determining the type no depends on the interrupt source.

    1. Software Interrupts supply the no as a part of instruction (INT n , where n is the type number)

    2. Internal interrupts have predefined type number for example TYPE 0 – Divide by Zero.

        - NMI hardware interrupt is predefined as type 2.
        - INTR however has no predefined type number and must get its type no onto data bus D0-D7.

LIST OF PROBLEMS

PROBLEM 1

A particular real mode interrupt has a type no n = 41H. If the corresponding ISR begins at address 09E3:0010, determine the location in the vector table to store this address.

SOLUTION

- The vector address is calculated by multiplying 41H by 4.

- This is done most easily by rotating 41H left twice

  41H = 01000000 -> Rotate Twice -> **0001 0000 0100 = 104H**
  Hence 00104H is the starting address.

```
00107 = 09  |
00106 = E3  |   ->    CODE SEGMENT

00105 = 00  |
00104 = 10  |   ->    INSTRUCTION POINTER
```

PROBLEM 2

Calculate the IVT address when operated in real mode of the 80x86 processor's NMI interrupt input. Assume the vector points to memory location E010:1000H

PROBLEM 3

An 80x86 processor stores the byte 3A49:2F1C beginning at address 0008CH. What interrupt type number does this address correspond to?

 ©Achyuta Nand Mishra

INTERRUPT LATENCY

- The term interrupt latency refers to the amount of time it takes a system to respond to an interrupt.

HOW FAST DOES THE SYSTEM RESPOND TO EACH INTERRUPT

1. The long period of time during which that interrupt is disabled.
2. The period of time it takes to execute any interrupt routines for interrupts that are of higher priority.
3. How long it takes the microprocessor to stop what it is doing.
4. How long it takes the interrupt routine to save the context.

VECTORED AND POLLED INTERRUPT

1. VECTORED INTERRUPT

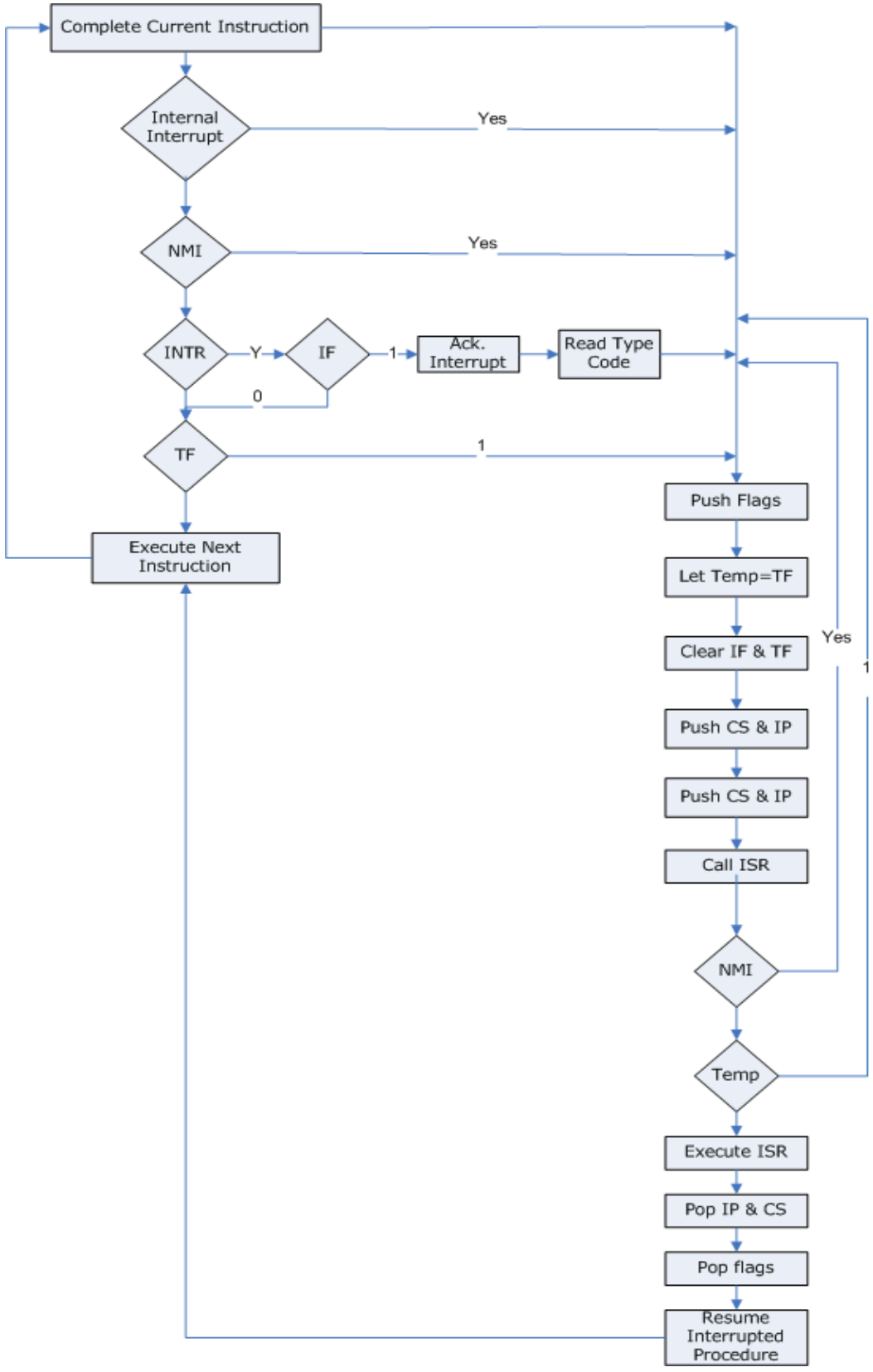
- An interrupt signal that includes the identity of the device sending the interrupt signal.
- In a computer, a vectored interrupt is an I/O interrupt that tells the part of the computer that handles I/O interrupts at the hardware level that a request for attention from an I/O device has been received and also identifies the device that sent the request.
- In a vectored interrupt, the source that interrupts supplies the branch information (Interrupt Vector) to the CPU.

2. POLLED INTERRUPT

- An interrupt handler that requires the interrupt handler poll or send a signal to each device in order to find out which one sent the interrupt request.
- In a computer, a polled interrupt is a specific type of I/O interrupt that notifies that part of the computer containing the I/O interface that a device is ready to be read or otherwise handled but does not indicate which device.
- The interrupt controller must poll each device to determine which one made the request.
- In a non-vectored interrupt, the branch address is assigned to a fixed location in the memory.

©Achyuta Nand Mishra

INTERRUPT PROCESSING SEQUENCE



©Achyuta Nand Mishra

Chapter VII
Multi-programming

©Achyuta Nand Mishra

©Achyuta Nand Mishra