MICROPROCESSOR

CSC - 153

Er. Anil Sah
6/4/2015

The course objective is to introduce the operation , programming and application of microprocessor.

UNIT 1

INTRODUCTION TO MICROPROCESSOR

MICROPROCESSOR

- **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.**

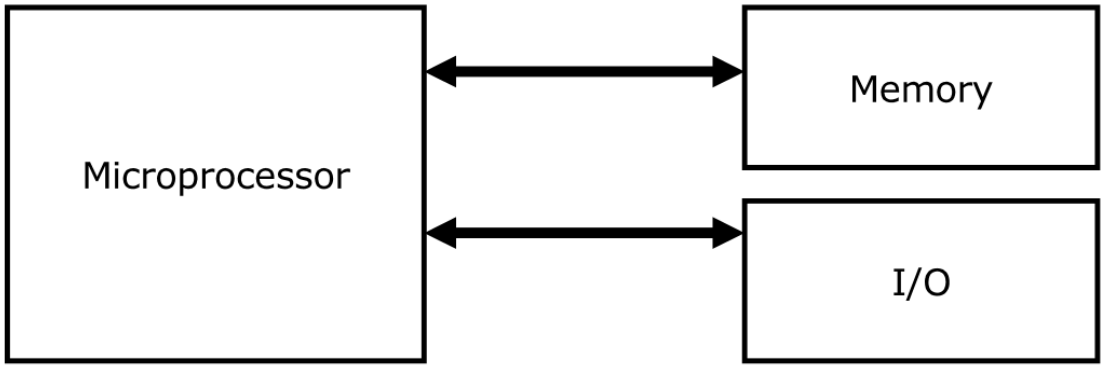


Figure: A Programmable Machine

- **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.

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**

MEMORY CALCULATIONS

- **210 =1K(Kilo)**
- **220=1M(Mega)**
- **230=1G(Giga)**
- **240=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**

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 220 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.

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.

Stored Program concept: The task of entering and altering the programs for the ENIAC (electronic numerical integrator and computer) was extremely tedious. The programming concept could be faciliated if the program could represented in a form suitable for storing in memory along side the data. Than a computer could get it's instruction by reading them form the memory and a program could be set or

altered by setting the values of a portion of memory . This approach is known stored program concept.

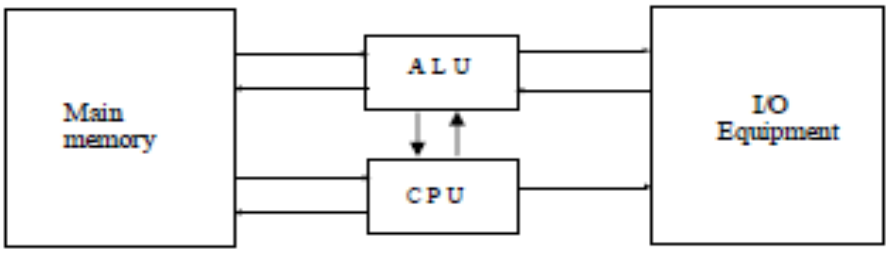


Fig: Von-Numann Architecture.

Main memory is used to store both data and instruction ALU is capable for performing Arithmetic and logical operation binary data. The program control unit(cpu) interprets the instruction in memory and causes them to be a execute. The input/output unit and helps inputting data and getting results. The memory of Von-Neumann machine consists of thousand storage location called words of 40 binary digits(bits). Both data and instruction are stored in it. The storage locations of control unit and ALU are called registers. The various registers of this model are MBR, MAR, IR, IBR, PC, AC.

Memory Buffer Register: It consist of a word to be stored in memory or is used to receives a memory or is used to receive a word from memory.

Memory address Register: It contain the address in memory of the world to be written from or read into the MBR.

IR(Instruction register): Contain the 8 bit upcode (operation code) instruction being executed.

IBR(instruction buffer register): It is used to temporarily hold the instruction from a word in memory.

PC (program counter): It contain address of next instruction to be fetched from memory.

Ac (Accumulator) and MQ(multiplier quotient): They are employed to temporarily hold operands and results of ALU operations.

Harvard Architecture:

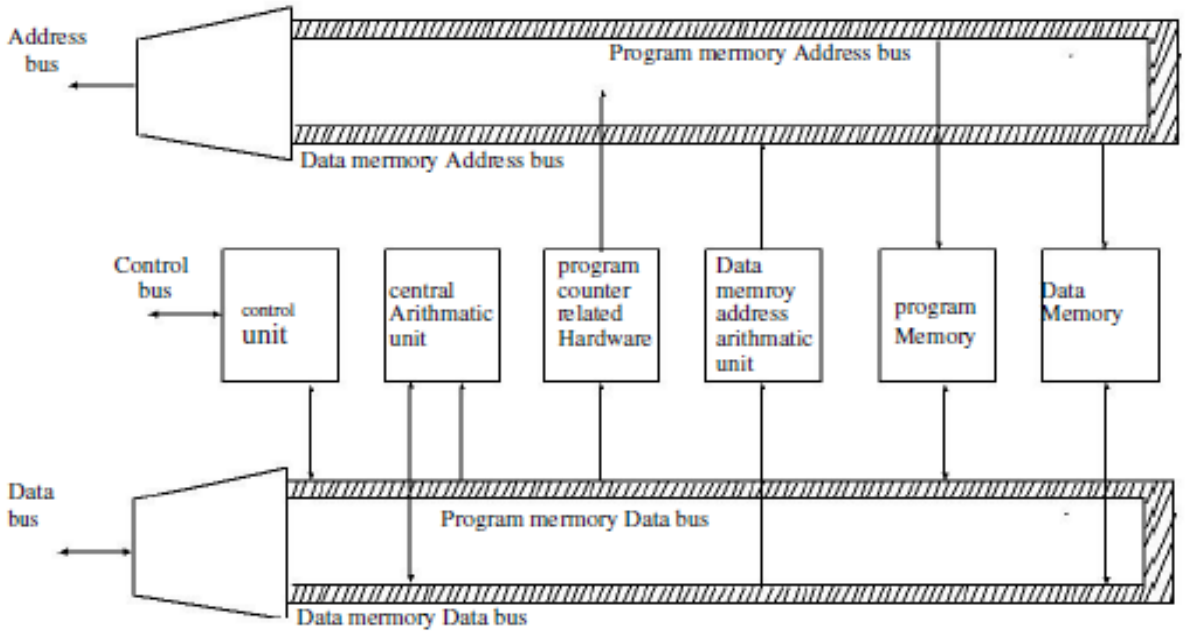


Fig. Block diagram of the Harvard architecture based µP

Harvard Architecture based computer consist so separate memory spaces for the programs (or instruction) and data.Each memory space has it's own address and data bus. Thus both instruction and data can fetch from memory concurrently. From the figure it is seen that there are two data and two and address buses for the program and data memory spaces respectively. The program memory data bus and data memory data are multiplexed to form single data bus where as program memory Address and data memory address are multiplexed to form single address bus. Hence there are two blocks of ram chip. One for program memory and another for data memory space. Data memory address arithmetic unit generates data memory address. The data memory address bus carries the memory address of data

where as program memory address bus carries the memory address of the instruction. Central arithmetic logic unit consists of the ALU, multiplier, Accumulator, etc. The program counter is used to address program memory. Pc always contains the address of next instruction to be fetched. Control unit control the sequence of operations to be executed. The data and control bus are bidirectional where as address bus is unidirectional.

GENERAL ARCHITECTURE OF MICROCOMPUTER SYSTEM

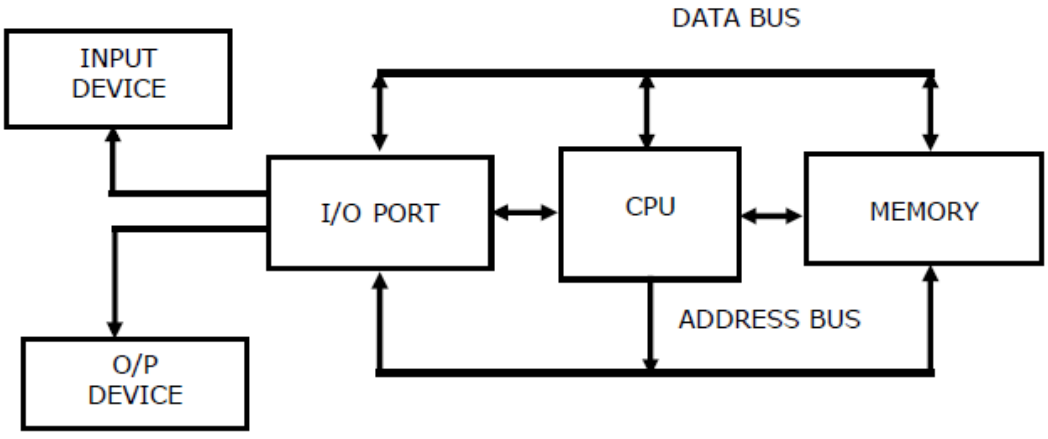


Figure: Block Diagram of a simple Microcomputer

- **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.
INPUT
DEVICE
O/P
DEVICE
I/O PORT CPU MEMORY

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 2N memory locations i.e. CPU with 16 address lines can address 216 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.

COMPONENTS OF CPU

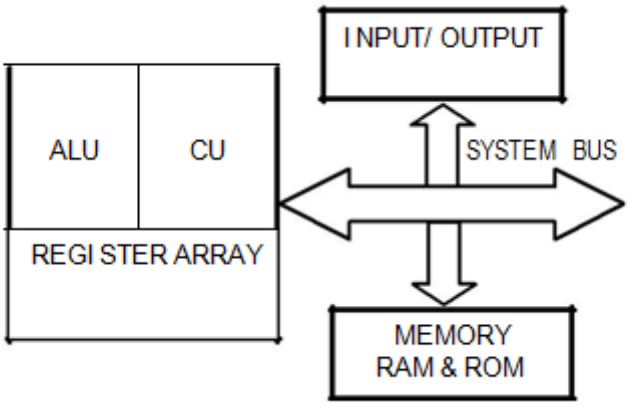


Figure: Microprocessor Based System with Bus Architecture.

• 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.

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.**

LEVEL OF INTEGRATION

SSI: Small Scale Integration
- **No of gates Less than 10.**

MSI: Medium Scale Integration
- **No of gates between 10-100.**

LSI: Large Scale Integration
- **No of gates between 100-1000.**

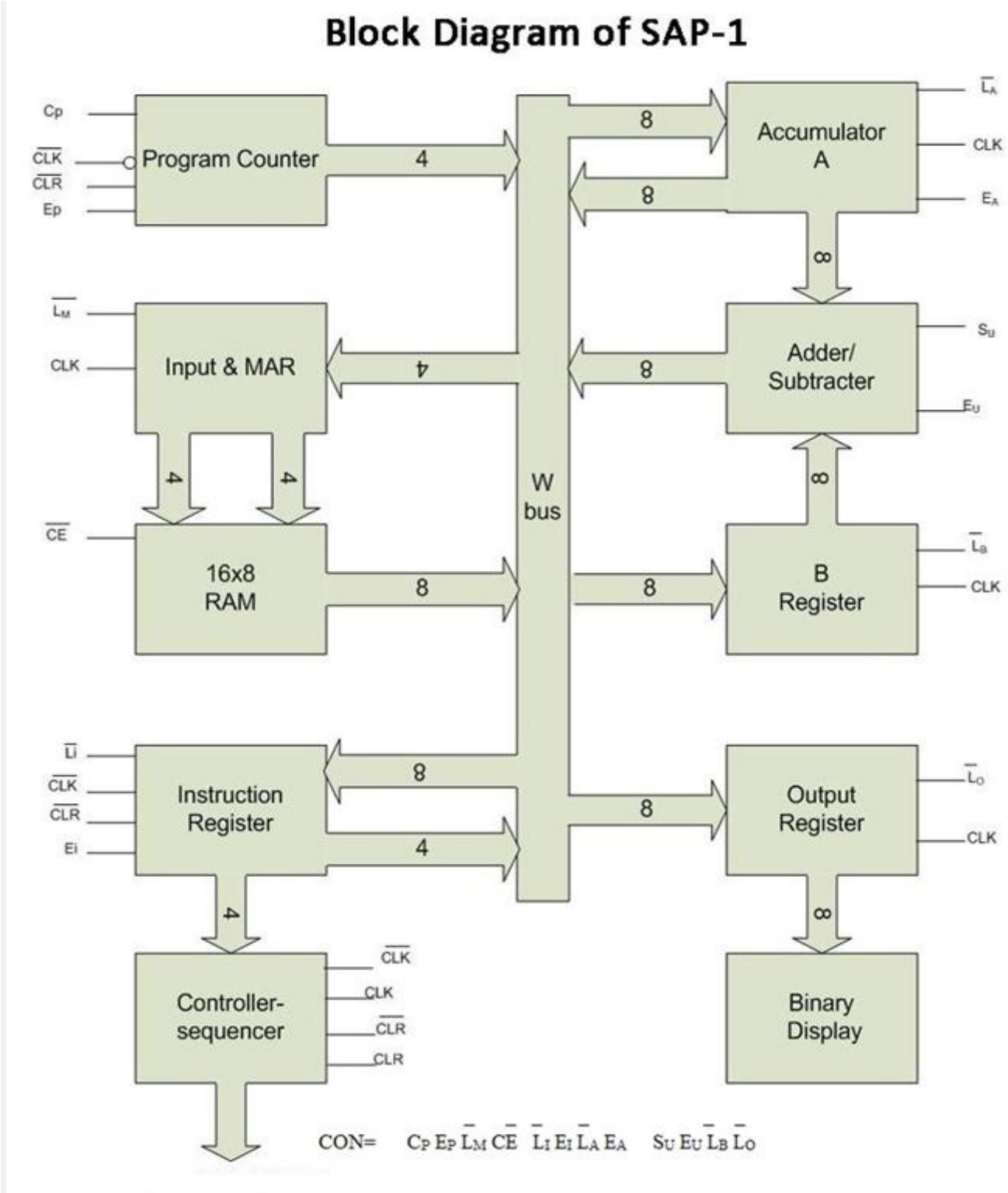
VLSI: Very Large Scale Integration
- **More than 1000 gates in a single chip.**

ULSI: Ultra Large Scale Integration
- **Millions of gates in a single chip.**

Unit 2

Basic Computer Architecture

SAP-1(Simple as Possible-1) Computer Architecture Introduction



Block Diagram of SAP 1 Computer

The Simple-As-Possible (SAP)-1 computer is a very basic model of a microprocessor explained by Albert Paul Malvino. The SAP-1 design contains the basic necessities for a functional Microprocessor. Its primary purpose is to develop a basic understanding of how a microprocessor works, interacts with memory and other parts of the system like input and output. The instruction set is very limited and is simple.

The features in SAP-1 computer are:

- **W bus – A single 8 bit bus for address and data transfer.**
- **16 Bytes memory (RAM)**
- **Registers are accumulator and B-register each of 8 bits.**
- **Program counter – initializes from 0000 to 1111 during program execution.**
- **Memory Address Register (MAR) to store memory addresses.**
- **Adder/ Subtracter for addition and subtraction instructions.**
- **A Control Unit**
- **A Simple Output.**
- **6 machine state reserved for each instruction**
- **The instruction format of SAP-1 Computer is**

(XXXX) (XXXX)

The first four bits make the Opcode while the last four bits make the operand (address).

Program Counter:

The program/Instructions are stored at the beginning of the memory with the first instruction at binary address 0000, the second at 0001 and so on. The PC is a part of the control unit. Its job is to send to the memory address register, the address of the next instruction to be executed and fetched. The PC is reset to 0000 before each computer run

Input and MAR:

The memory address register (MAR) is latched with the address of the pc during a computer run. A bit later, the MAR applies this 4-bit address to the RAM, where a real read operation is performed. It has some switch registers which helps it to do so.

RAM:

The RAM is a 16×8 static TTL (Transistor Transistor Logic) RAM. During a computer run, the RAM receives 4-bit addresses from the MAR and a READ operation is performed. In this way, the instruction or data word stored in the RAM is placed on the W-bus.

Instruction Register:

1) The instruction register is part of the control unit.
2) To fetch an instruction from the memory the computer does a memory read operation. This places the contents of the addressed memory location on the W-bus.
3) At the same time, the IR is set up for loading on the next positive clock edge.
4) The contents of the IR are split into two nibbles.
5) The upper nibble is a two state output that goes directly to the block labeled 'Controller-sequencer'.
6) The lower nibble is a three state output that is read onto the W-bus when needed.

Controller-Sequencer:

1) Before each computer run, (CLR') signal is sent to the PC and CLR signal to the IR.
2) This resets the PC to 0000 and wipes out the last instruction in the IR.
3) A clock signal CLK is sent to all buffer registers, this

synchronizes the operation of the computer.
4) The 12 bits that come out of the CS form a word controlling the rest of the computer. The 12 wires carrying the control word are called the control bus.
5) The control word has the format:

$$CON= \quad C_P E_P \overline{L_M} \overline{C_E} \quad \overline{L_I} E_I \overline{L_A} E_A \quad S_U E_U \overline{L_B} \overline{L_O}$$

This word determines how the registers will react to the next positive CLK edge.

Accumulator:

1) The accumulator is a buffer register that stores immediate answers during a computer run.
2) It has two output. The first one goes directly to the Adder-Subtracter.
3) The three state output goes to the W-bus when E_A is high.

Adder-Subtractor:

1) When S_U is low, the sum out of the adder-subtracter is S=A+B

2) When S_U is high, the sum out of the adder-subtracter is S=A+B'

3) The adder-subtractor is asynchronous (unlocked); this means that its contents can change as soon as the input words change.

4) When E_U is high, these contents appear on the W-bus.

B Register:

1) The B register is also a buffer register.
2) A low L_B' and positive CLK edge load the word on the W-bus into the B-register.
3) The two state output of the B register drives the B-register.

Output Register:

1) At the end of a computer run, the accumulator contains the answer to the problem being solved.
2) At this point, we need to transfer the answer to the outside world. This is where the output register is used.
3) When E_A is high, is low, the next positive clock edge loads the word of the accumulator into the output register.
4) The output register is often called an output port processed data can leave the computer through these register.

Output Port:

1) The binary display is a row of 8 LEDs.
2) Each LED connects to one flip-flop of the output port.
3) After we have transferred an answer from the accumulator to the output port, we can see the answer in binary form.

INSTRUCTION FORMAT:

Instruction of SAP-1 is of 8 bit length

XXXX XXXX

First 4 bits are Opcode and last 4 bits are Operand

Example: **LDA 9H and its binary equivalent is 0000 1001**

INSTRUCTION SET:

	Mnemonic	Operand Number	Operation	Opcode
1	LDA	1	Load memory data to ACC	0000
2	ADD	1	Add ACC data with B register data	0001
3	SUB	1	Sub B register data from data in ACC	0010
4	OUT	0	Move out the ACC data to the output register	1110
5	HLT	0	Stop program	1111

PROGRAMMING SAP-1:

Example:

Write a program to compute 16+20+24-32 (decimal) and display result in SAP-1 computer

Solution:

```
LDA 8H
ADD 9H
ADD AH
SUB BH
OUT
HLT
```

In SAP-1 Computer, before program Execution begins Instruction and data are stored in static TTL (Transistor-Transistor Logic) 16 X8 RAM.

0H	LDA 8H
1H	ADD 9H
2H	ADD AH
3H	SUB BH
4H	OUT
5H	HLT
6H	
7H	
8H	10H
9H	14H
AH	18H
BH	20H
CH	
DH	
EH	
FH	

Instructions and Data (Hexadecimal) in SAP-1 Memory

0H	0000 1000
1H	0001 1001
2H	0001 1010
3H	0010 1011
4H	1110
5H	1111
6H	
7H	
8H	0001 0000
9H	0001 0100
AH	0001 1000
BH	0010 0000
CH	
DH	
EH	
FH	

Instructions and Data (binary) in SAP-1 Memory

Instruction Cycle:

Fetch Cycle

- **T1 (Address State)**
- **T2 (Increment State)**
- **T3 (Memory State)**

Execution Cycle

- **3 step (T4, T5, T6), but the task of each steps depends on the instruction**

FETCH CYCLE

The control unit is the key to a computer's automatic operation. The CU generates the control words that fetch and execute each instruction. While each instruction is fetched and executed, the computer passes through different timing states (T states), periods during which register contents modify

CON=	C_P	E_P	L'_M	C'_E	L'_I	E'_I	L'_A	E_A	S_U	E_U	L'_B	L'_O	
	0	0	1	1	1	1	1	0	0	0	1	1	Initial State
CON=	C_P	E_P	L'_M	C'_E	L'_I	E'_I	L'_A	E_A	S_U	E_U	L'_B	L'_O	
	0	1	0	1	1	1	1	0	0	0	1	1	T_1 (Address) State
CON=	C_P	E_P	L'_M	C'_E	L'_I	E'_I	L'_A	E_A	S_U	E_U	L'_B	L'_O	
	1	0	1	1	1	1	1	0	0	0	1	1	T_2 (Increment) State
CON=	C_P	E_P	L'_M	C'_E	L'_I	E'_I	L'_A	E_A	S_U	E_U	L'_B	L'_O	
	0	0	1	0	0	1	1	0	0	0	1	1	T_3 (Memory) State

Red text denotes the active control signal during each T state

Control Signal During Each T state

Execution Cycle (LDA Instruction)

- **For LDA instruction, only T4 and T5 states that will be active**
- **T4: memory address is sent from IR to MAR**
- **T5: data from memory is fetched and send to ACC**
- **T6: do nothing (No Op)**

Execution Cycle (ADD and SUB Instruction)

- **For ADD instruction, T4, T5 and T6 states that will be active**
- **T4: memory address is sent from IR to MAR**

- **T5: data from memory is fetched and send to B register**
- **T6: Addition/ Subtraction takes place using the values stored in accumulator and B register. Addition takes place in ALU if S_U low, Subtraction takes place if S_U is high. Calculated value is stored in ACC when E_U is high through w-bus.**

Execution Cycle (OUT Instruction)

- **For OUT instruction, only T4 state will be active**
- **T4: E_A and L_O' will be active so data stored in Accumulator is loaded into Output register and displayed in the output unit.**

Execution Cycle (HLT Instruction)

There is no execution cycle for HLT Cycle. When IR sends 1111 to the controller, it halts computer by turning off the clock

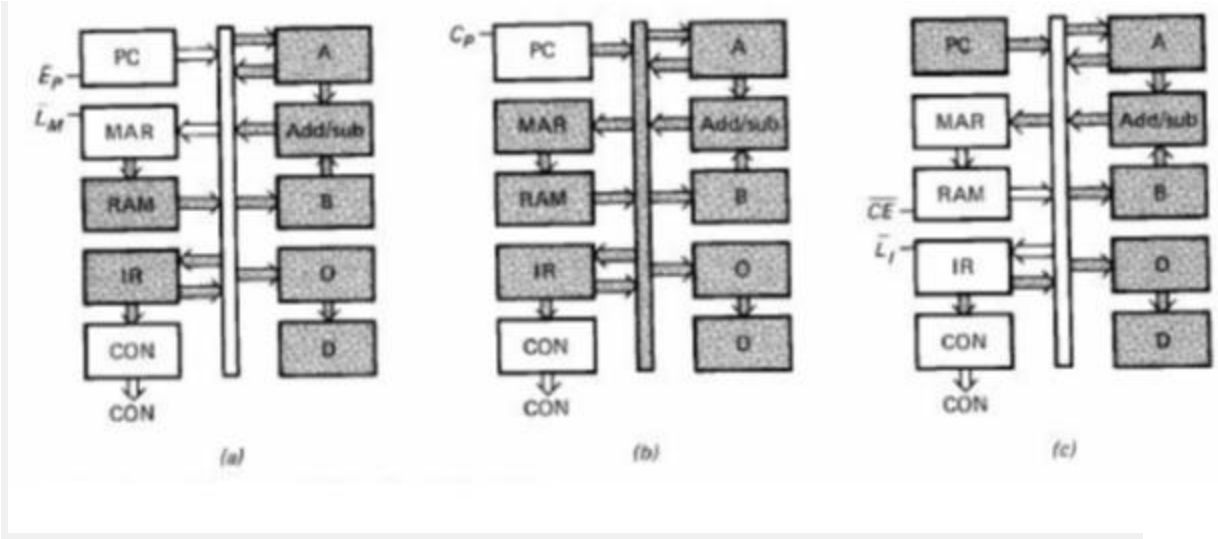


Figure: Fetch Cycle of SAP-1 Computer during program Execution

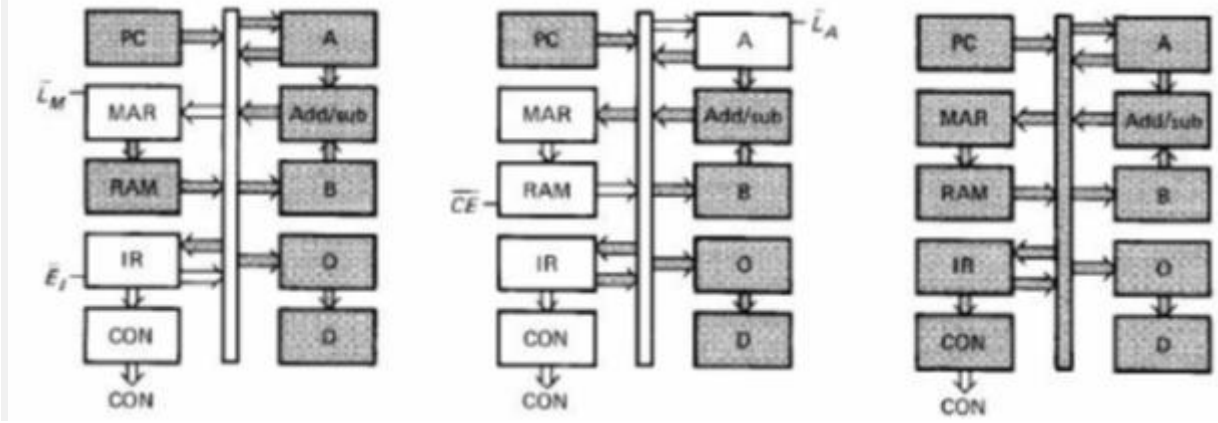


Figure: Execution Cycle of LDA Instruction

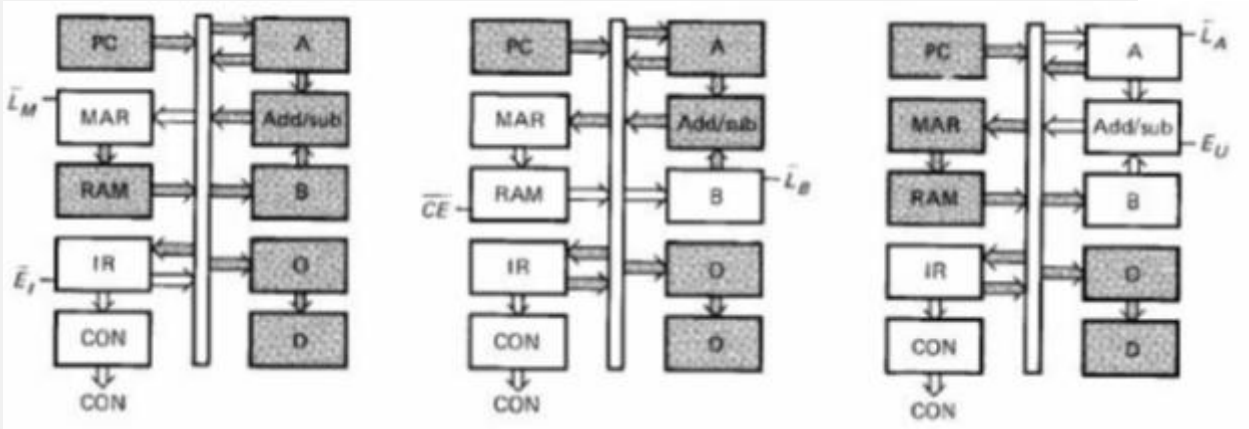


Figure: Execution Cycle of ADD Instruction

Timing Diagrams

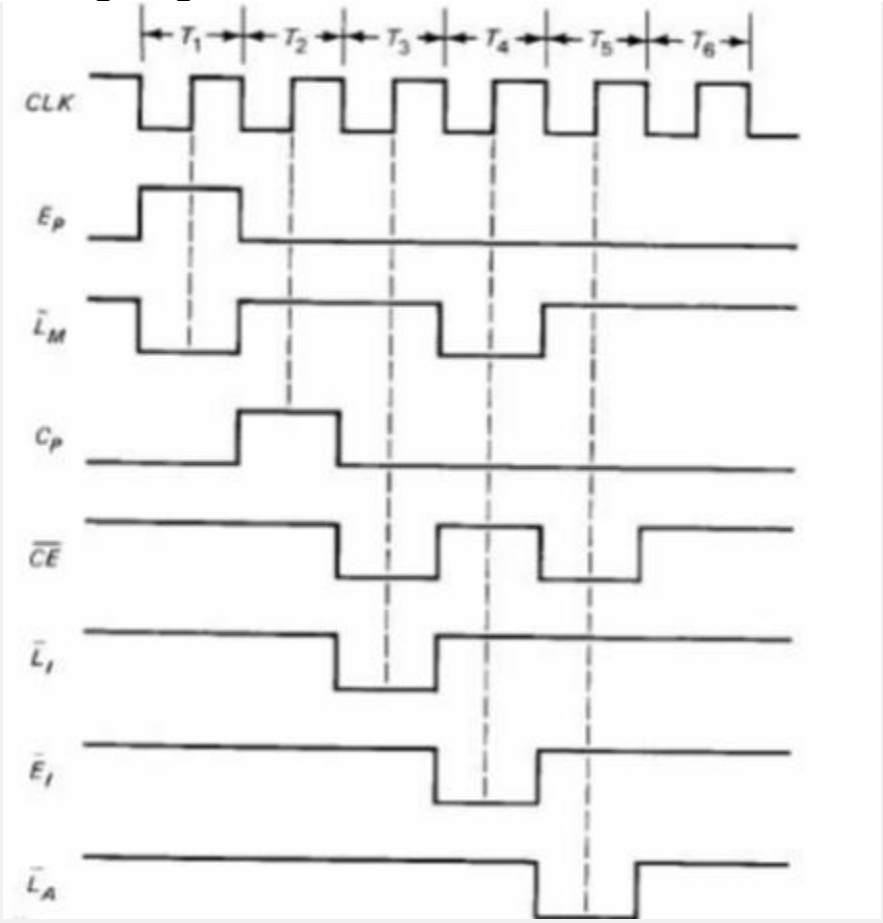


Figure: Fetch and LDA timing diagram

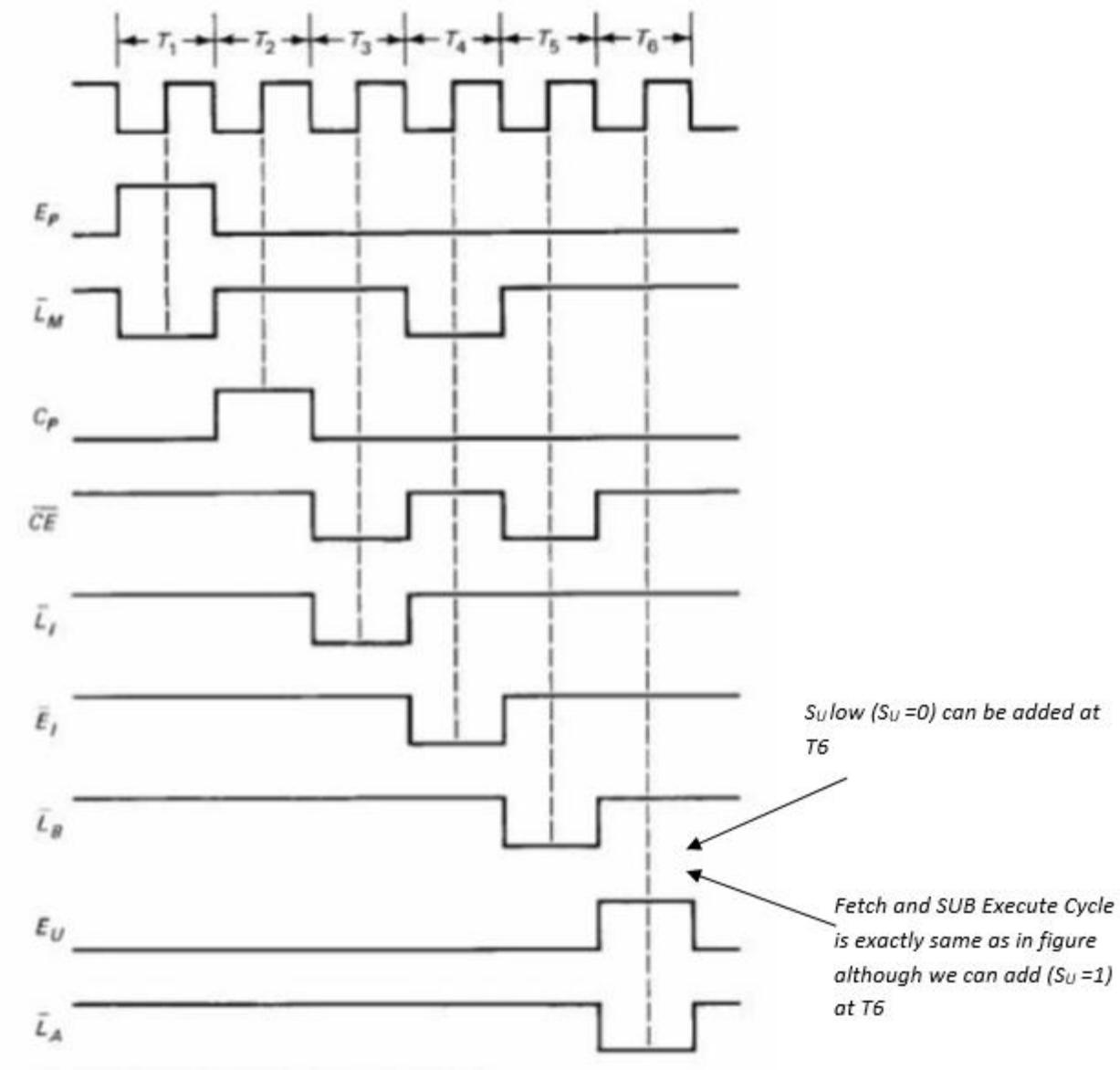


Figure: Fetch and ADD timing diagram

The Sap-1 Microprogram

Microinstructions

The controller-sequence sends out control words, one during each T- state or clock cycle. These words are like directions telling the rest of the computer what to do. Because it produces a small step in the data processing, each control word is called a micro-instruction.

Macro-instruction

The instructions LDA, ADD, SUB are sometimes called macro-instructions. Each Sap-1 macro-instruction is made up of three micro-instructions (i.e. T4, T5 and T6 state).

Control Matrix

The LDA, ADD, SUB and OUT signals from the instruction decoder drive the control matrix, at the same time, the ring counter signals, T1 to T6, are driving the matrix. The matrix produces CON, a 12-bit micro-instruction that tells the rest of the computer what to do.

Macro	State	CON	Active
LDA	T_4	1A3H	$\overline{L}_M$, $\overline{E}_I$
	T_5	2C3H	$\overline{CE}$, $\overline{L}_A$
	T_6	3E3H	None
ADD	T_4	1A3H	$\overline{L}_M$, $\overline{E}_I$
	T_5	2E1H	$\overline{CE}$, $\overline{L}_B$
	T_6	3C7H	$\overline{L}_A$, E_U
SUB	T_4	1A3H	$\overline{L}_M$, $\overline{E}_I$
	T_5	2E1H	$\overline{CE}$, $\overline{L}_B$
	T_6	3CFH	$\overline{L}_A$, S_U, E_U
OUT	T_4	3F2H	E_A, $\overline{L}_O$
	T_5	3E3H	None
	T_6	3E3H	None

† CON = $C_P E_P \overline{L}_M \overline{CE}$ $\overline{L}_I \overline{E}_I \overline{L}_A E_A$ $S_U E_U \overline{L}_B \overline{L}_O$.

Figure: SAP -1 Micro-Program

Microprogramming

The control matrix is one way to generate the microinstructions needed for each execution cycle. With larger instruction sets, the control matrix becomes very complicated and requires hundreds or even thousands of gates. Microprogramming is the alternative. The basic idea is to store microinstructions in a ROM rather than produce them with a control matrix. This approach simplifies the problem of building a controller-sequencer.

Storing the Micro-Program

These microinstructions can be stored in control ROM with the fetch routine at addresses at 0H to 2H, the LDA routine at addresses 3H to 5H,the ADD routine at 6H to 8H. To access any routine, we need to supply the correct addresses. For instance, to get the ADD routine, we need to supply addresses 6H, 7H and 8H.To get the OUT routine, we supply addresses CH, DH, EH. Therefore, accessing any routine requires three steps:

1. **Knowing the starting address of the routine**
2. **Stepping through the routine addresses**
3. **Applying the addresses to the control ROM.**

Address ROM

The address ROM contains the starting addresses. The starting address of the LDA routine is 0011; the starting address of the ADD routine is 0110 and so on. When the op-code bits $I_7I_6I_5I_4$ drive the address ROM, the starting address is generated. For instance, if the ADD instruction is being executed, $I_7I_6I_5I_4$ is 0001.This is the input to the address ROM, the output of the ROM is 0110.

Presettable Counter

When computer run begins, the counter output is 0000 during T1 state, 0001 during T2 state, and 0011 during T3 state. Every Fetch cycle is the same because 0000, 0001, and 0010 come out of the counter during states T1, T2 and T3The op code in the IR controls the execution cycle. If an ADD instruction has been fetched, the $I_7I_6I_5I_4$ bits are 0001. These opcode bits drive the address ROM, producing an output of 0110.This starting address is the input to the presettable counter. When T3 is high the negative clock edge loads 0110 into the presettable counter. The counter is now preset, and

counting can resume at the starting address of the ADD routine. The counter output is 0110 during T4 state, 0111 during T5 state and 1000 during T6 state.

Address	Contents	Routine
0000	0011	LDA
0001	0110	ADD
0010	1001	SUB
0011	XXXX	None
0100	XXXX	None
0101	XXXX	None
0110	XXXX	None
0111	XXXX	None
1000	XXXX	None
1001	XXXX	None
1010	XXXX	None
1011	XXXX	None
1100	XXXX	None
1101	XXXX	None
1110	1100	OUT
1111	XXXX	None

Figure: Address ROM

Address	Contents†	Routine	Active
0H	5E3H	Fetch	$E_P, \overline{L}_M$
1H	BE3H		C_P
2H	263H		$\overline{CE}, \overline{L}_I$
3H	1A3H	LDA	$\overline{L}_M, \overline{E}_I$
4H	2C3H		$\overline{CE}, \overline{L}_A$
5H	3E3H		None
6H	1A3H	ADD	$\overline{L}_M, \overline{E}_I$
7H	2E1H		$\overline{CE}, \overline{L}_B$
8H	3C7H		$\overline{L}_A, E_U$
9H	1A3H	SUB	$\overline{L}_M, \overline{E}_I$
AH	2E1H		$\overline{CE}, \overline{L}_B$
BH	3CFH		$\overline{L}_A, S_U, E_U$
CH	3F2H	OUT	$E_A, \overline{L}_O$
DH	3E3H		None
EH	3E3H		None
FH	X	X	Not used

† CON = $C_P E_P \overline{L}_M \overline{CE}$ $\overline{L}_I \overline{E}_I \overline{L}_A E_A$ $S_U E_U \overline{L}_B \overline{L}_O$

Figure: Control ROM

Control ROM

The control ROM stores the SAP-1 microinstructions .During the fetch cycle; it receives addresses 0000, 0001, 0010. Therefore its outputs are

5E3H //0101 1110 0011

BE3H //1011 1110 0011

263H //0010 0110 0011

These control bits is sent to different components of computer during T1, T2 and T3 state respectively.

SAP 2 (SIMPLE AS POSSIBLE 2)

- **Bidirectional registers**
- **Includes jump instructions**
- **All register outputs to W bus are three-state; those not connected to the bus are two-state**

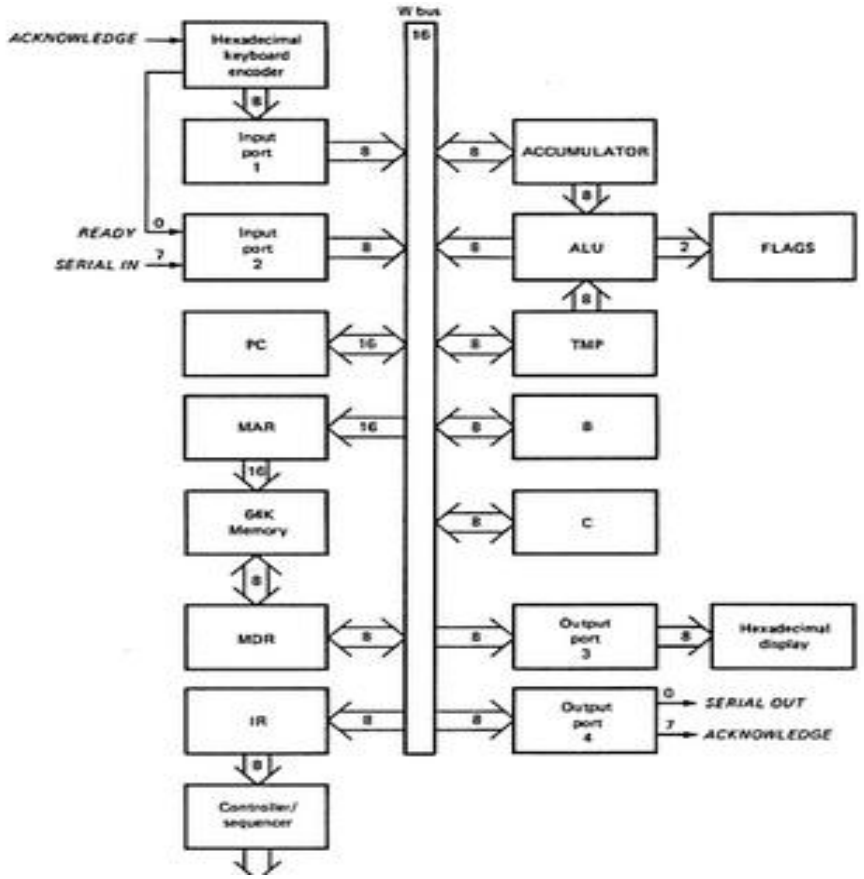


Fig: Block diagram of SAP 2 architecture

Input Ports

- **2 input ports numbered 1 and 2**
- **Hexadecimal keyboard encoder connected to port 1, sends READY signal to bit 0 of port 2**
- **This signal indicates when the data in port 1 is valid, the SERIAL IN, signal going to pin 7 or port 2.**

Program Counter

- **Program counter has 16 bits; therefore it can count from PC= 0000 0000 0000 000 to PC=1111 1111 1111 1111**
- **Its job is to send to the memory the address of the next instructions to be fetched and executed**
- **A low resets the PC before each computer run; so the data processing starts with the instruction stored in memory location 0000H**

MAR and Memory

- **During the fetch cycle, the MAR receives 16 bit addresses from PC. The two state MAR output then addresses memory location.**
- **The memory has 2K ROM with address 0000H too 07FFH. The ROM contains a program called monitor that initializes the computer on power-up, interprets the keyboard inputs and so on.**
- **The rest of memory is 64 K RAM with addresses from 0800H to FFFFH.**

MDR(Memory Data Register)

- **8 bit buffer register**
- **Its output sets up the RAM**
- **Receives data from the bus before a write operation and it sends data to the bus after a read operation**

IR(Instruction Register)

- **Part of control unit**

- **Memory read operation performed by computer to fetch an instruction from memory; this places the contents of the addressed memory location on W bus**
- **Contents split into 2 nibbles**
- **SAP 2 use 8 bits for op code which can accommodate 256 instructions**

Controller Sequencer

- **Produces the control words or microinstructions that coordinate and direct the rest of the computer**
- **Has more hardware since SAP 2 has bigger instruction set**
- **Microinstruction determines how the registers react to the next positive clock edge**

Accumulator

- **A buffer register that stores intermediate answers during the computer run**
- **Has two outputs two-state and three-state**
- **Two-state output goes to ALU and three-state to W bus**
- **Hence the 8 bit word in the accumulator continuously drives the ALU, but this same word appears on the bus only when E_A is active**

ALU and Flags

- **Includes arithmetic and logic operations**
- **Has 4 or more control bits that determine the arithmetic or logic operation performed on words A and B**

- **Flag is a flip-flop that keeps track of a changing condition during a computer run**
- **SAP-2 has two flags; sign and zero flag**

TMP, B and C Registers

- **TMP(temporary) register is used instead of register B to hold data being added or subtracted; which allows us more freedom in using the B register**
- **Besides B and TMP SAP 2 has a register C which gives us more flexibility in moving data during the computer run**

Output Ports

- **Two output ports numbered 3 and 4**
- **Contents of the accumulator can be loaded into port 3, which drives a hexadecimal display**
- **The contents can also be seen through port 4**

Architectural difference between SAP 1 and SAP 2 architecture

Major differences:

- **Bidirectional registers**
- **Flags**
- **Large instruction set having jump, call & loop.**
- **Standard Input Output devices in SAP 2.**
- **Large no. of processor registers in SAP 2.**

- **High memory capacity(64K) in SAP 2.**
- **Address bus 16 bit in SAP 2.**
- **Separate MAR & MDR in SAP 2.**
- **Complex programming possible in SAP 2**
- **Introduction of ALU in SAP 2**
- **Serial as well as parallel I/O in SAP 2.**

Flag

- **Represent the status of the arithmetic and logical operation.**
- **flip-flop concepts are used in flag to represent the status of arithmetic and logical operation.**
- **Two types of flag are there sign flag and zero flag.**

  1)Sign flag: gets set when the accumulator content is negative.

  2)Zero flag : gets set when the accumulator content is negative.

Instruction Set

Forewords:

- **The basic set of commands, or instructions, that a microprocessor understands.**
- **Also called the "Command Set".**
- **Deals with programming, including the native data types, instructions, memory architecture and external I/O Devices.**
- **Step-by-step processes that are loaded into the memory before the start of a computer run.**
- **It is essential to understand the Instruction Set before programming a computer.**
- **Includes a specification of the set of opcodes (machine language), and the native commands implemented by a particular processor.**

The SAP-2 Instruction Set:

MEMORY REFERENCE INSTRUCTIONS

1. **LDA and STA :**

LDA(Load to Accumulator):

- **Loads the accumulator with the content of memory location.**
- **More memory location can be accessed in SAP-2 than SAP-1 because the addresses are from 0000H to FFFFH.**
- **E.g. LDA 2000H: loads the accumulator with content of memory location 2000H**

STA(Store the Accumulator):

- **Stores the accumulator contents at different memory locations.**
- **E.g. STA 7FFFH: To store the accumulator contents at memory location 7FFFH. i.e. If A=8AH, STA 7FFFH stores 8AH at address 7FFFH.**

2. **MVI (Move Immediate):**

- **Loads a designated register with the byte that immediately follows the op code.**
- **E.g. MVI A,37H loads accumulator with 37H and binary content of A becomes A=0011 0111**

REGISTER INSTRUCTIONS

1. **MOV (Move):**

Moves data from one register to another register.

E.g. MOV A,B: Copies data from B to A

2. **ADD and SUB:**

Adds the content of assigned register to accumulator.

E.g. If A=04H and B=02H, ADD B gives result in A=06H

Subtract the content of the assigned register from the accumulator.

3. **INR and DCR (Increment and Decrement):**

Increases and decreases the register.

E.g. If B=56H and C=8AH then

INR B gives B=57H

DCR C gives C=89H

JUMP and CALL INSTRUCTIONS

1. **JMP (Jump):**

- **Tells the computer to get the next instruction from the designated memory location.**
- **E.g. JMP 3000H gives next instruction from memory location 3000H.**
- **2.JM (Jump if Minus):**
- **Jump to designated address if and only if the sign flag is set.**
- **E.g. Let JM 3000H is stored at 2005H. After this instruction is fetched, PC=3000H is executed if S=1 else (S=0) instruction is fetched from 2006H.**

3. **JZ (Jump if Zero):**

- **Tells the computer to jump to designated address only if the zero flag is set.**

- **E.g. JZ=3000H is stored at 2005H. Next instruction is fetched from 3000H only if Z=1 else from 2006.**

4. JNZ (Jump if Not Zero):

We get a jump when the zero flag is clear and no jump when it is set.

(Vice-versa of JZ)

5. CALL and RET:

CALL the Subroutine :

E.g. CALL 5000H will jump to the square root subroutine and CALL 6000H produces a jump to the logarithm subroutine.

RETURN:

Used at the end of every subroutine to tell the computer to go back to the original program.

LOGIC INSTRUCTIONS:

1. CMA (Complement the Accumulator):

Inverts each bit in the accumulator, producing a 1's complement.

2. ANA (AND the Accumulator):

ANDs the content of accumulator with the content of specified register.

E.g. A = 1100 1100, B = 1111 0001

ANA B results A = 1100 0000

3. ORA (OR the Accumulator):

ORs the content of accumulator with the content of specified register.

E.g. A = 1111 1101, B = 1111 0001

ORA B results A = 1111 1101

4. **ANI (AND Immediate):**

ANDs the accumulator content with the byte that immediately follows the op code.

E.g. A = 0101 1110 and ANI C7H gives A = 0100 0110 since C7= 1100 0111.

5. **ORI (OR immediate):**

The accumulator contents are ORed with the byte that follows the op code.

6. **XRI (XOR immediate):**

A = 0001 1100

XRI D4H will XOR 0001 1100 and 1101 0100 to produce A = 1100 1000 (High only if different inputs else low if same inputs)

OTHER INSTRUCTIONS

1. **NOP (No Operation):**

Do nothing. (Used to waste time)

By repeating NOP a number of times, we can delay the data processing, which is useful in timing operations.

2. **HLT (Halt):**

Ends the data processing.

3. **RAL (Rotate the accumulator Left):**

Shift all bits to the left and move the MSB to the LSB position.

E.g. A = 1011 0100

Then, RAL will produce A = 0110 1001

4. RAR (Rotate the accumulator Right):

The bits shift to the right, the LSB going to the MSB position.

E.g. A = 1011 0100

Then, RAR results A = 0101 1010

5. IN (Input):

Tells a computer to transfer data from the designated port to the accumulator.

E.g. IN 02H means to transfer the data in port 2 to the accumulator.

6. OUT *(Output):*

When this instruction is executed, the accumulator word is loaded into the designated output port.

E.g. OUT 03H will transfer the contents of the accumulator to port 3.

UNIT 3
INSTRUCTION CYCLE

INTRODUCTION

Instruction cycle: The necessary steps that the cpu carries out to fetch an instruction and necessary data from the memory and to execute it constitute and instruction cycle it is defined as the time required to complete the execution of an instruction.

An instruction cycle consists of fetch cycle and execute cycle. In fetch cycle CPU fetches upcode from the memory . The necessary steps which are carried out to fetch an upcode from memory constitute a fetch cycle. The necessary steps which are carried out to get data if

any from the memory and to perform the specific operation specified in an instruction constitute and execute cycle . The total time required to execute an instruction given by IC = Fc+ Ec

The 8085 consists of 1-6 machine cycles or operations.

Fetch cycle: The first byte of an instruction is it's upcode . The program counter keeps the memory address of the next instruction to be executed in the beginning of fetch cycle the content of the program counter ,which is the address of the memory location where upcode is

available , is send to the memory. The memory places the upcode on the data bus so as to transfer it to CPU. The entire process takes 3 clock cycle.

Execute cycle/Operation: The upcode fetched from the memory goes to IR from the IR it goes to the decoder which decodes instruction. After the instruction is decoded execution begins.

- **If the operand is in general purpose register, execution is performed immediately. I,e in one clock cycle.**
- **If an instruction is contains data or operand address, then CPU has to perform some read operations to get the desired data.**
- **In some instruction write operation is performed. In write cycle data are sent from the CPU to the memory of an o/p device.**
- **In some cases execute cycle may involve one or more read or write cycle or both.**

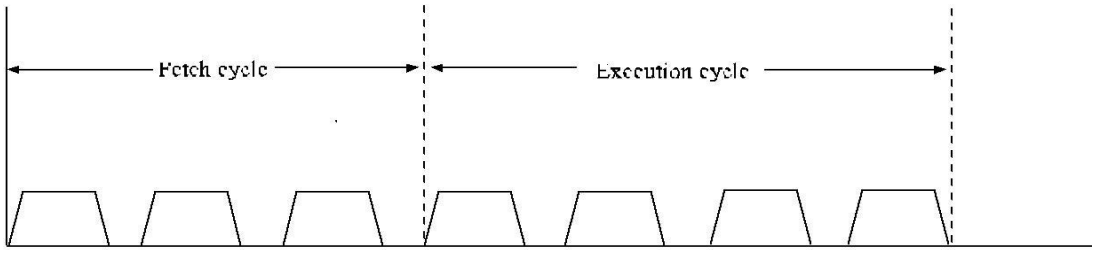


Figure1: Instruction Cycle

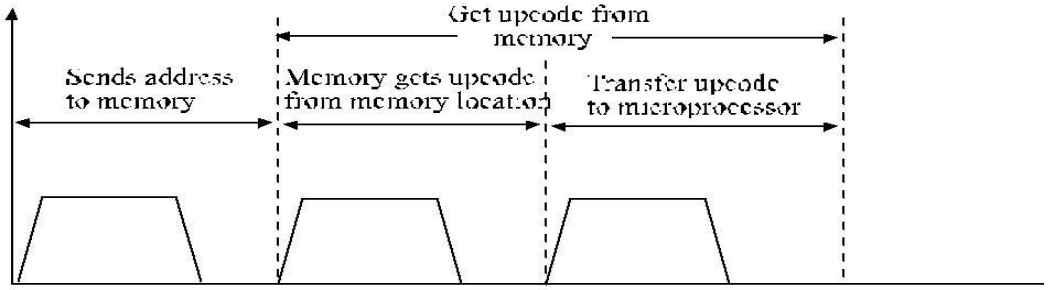
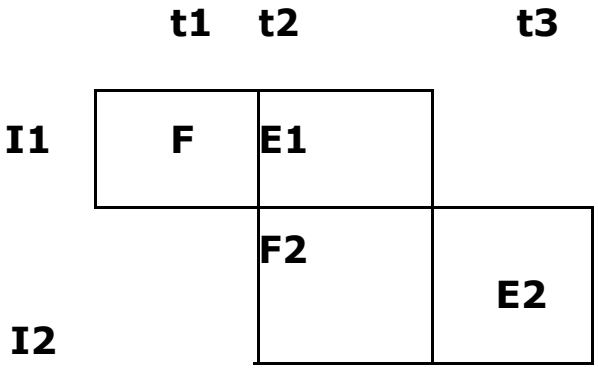


Figure2: Fetch Cycle

Fetch execution overlap:



Machine cycle: It is defined as the time required to complete one operation of accessing memory , i/p, o/p or acknowledging and external request. This cycle may consists of 3 to 6 T states.

T-states: It is defined as one sub division of the operation performed in one clock period. These sub division are internal states synchronized with system

clock and each T states precisely equal to one clock period.

Timing diagram: The necessary steps which are carried in a machine cycle can represented graphically. Such graphical representation is called timing diagram.

Opcode Fetch

A microprocessor either reads or writes to the memory or I/O devices. The time taken to read or write for any instruction must be known in terms of the μP clock. The 1st step in communicating between the microprocessor and memory is reading from the memory. This reading process is called

opcode fetch: The process of opcode fetch operation requires minimum 4- clock cycles T1, T2, T3, and T4 and is the 1st machine cycle (M1) of every instruction.

In order to differentiate between the data byte pertaining to an opcode or an address, the machine cycle takes help of the status signal IO/M, S1, and S0. The IO/M = 0 indicates memory operation and S1 = S0 = 1 indicates Opcode fetch operation. The opcode fetch machine cycle M1 consists of 4-states (T1, T2, T3, and T4). The 1st 3-states are used for fetching (transferring) the byte from the memory and

the 4th-state is used to decode it. Thus, thorough understanding about the communication between memory and microprocessor can be achieved only after knowing the processes involved in reading or writing into the memory by the microprocessor and time taken w.r.t. its clock period. This can be explained by examples. The process of implementation of each instruction follows the fetch and execute cycles. In other words, first the instruction is fetched from memory and then executed. Figure 3 and 4 depict these 2-steps for implementation of the instruction ADI 05H. Let us assume that the accumulator contains the result of previous operation i.e., 03H and instruction is held at memory locations 2030H and 2031H.

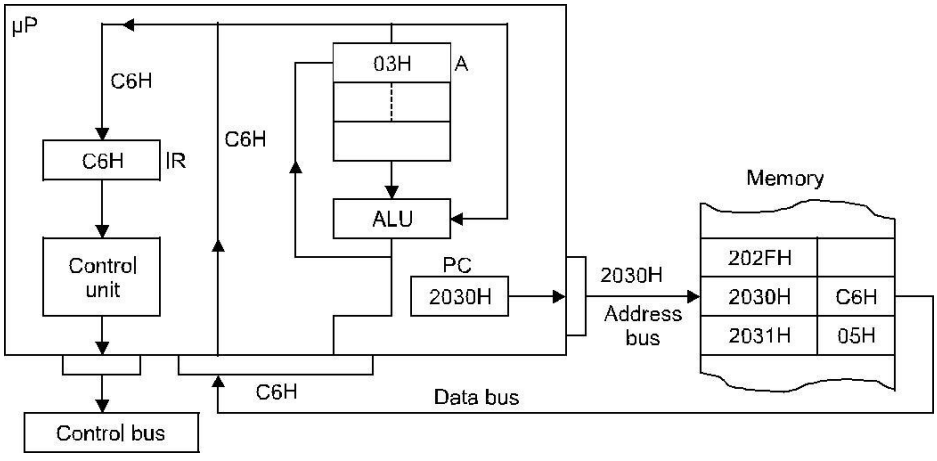


Figure3: Instruction fetch : reads 1st byte (Opcode) in instruction register (IR)

The fetch part of the instruction is the same for every instruction. The control unit puts the contents of the program counter (PC) 2030H on the address bus. The 1st byte (opcode C6H in this example) is passed to the instruction register. In the execute cycle of the instruction, the control unit examines the opcode and as per interpretation further memory read or write operations are performed depending upon whether additional information/ data are required or not. In this case, the data 05H from the memory is transferred through the data bus to the ALU. At the same time the control unit sends the contents of the accumulator (03H) to the ALU and performs the addition operation. The result of the addition operation 08H is passed to the accumulator overriding the previous contents 03H. On the completion of one instruction, the program counter is automatically incremented to point to the next memory location to execute the subsequent instruction.

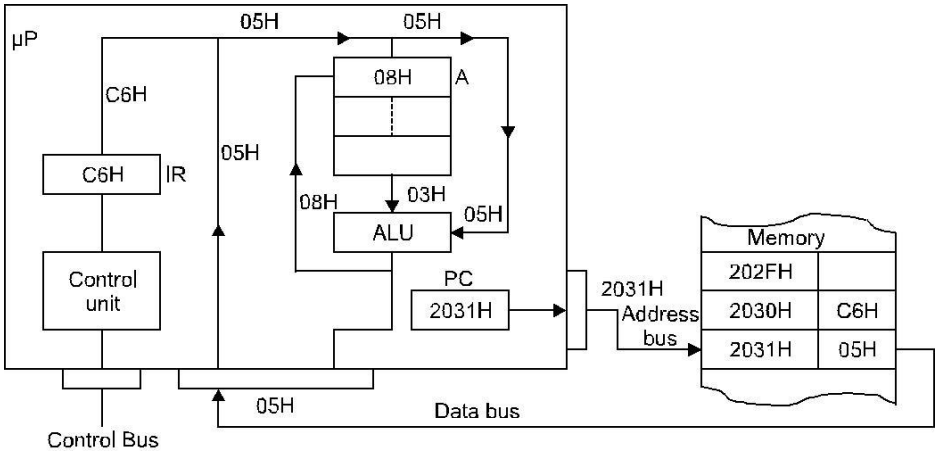


Figure4: Instruction execute : reads 2nd byte from memory and adds to accumulator

Note : The slope of the edges of the clock pulses has been shown to be much exaggerated to indicate the existence of rise and fall time.

TIMING DIAGRAM OF OPCODE FETCH

The process of opcode fetch operation requires minimum 4-clock cycles T1, T2, T3, and T4 and is the 1st machine cycle (M1) of every instruction.

Example

Fetch a byte 41H stored at memory location 2105H.

For fetching a byte, the microprocessor must find out the memory location where it is stored.

Then provide condition (control) for data flow from memory to the microprocessor. The process of data flow and timing diagram of fetch operation are shown in figures. TheμP fetches opcode of the instruction from the memory as per the sequence below

- **A low IO/M means microprocessor wants to communicate with memory.**
- **The μP sends a high on status signal S1 and S0 indicating fetch operation.**
- **The μP sends 16-bit address. AD bus has address in 1st clock of the 1st machine cycle,T1.**

- **AD7 to AD0 address is latched in the external latch when ALE = 1.**
- **AD bus now can carry data.**
- **In T2, the RD control signal becomes low to enable the memory for read operation.**
- **The memory places opcode on the AD bus**
- **The data is placed in the data register (DR) and then it is transferred to IR.**

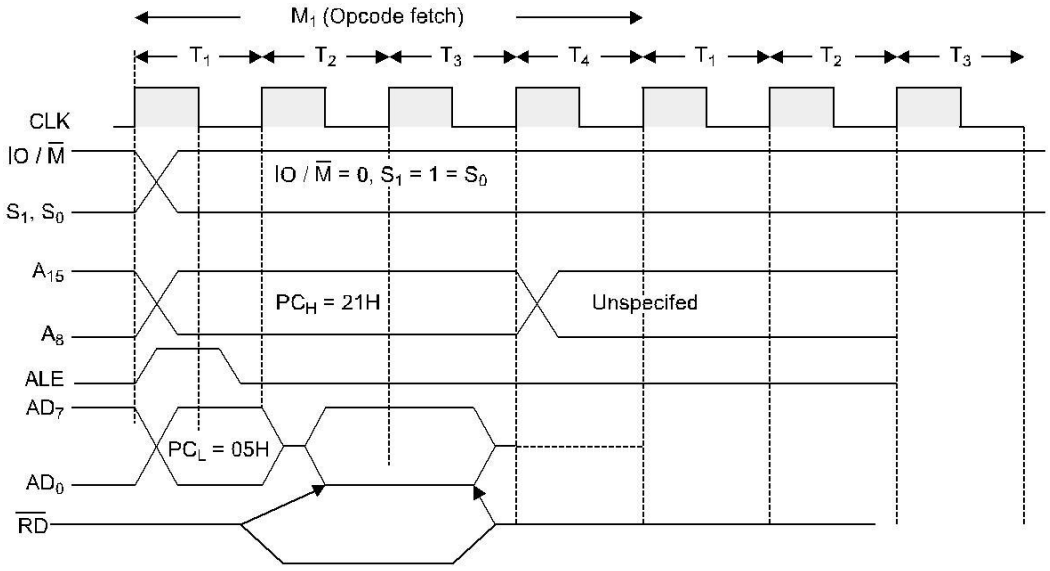


Figure5: Opcode fetch

- **During T3 the RD signal becomes high and memory is disabled.**
- **During T4 the opcode is sent for decoding and decoded in T4.**
- **The execution is also completed in T4 if the instruction is single byte.**
- **More machine cycles are essential for 2- or 3-byte instructions. The 1st machine cycle M1 is meant for fetching the opcode. The machine cycles M2 and M3 are required either to read/ write data or address from the memory or I/O devices.**

Example

- **Opcode fetch MOV B,C.**
- **T1 : The 1st clock of 1st machine cycle (M1) makes ALE high indicating address latch enabled which loads low-order address 00H on AD7 _ AD0 and high-order address 10H simultaneously on A15 _ A8. The address 00H is latched in T1.**
- **T2 : During T2 clock, the microprocessor issues RD control signal to enable the memory and memory places 41H from 1000H location on the data bus.**

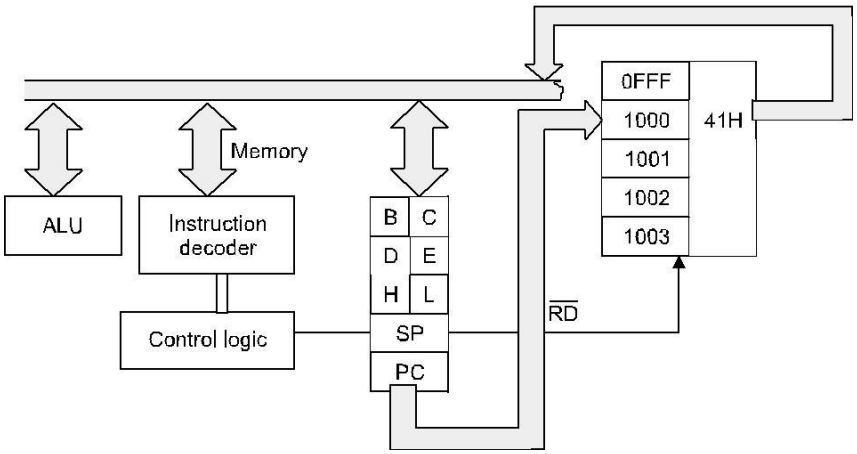


Figure6: Data flow from memory to microprocessor

- **T3 : During T3, the 41H is placed in the instruction register and RD = 1 (high) disables signal.It means the memory is disabled in T3 clock cycle. The opcode cycle is completed by end of T3 clock cycle.**

- **T4 : The opcode is decoded in T4 clock and the action as per 41H is taken accordingly. In other word, the content of C-register is copied in B-register. Execution time for opcode 41H is**

Clock frequency of 8085 = 3.125 MHz

Time (T) for one clock = 1/3.125 MHz = 325.5 ns = 0.32 µS

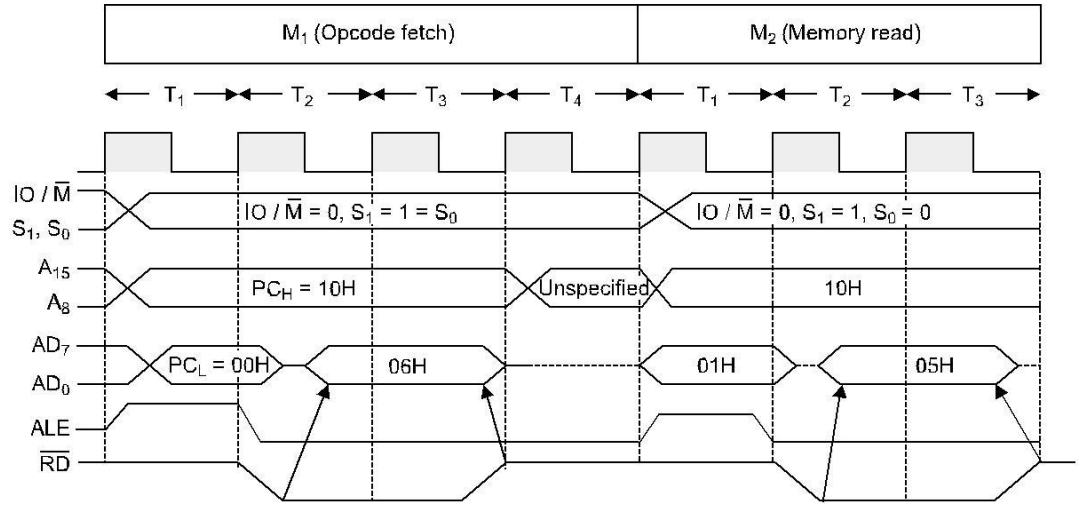
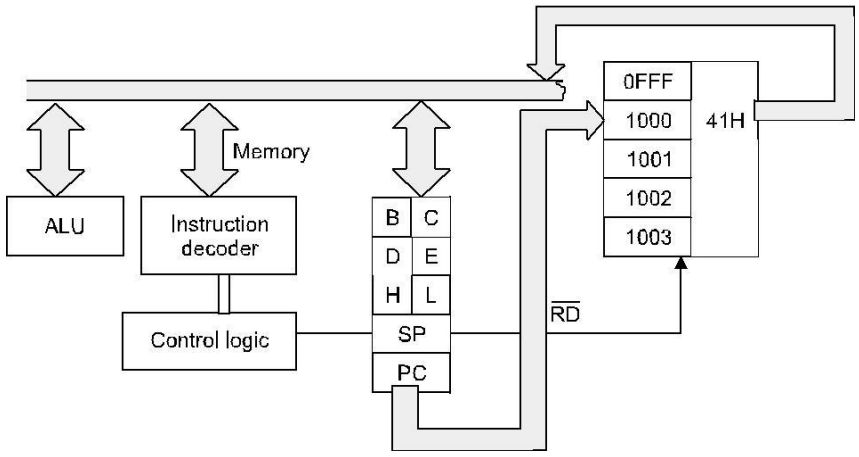


Figure7: Timing diagram for MVI B,05H



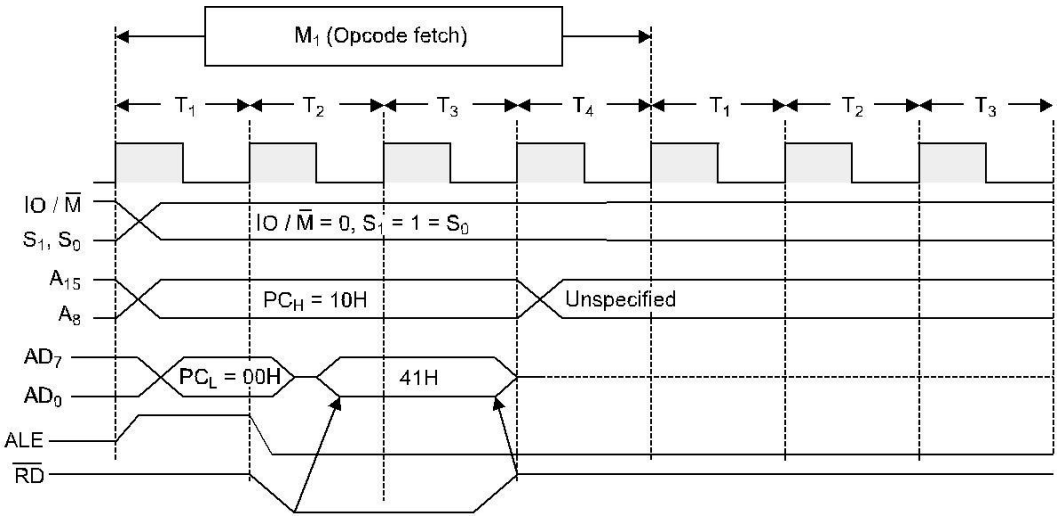


Figure8: Opcode fetch (MOV B,C)

The MVI B,05H instruction requires 2-machine cycles (M1 and M2). M1 requires 4-states and M2 requires 3-states, total of 7-states as shown in Fig. 5.3 (*d*). Status signals IO/M, S1 and S0 specifies the 1st machine cycle as the op-code fetch.

In T1-state, the high order address {10H} is placed on the bus A15 □□A8 and low-order address {00H} on the bus AD7 □□AD0 and ALE = 1. In T2 - state, the RD line goes low, and the data 06H from memory location 1000H are placed on the data bus. The fetch cycle becomes complete in T3-state. The instruction is decoded in the T4-state. During T4-state, the contents of the bus are unknown. With the change in the status signal, IO/M = 0, S1 = 1 and S0 = 0, the 2nd machine cycle is identified as the memory read. The address is 1001H and the data byte[05H] is fetched via the data bus. Both M1 and M2 perform memory read operation, but the M1 is called op-code fetch i.e., the 1st machine cycle of each instruction is identified as the opcode fetch cycle. Execution time for MBI B,05H i.e., memory read machine ycle and instruction cycle is

Mnemonics	Machine	Memory
MVI B,05H	code	Location
	06H	1000H
	05H	1001H

Read Cycle

The high order address ($A_{15} \Leftrightarrow A_8$) and low order address ($AD_7 \Leftrightarrow AD_0$) are asserted on 1st low going transition of the clock pulse. The timing diagram for $_{IO/M}$ read are shown in Figures 9 & 10The $A_{15} \Leftrightarrow A_8$ remains valid in T_1, T_2, and T_3 i.e. duration of the bus cycle, but $AD_7 \Leftrightarrow AD_0$ remains valid only in T_1. Since it has to remain valid for the whole bus cycle, it must be saved for its use in the T_2 and T_3.

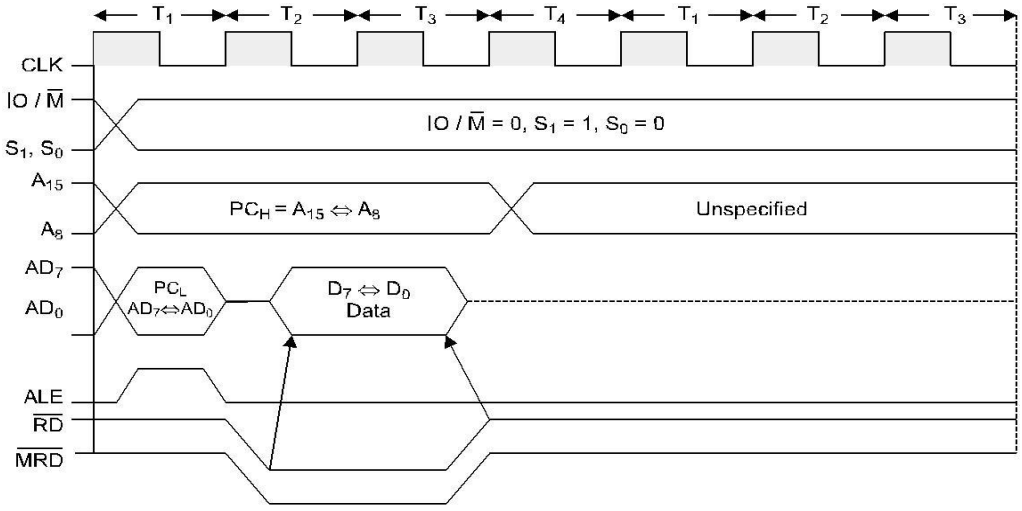


Figure9: Memory read timing diagram

ALE is asserted at the beginning of T_1 of each bus cycle and is negated towards the end of T_1. ALE is active during T_1 only and is used as the clock pulse to latch the address ($AD_7 \Leftrightarrow AD_0$) during T_1. The RD is asserted near the beginning of T_2. It ends at the end of T_3. As soon as the $_{RD}$ becomes active, it forces the memory or I/O port to assert data. $_{RD}$ becomes inactive towards the end of T_3, causing the port or memory to terminate the data.

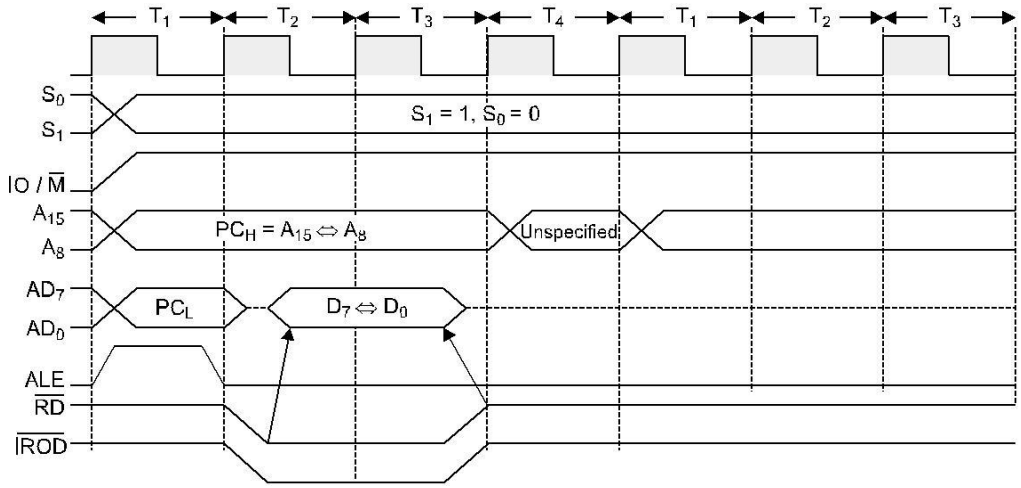


Figure10: I/O Read timing diagra

Write Cycle

Immediately after the termination of the low order address, at the beginning of the T_2, data is asserted on the address/data bus by the processor. WR control is activated near the start of T_2 and becomes inactive at the end of T_3. The processor maintains valid data until after WR is terminated. This ensures that the memory or port has valid data while WR is active.

It is clear from Figure11 & 12that for READ bus cycle, the data appears on the bus as a result of activating RD and for the WR bus cycle, the time the valid data is on the bus overlaps the time that the WR is active.

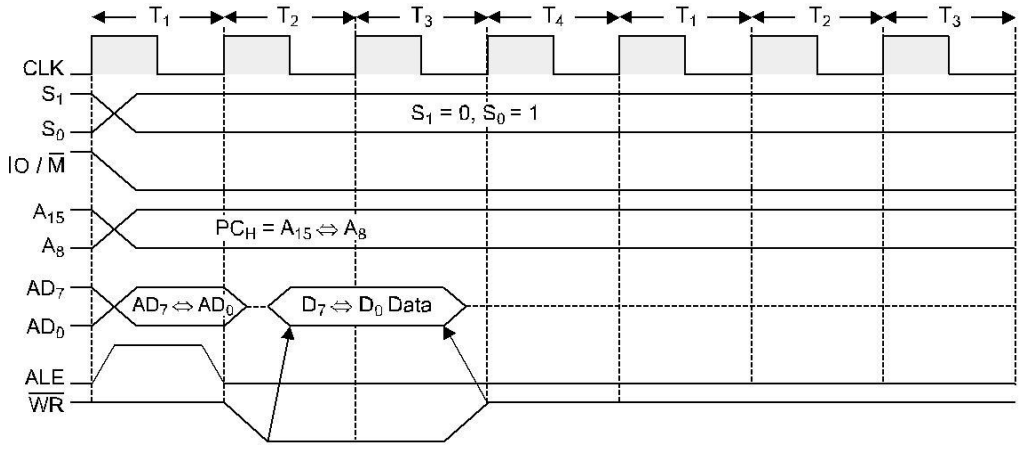


Figure11: Memory write timing diagram

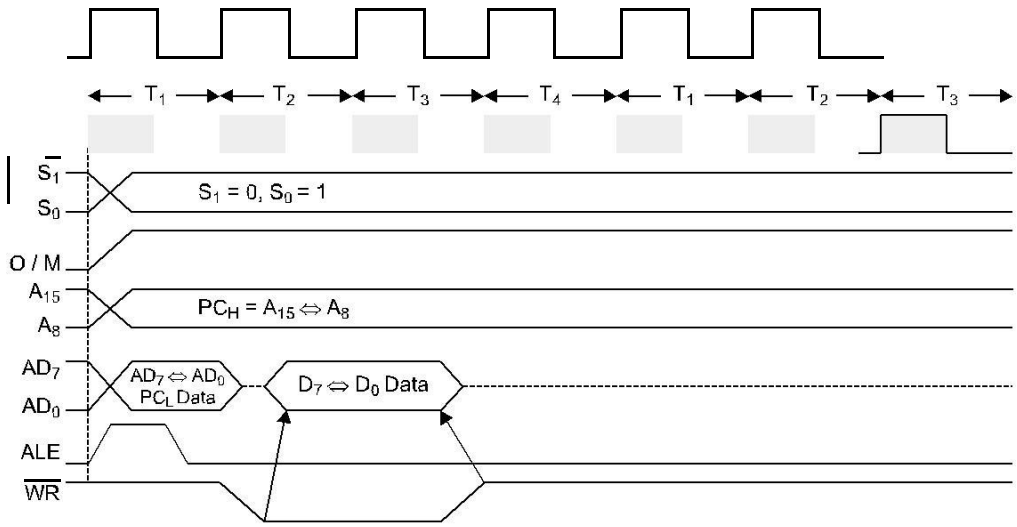


Figure12: I/O write timing diagram

STA

The STA instruction stands for storing the contents of the accumulator to a memory location whose address is immediately available after the instruction (STA). The 8085 have 16-address lines, it can address 2^{16} = 64 K. Since the STA instruction is meant to store the contents of the accumulator to the memory location, it is a 3-byte instruction. 1st byte is the opcode, the 2nd and 3rd bytes are the address of the memory locations. The storing of the STA instruction in the memory locations is as

Opcode 1st byte

Low address 2nd byte

High address 3rd byte

Three machine cycles are required to fetch this instruction : opcode Fetch transfers the opcode from the memory to the instruction register. The 2-byte address is then transferred, 1-byte at a time, from the memory to the temporary register. This requires two Memory read machine cycles. When the entire instruction is in the microprocessor, it is executed. The execution process transfers data from the microprocessor to the memory. The contents of the accumulator are transferred to memory, whose

address was previously transferred to the microprocessor by the preceding 2-Memory Read machine cycles. The address of the memory location to be written is generated as

Mnemonic	*Instruction Byte*	*Machine Cycle*	*T-states*
STA	Opcode LOW Address HIGH Address	Opcode Fetch Memory Read Memory Read Memory Write	4 3 3 3
			13

The high order address byte in the temporary register is transferred to the address latch and the low order address byte is transferred to the address/data latch. This data transfer is affected by

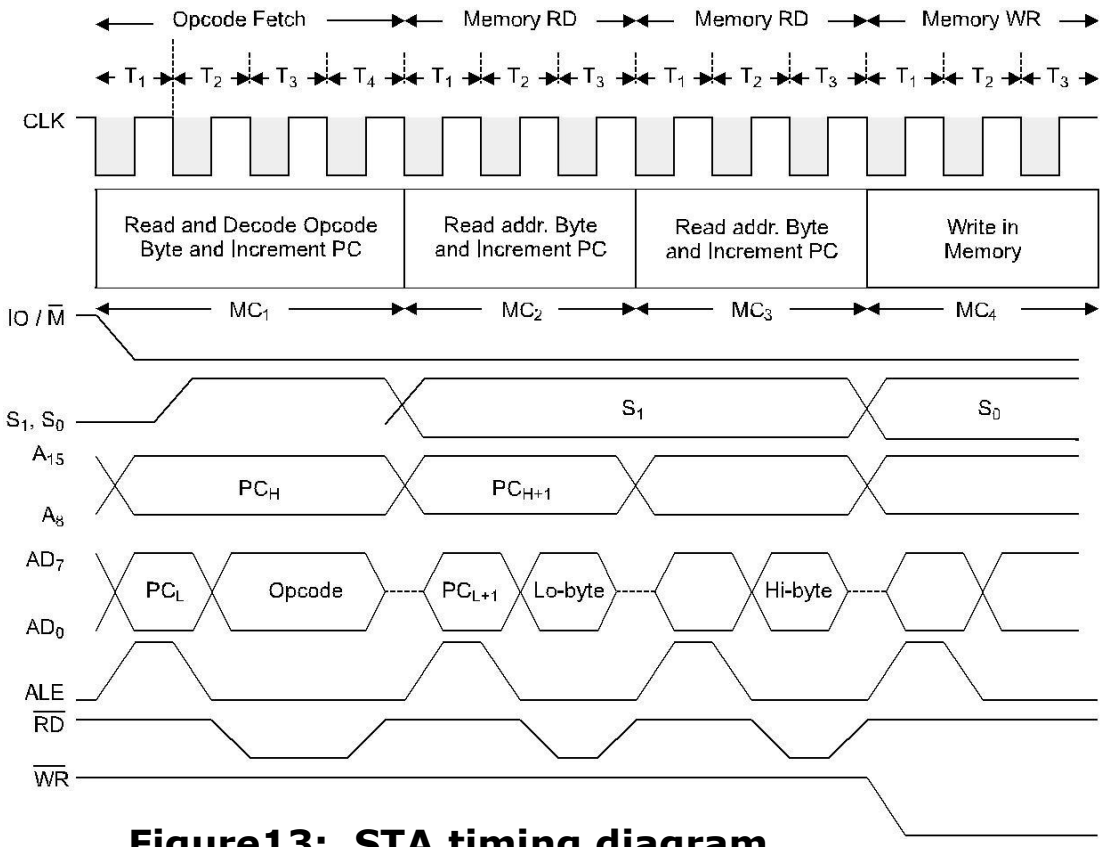


Figure13: STA timing diagram

Memory Write machine cycle. Thus 3-byte STA

instruction has four machine cycles in its instruction cycle.

The timing and control section of the microprocessor automatically generates the proper machine cycles required for an instruction cycle from the information provided by the opcode. The timing diagram of the instruction STA is shown in Fig. 5.3 (*i*). The status of IO / M , S_1 and S_0 for 4-machine cycles are obtained from Table 5.1. The condition of IO / M , S_1 and S_0 would be 0, 1 and 1 respectively in MC_1. The status of ALE is high at the beginning of 1st state of each machine cycle so that $AD_7 \Leftrightarrow AD_0$ work as the address bus. RD remains high during 1st state of each machine cycle, since during 1st state of each machine cycle $AD_7 \Leftrightarrow AD_0$ work as address bus. It remains high during 4th state of the 1st machine cycle also as the 4th state is used to decode the op code for generating the required control signals.

The opcode fetch of STA instruction has 4-states (clock cycles). Three states have been used to read the opcode from the main memory and the 4th to decode it and set up the subsequent machine cycle.

The action of memory read or write cycles containing 3-states *i.e.*, T_1, T_2, and T_3 are explained as:

T_1 : During this period the address and control signals for the memory access are set up.

T_2 : The µP checks up the READY and HOLD control lines. If READY = 0, indicating a slow memory device, the µP enters in the wait state until READY = 1, indicating DMA request, then only the µP floats the data transfer lines and enters into wait until HOLD = 0.

T_3 : In memory read cycles the µP transfers a byte from the data bus to an internal register and in memory write cycle the µP transfers a byte from an internal register to the data bus.

Thus STA instruction requires 4-machine cycles containing 13-states (clock cycles). With a typical clock of 3 MHz (= 330 ns), the STA instruction requires 13*330 ns = 4.29 ms for its execution.

General Control signal:

IO/M	S1	S0	RD	WR	Signals
0	1	0	0	1	MEMR
0	0	1	1	0	MEMW
1	1	0	0	1	IOR
1	0	1	1	0	IOW
0	1	1	0	1	OPCEDE FETCH

MOV A, B = 1MC – opcode fetch – 4 T state

MVI A, 32H = 2MC – opcode fetch, 1 memory read – 7 T states

STA 2050H- 4 MC- opcode fetch, 2 memory read , 1 memory write.

MOV A, M- 2 MC- 1 opcode fetch, 1 memroy write.

IN 84H – 3 MC – opcode fetch, 1 memory read, i/o read.

OUT 02H – 3 MC – opcode fetch, 1 memory read, 1 i/o write.

JMP 2050 – 3 MC – opcode fetch, 2 memory read. (when condition satisfied)

-2 MC- opcode fetch, 1 memory read. (When condition unsatisfied)

ADI 12H- 2MC- opcode feth, 1 memory read.

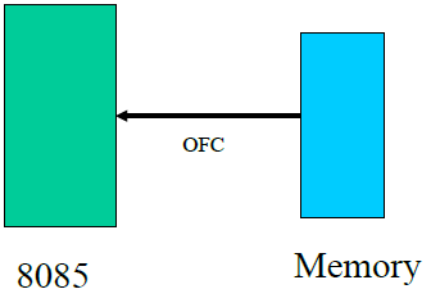
Timing Diagram

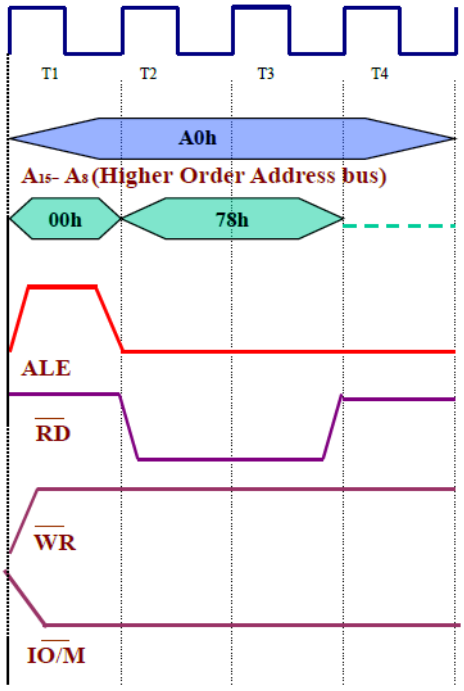
Instruction:

A000h MOV A,B

Corresponding Coding:

A000h 78





Op-code fetch Cycle

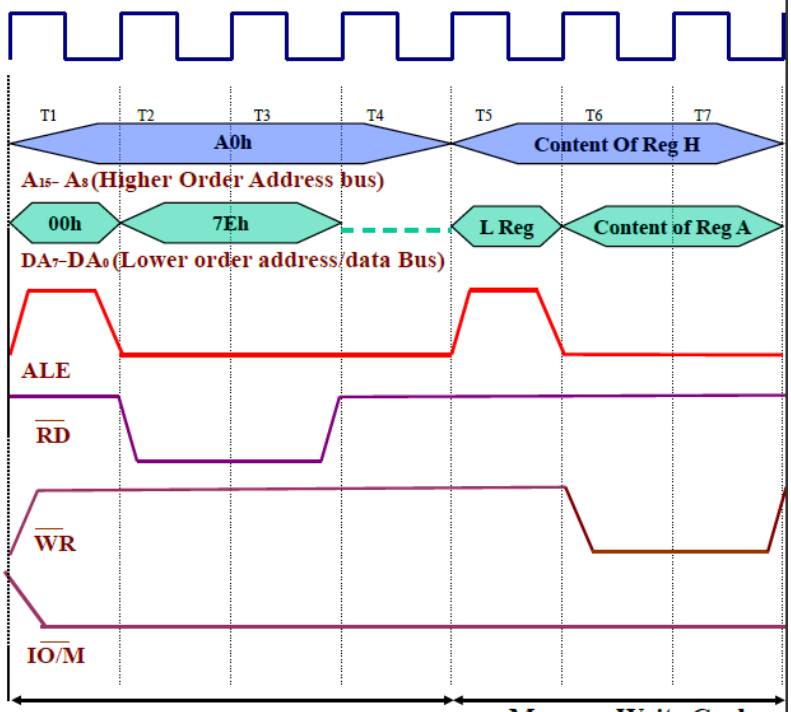
Timing Diagram

Instruction:

A000h MOV M,A

Corresponding Coding:

A000h 77



Timing Diagram

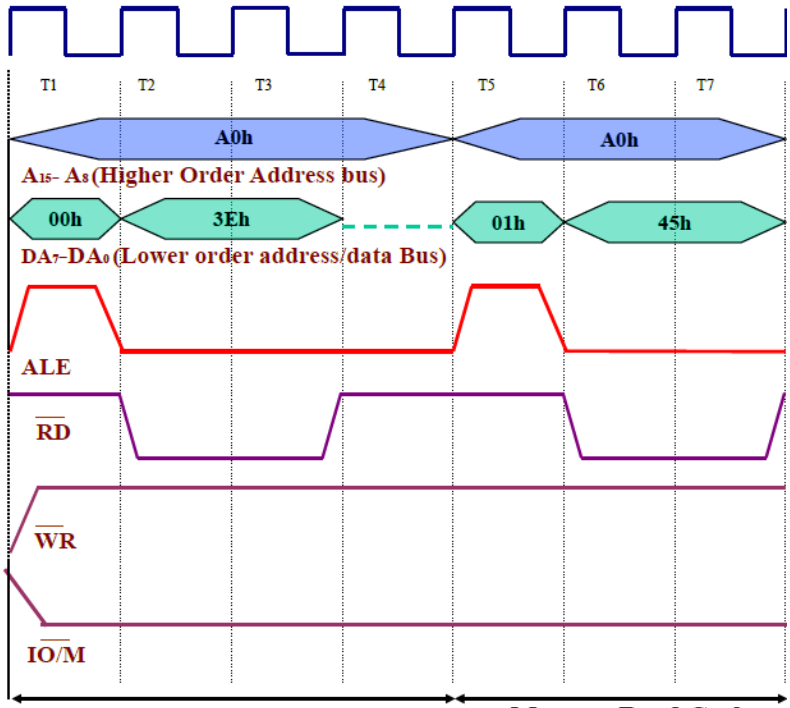
Instruction:

A000h MVI A,45h

Corresponding Coding:

A000h 3E

A001h 45



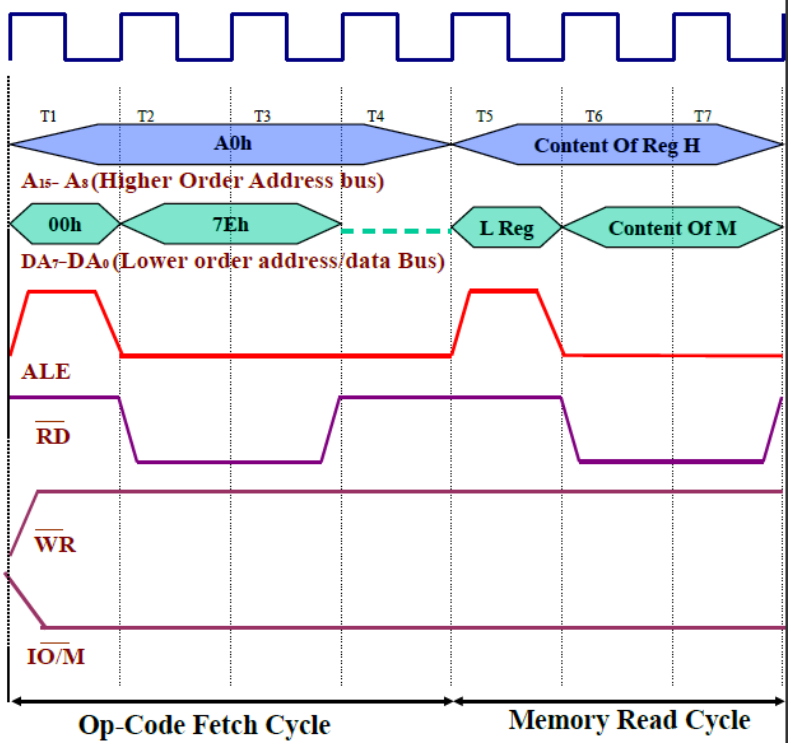
Instruction:

A000h MOV A,M

Corresponding Coding:

A000h 7E

Timing Diagram



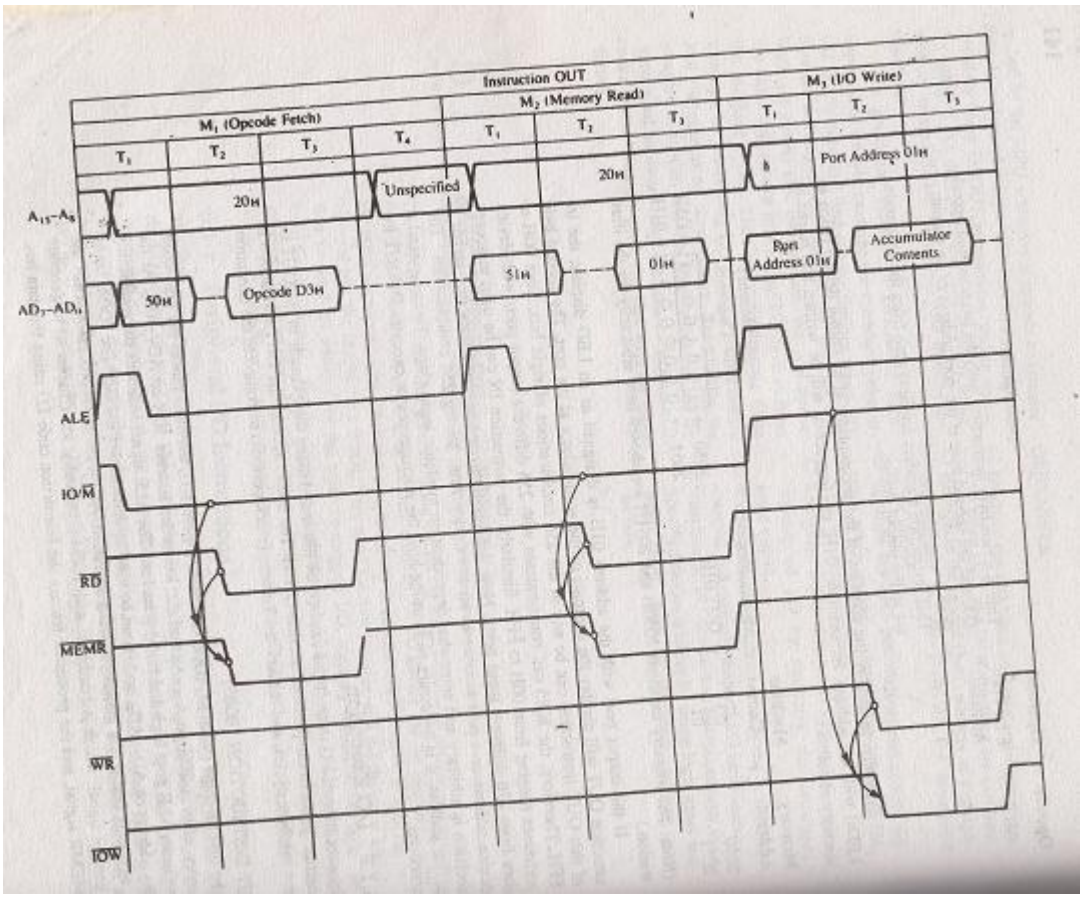


Figure14: Timing Diagram of OUT 01H

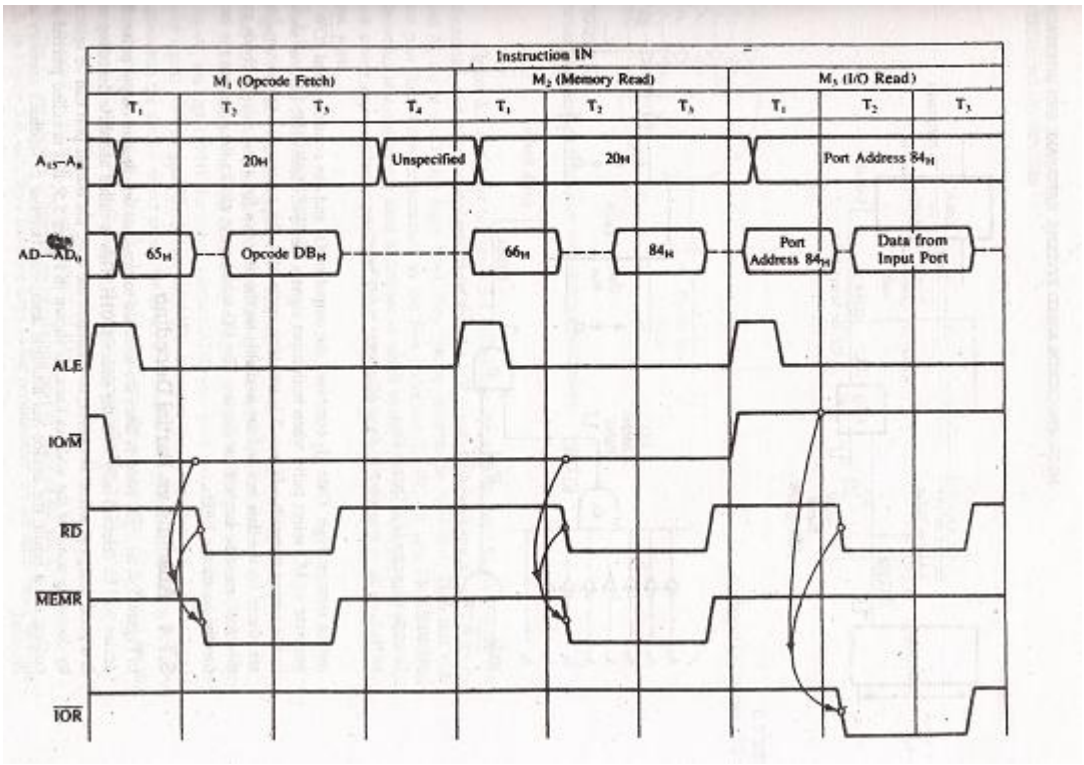


Figure15: Timing Diagram of IN 84H

UNIT 4

INTEL 8085/8086/8088 ARCHITECTURE

8085 MicroprocessorArchitecture

- **8-bit general purpose µp**
- **Capable of addressing 64 k of memory**
- **Has 40 pins**
- **Requires +5 v power supply**
- **Can operate with 3 MHz clock**
- **8085 upward compatible**

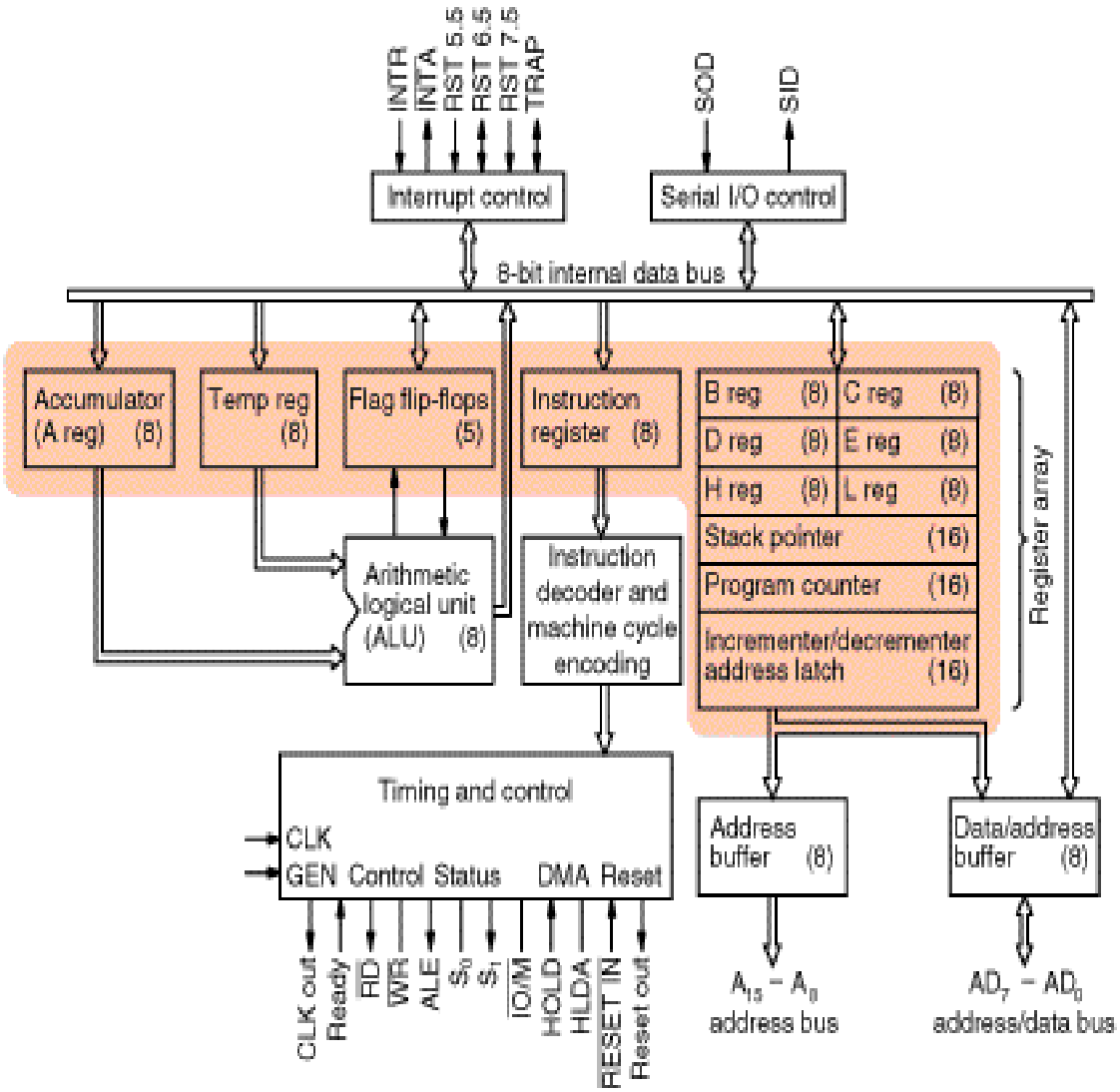
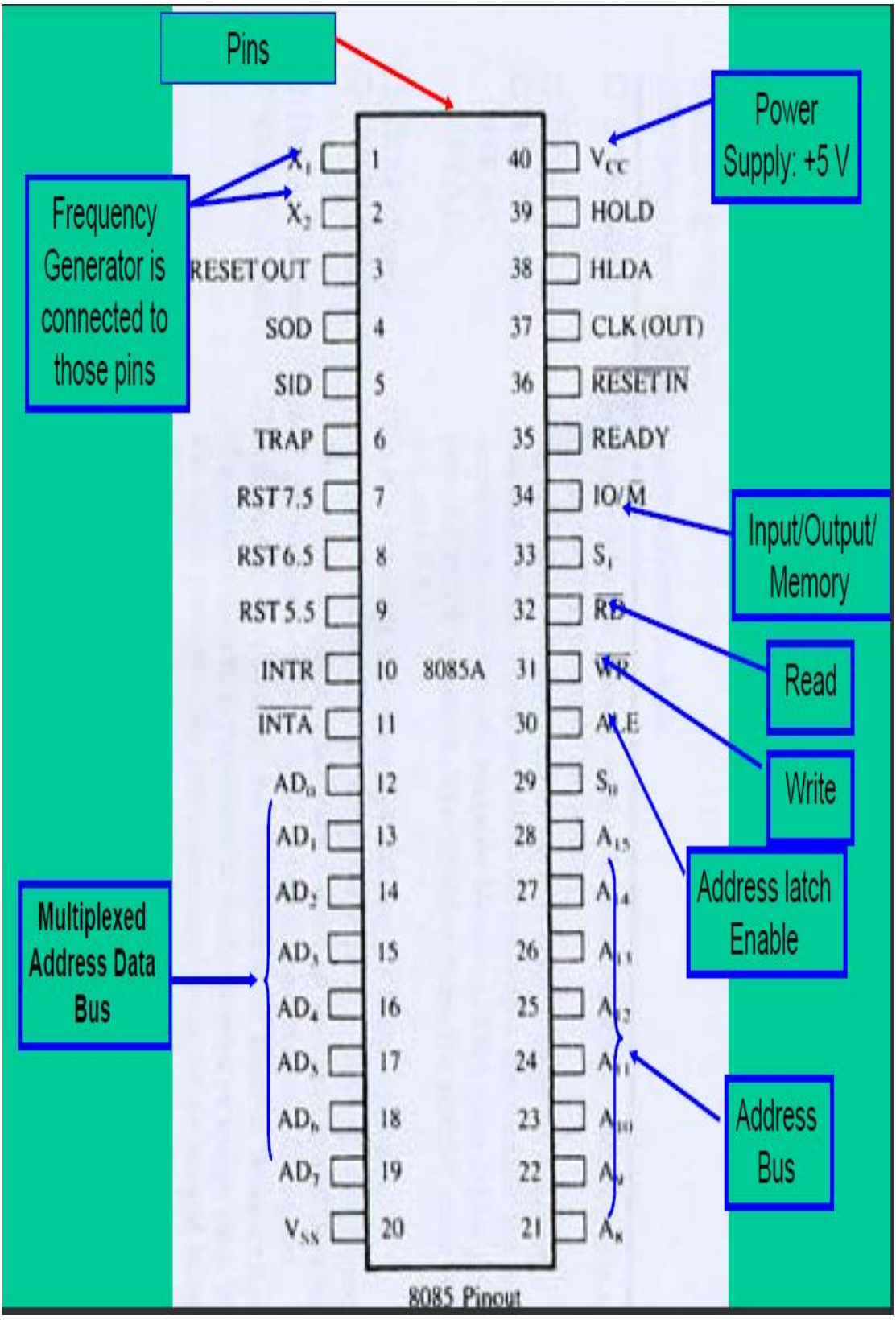
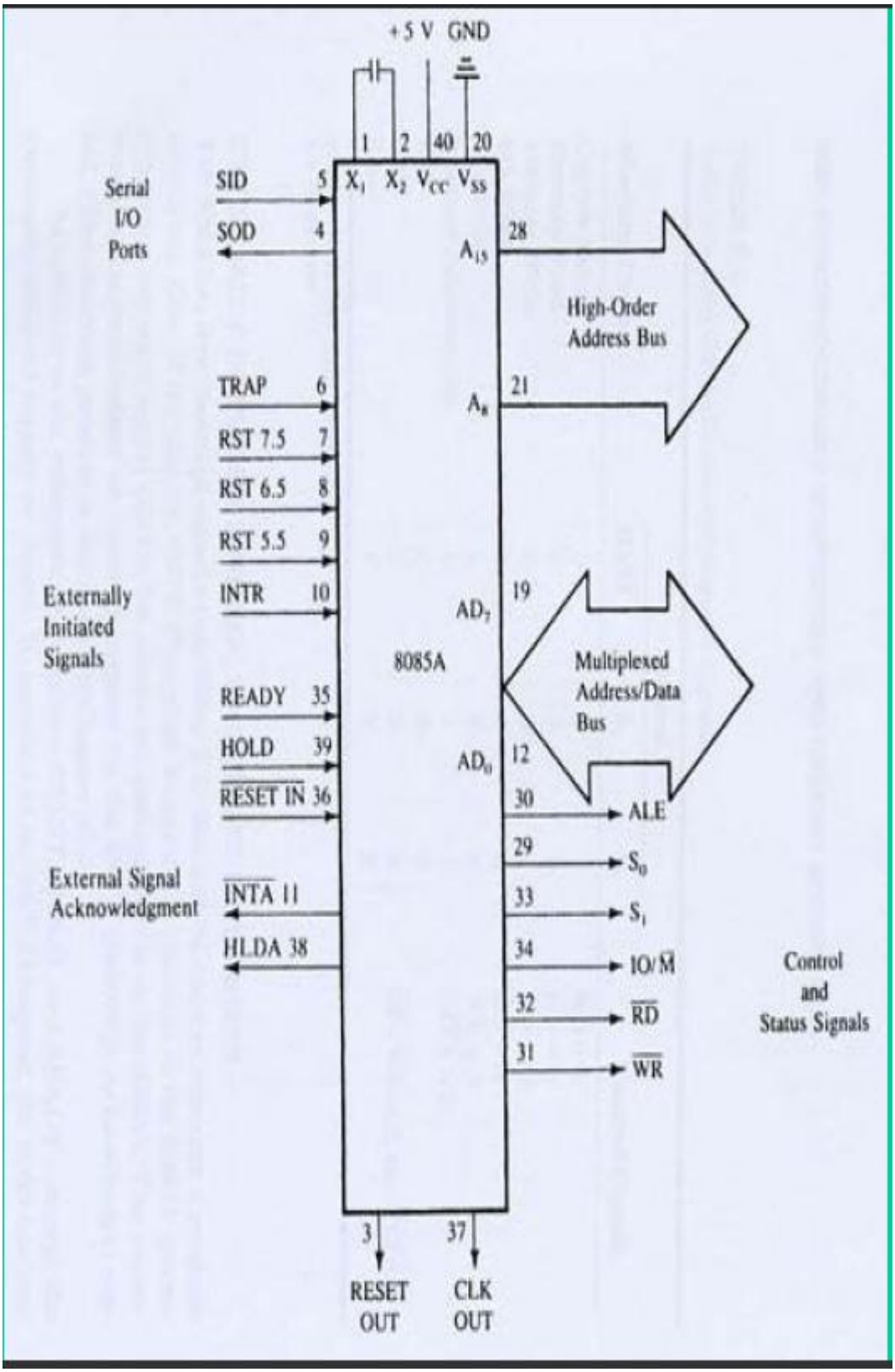


Figure: Architecture of Intel 8085 Microprocessor


Pins
Power Supply: +5 V
Frequency Generator is connected to those pins
Input/Output/ Memory
Read
Write
Address latch Enable
Multiplexed Address Data Bus
Address Bus
8085A
8085 Pinout


+5 V
GND
1
2
40
20
X1
X2
VCC
VSS
Serial I/O Ports
SID
5
SOD
4
A15
28
High-Order Address Bus
TRAP
6
RST 7.5
7
RST 6.5
8
RST 5.5
9
A8
21
Externally Initiated Signals
INTR
10
AD7
19
8085A
Multiplexed Address/Data Bus
READY
35
HOLD
39
AD0
12
RESET IN
36
30
ALE
29
S0
External Signal Acknowledgment
INTA
11
33
S1
HLDA
38
34
IO/M
Control and Status Signals
32
RD
31
WR
3
37
RESET OUT
CLK OUT

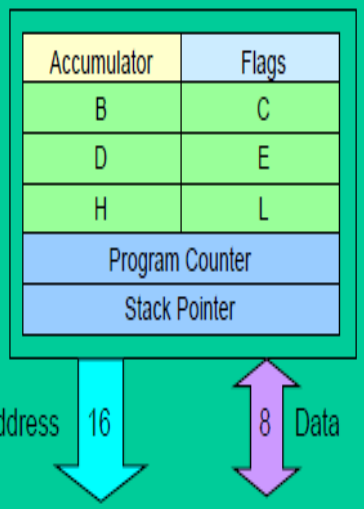
- System Bus – wires connecting memory & I/O to microprocessor
  - Address Bus
    - Unidirectional
    - Identifying peripheral or memory location
  - Data Bus
    - Bidirectional
    - Transferring data
  - Control Bus
    - Synchronization signals
    - Timing signals
    - Control signal

Intel 8085 Microprocessor consists of:

Control unit: control microprocessor operations.
ALU: performs data processing function.
Registers: provide storage internal to CPU.
Interrupts
Internal data bus
The ALU
• In addition to the arithmetic & logic circuits, the ALU includes the accumulator, which is part of every arithmetic & logic operation.

• Also, the ALU includes a temporary register used for holding data temporarily during the execution of the operation. This temporary register is not accessible by the programmer.

- Registers
  - General Purpose Registers
    - B, C, D, E, H & L (8 bit registers)
    - Can be used singly
    - Or can be used as 16 bit register pairs
      - BC, DE, HL
    - H & L can be used as a data pointer (holds memory address)
  - Special Purpose Registers
    - Accumulator (8 bit register)
      - Store 8 bit data
      - Store the result of an operation
      - Store 8 bit data during I/O transfer



- Flag Register
  - 8 bit register – shows the status of the microprocessor before/after an operation
  - S (sign flag), Z (zero flag), AC (auxillary carry flag), P (parity flag) & CY (carry flag)

D7	D6	D5	D4	D3	D2	D1	D0
S	Z	X	AC	X	P	X	CY

  - Sign Flag
    - Used for indicating the sign of the data in the accumulator
    - The sign flag is set if negative (1 – negative)
    - The sign flag is reset if positive (0 –positive)

- Zero Flag
  - Is set if result obtained after an operation is 0
  - Is set following an increment or decrement operation of that register

```
  10110011
+ 01001101
----------
1 00000000
```

- Carry Flag
  - Is set if there is a carry or borrow from arithmetic operation

```
    1011 0101
+   0110 1100
-------------
Carry 1 0010 0001
```

```
    1011 0101
-   1100 1100
-------------
Borrow 1 1110 1001
```

● Auxillary Carry Flag

–Is set if there is a carry out of bit 3

● Parity Flag

–Is set if parity is even

–Is cleared if parity is odd

The Internal Architecture

● We have already discussed the general purpose registers, the Accumulator, and the flags.

● The Program Counter (PC)

– This is a register that is used to control the sequencing of the execution of instructions.

– This register always holds the address of the next instruction.

– Since it holds an address, it must be 16 bits wide.

● The Stack pointer

–The stack pointer is also a 16-bit register that is used to point into memory.

–The memory this register points to is a special area called the stack.

–The stack is an area of memory used to hold data that will be retreived soon.

–The stack is usually accessed in a Last In First Out (LIFO) fashion.

Non Programmable Registers

● Instruction Register & Decoder

– Instruction is stored in IR after fetched by processor

– Decoder decodes instruction in IR

Internal Clock generator

– 3.125 MHz internally

– 6.25 MHz externally

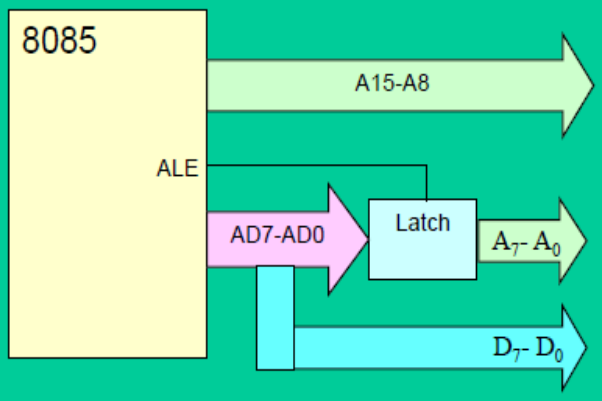
The Address and Data Busses

- The address bus has 8 signal lines A8 – A15 which are unidirectional.
- The other 8 address bits are multiplexed (time shared) with the 8 data bits.
  - So, the bits AD0 – AD7 are bi-directional and serve as A0 – A7 and D0 – D7 at the same time.
    - During the execution of the instruction, these lines carry the address bits during the early part, then during the late parts of the execution, they carry the 8 data bits.
  - In order to separate the address from the data, we can use a latch to save the value before the function of the bits changes.

Demultiplexing AD7-AD0

- From the above description, it becomes obvious that the AD7– AD0 lines are serving a dual purpose and that they need to be demultiplexed to get all the information.
- The high order bits of the address remain on the bus for three clock periods. However, the low order bits remain for only one clock period and they would be lost if they are not saved externally. Also, notice that the low order bits of the address disappear when they are needed most.
- To make sure we have the entire address for the full three clock cycles, we will use an external latch to save the value of AD7– AD0 when it is carrying the address bits. We use the ALE signal to enable this latch.

Demultiplexing AD7-AD0



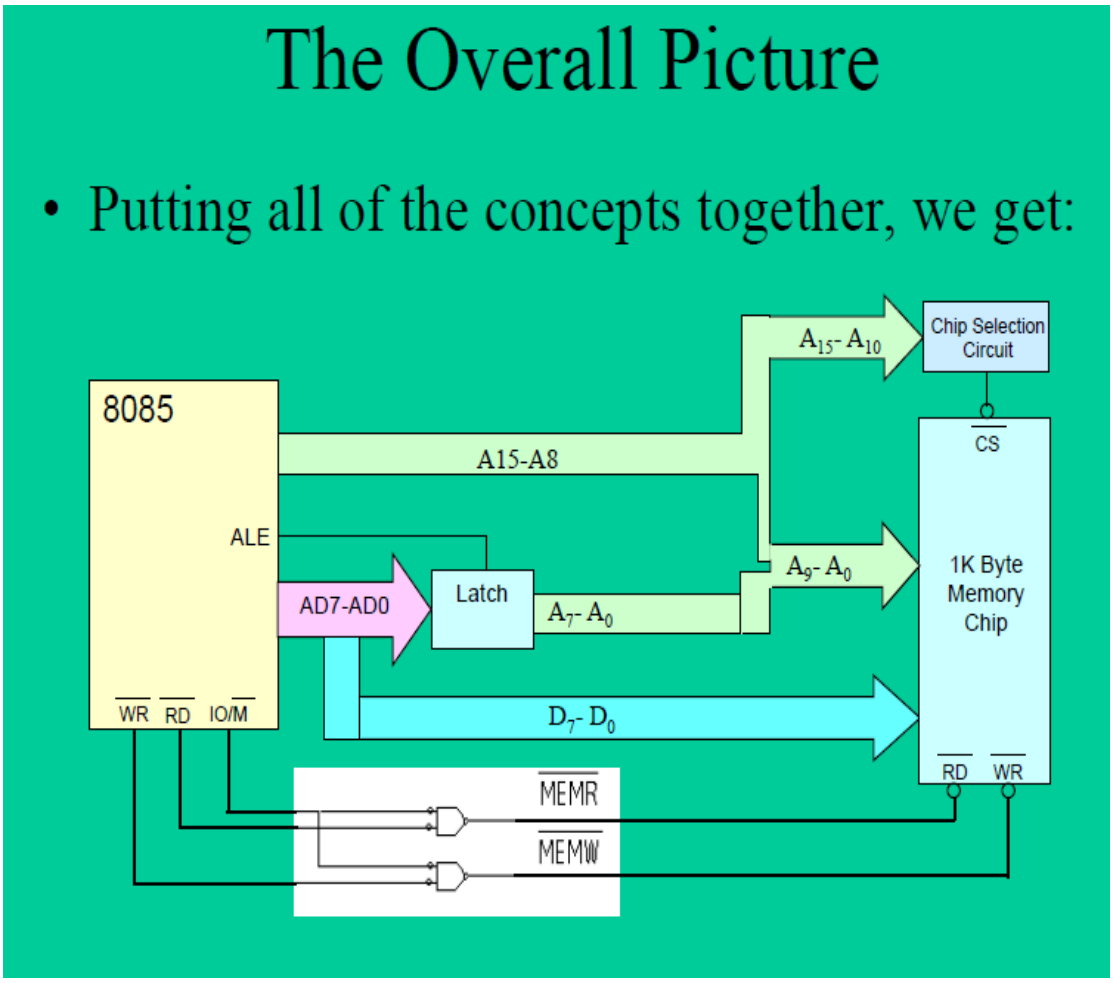
– Given that ALE operates as a pulse during T1, we will be able to latch the address. Then when ALE goes low, the address is saved and the AD7– AD0 lines can be used for their purpose as the bi-directional data lines.

• The high order address is placed on the address bus and hold for 3 clk periods,

• The low order address is lost after the first clk period, this address needs to be hold however we need to use latch

• The address AD7 –AD0 is connected as inputs to the latch 74LS373.

• The ALE signal is connected to the enable (G) pin of the latch and the OC –Output control –of the latch is grounded



Serial I/O Controller

Controls in serial I/O communication

Two I/O Pins SID(Serial Input Data) and SOD(Serial output Data)

Interrupt controller

☐ Managing the interrupt.

☐ Can vector an interrupt request

☐ Solve levels of interrupt priorities.

☐ Have 6 interrupt pins

☐ INTR (Input)

o INTERRUPT REQUEST; is used as a general purpose interrupt.

o It is sampled only during the next to the last clock cycle of the instruction. If it is active, the Program Counter (PC) will be inhibited from incrementing and an INTA will be issued.

o During this cycle a RESTART or CALL instruction can be inserted to jump to the interrupt service routine.

o The INTR is enabled and disabled by software.

o It is disabled by Reset and immediately after an interrupt is accepted.

☐ INTA (Output)

o INTERRUPT ACKNOWLEDGE; is used instead of (and has the same timing as) RD during the Instruction cycle after an INTR is accepted.

o It can be used to activate the 8259 Interrupt chip or some other interrupt port.

RST 5.5

RST 6.5 - (Inputs)

RST 7.5

RESTART INTERRUPTS; These three inputs have the same timing as I NTR except they cause an internal RESTART to be automatically inserted.

RST 7.5 Highest Priority

RST 6.5

RST 5.5 Lowest Priority

The priority of these interrupts is ordered as shown above. These interrupts have a higher priority than the INTR.

TRAP (Input)

Trap interrupt is a non maskable restart interrupt. It is recognized at the same time as

INTR. It is unaffected by any mask or Interrupt Enable. It has the highest priority of

any interrupt.

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

o Sequencing

o 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.

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 :

o Control signals to Memory

o Control signals to I/O

o Control Signals within the Processor.

Control and status Signal in 8085

☐ This group of signal indicates two control signals RD and WR.

☐ Three status signals (IO/M, S1 and S0) to identify the nature of operation.

☐ One special signal (ALE) to indicate the beginning of the operation.

☐ ALE (Address Latch Enable)

☐ This is a positive going pulse generated every time the 8085 begins an operation (Machine cycle); it indicates that the bits on AD7- AD0 are address bits. This signal is used primarily to latch the low-order address from the multiplexed bus and generate a separate set of eight address lines A7-a0.

☐ RD-Read

RESET IN (Input)

Reset sets the Program Counter to zero and resets the Interrupt Enable and HLDA flipflops. None of the other flags or registers (except the instruction register) are

affected The CPU is held in the reset condition as long as Reset is applied.

RESET OUT (Output)

Indicates CPlJ is being reset. Can be used as a system RESET. The signal is synchronized to the processor clock.

X1, X2 (Input)

Crystal or R/C network connections to set the internal clock generator X1 can also be an external clock input instead of a crystal. The input frequency is divided by 2 to

give the internal operating frequency.

CLK (Output)

Clock Output for use as a system clock when a crystal or R/ C network is used as an input to the CPU. The period of CLK is twice the X1, X2 input period.

IO/M (Output)

IO/M indicates whether the Read/Write is to memory or l/O Tristated during Hold and Halt modes.

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. **Implied 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 A,B	REGISTER B	REGISTER A

- **It is carried out with 8 bit registers A,B,C,D,E,H & L**
- **It is important to use registers of same size.**
- **Never mix an 8 bit register with a 16 bit register i.e. MOV A,SP**

EXAMPLES

MOV A,B : Copys B into A
MOV SP,H : Copys H pair into SP

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
MVI C,3AH	DATA 3AH	REGISTER C

EXAMPLES

MOV A,90 : Copys 90 into A

LXI H,1234H : Copys 1234H into H

3. DIRECT ADDRESSING MODE

- In this scheme, the address of the data is defined in the instruction itself.

INSTRUCTION	SOURCE	DESTINATION
LDA 2000H	MEMORY LOCATION 2000 H	REGISTER A

EXAMPLES

LHLD 1000h : Copies the content of 1000h address memory to L and 1000h memory to H.

LDA 2000H :Copies the content of 2000h memory to Accumulator

JMP 4000h

Call 5000H

4. REGISTER INDIRECT ADDRESSING MODE

- Register Indirect Addressing allows data to be addressed at any memory location through an address held in any of the H pair, B pair and D pair registers.

- **It transfers byte/word between a register and a memory location addressed.**

INSTRUCTION	SOURCE	DESTINATION
MOV C,M	ASSUME HL=1000H and M is the Content of 1000H Address.	REGISTER C

EXAMPLES

MOV C,M : Copys the word contents of the memory location addressed by HL pair into C.

STAX B : Copys A into the memory location addressed by B pair.

5. Implied Addressing Mode

- The addressing mode of certain instructions is implied by the instruction's function.

INSTRUCTION	SOURCE	DESTINATION
STC		Carry Flag

EXAMPLES

STC : Set carry Flag

CMC : Complement carry flag.

DAA : Decimal Adjust Accumulator content.

8085 Instruction Set:

DATA TRANSFER INSTRUCTIONS

Opcode	Operand	Description
Copy from source to destination		
MOV	Rd, Rs M, Rs Rd, M	This instruction copies the contents of the source register into the destination register; the contents of the source register are not altered. If one of the operands is a memory location, its location is specified by the contents of the HL registers. Example: MOV B, C or MOV B, M
Move immediate 8-bit		
MVI	Rd, data M, data	The 8-bit data is stored in the destination register or memory. If the operand is a memory location, its location is specified by the contents of the HL registers. Example: MVI B, 57H or MVI M, 57H
Load accumulator		
LDA	16-bit address	The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator. The contents of the source are not altered. Example: LDA 2034H
Load accumulator indirect		
LDAX	B/D Reg. pair	The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the accumulator. The contents of either the register pair or the memory location are not altered. Example: LDAX B
Load register pair immediate		
LXI	Reg. pair, 16-bit data	The instruction loads 16-bit data in the register pair designated in the operand. Example: LXI H, 2034H or LXI H, XYZ
Load H and L registers direct		
LHLD	16-bit address	The instruction copies the contents of the memory location pointed out by the 16-bit address into register L and copies the contents of the next memory location into register H. The contents of source memory locations are not altered. Example: LHLD 2040H
Store accumulator direct		
STA	16-bit address	The contents of the accumulator are copied into the memory location specified by the operand. This is a 3-byte instruction,

the second byte specifies the low-order address and the third byte specifies the high-order address.
Example: STA 4350H

Store accumulator indirect
STAX Reg. pair

The contents of the accumulator are copied into the memory location specified by the contents of the operand (register pair). The contents of the accumulator are not altered.
Example: STAX B

Store H and L registers direct
SHLD 16-bit address

The contents of register L are stored into the memory location specified by the 16-bit address in the operand and the contents of H register are stored into the next memory location by incrementing the operand. The contents of registers HL are not altered. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address.
Example: SHLD 2470H

Exchange H and L with D and E
XCHG none

The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E.
Example: XCHG

Copy H and L registers to the stack pointer
SPHL none

The instruction loads the contents of the H and L registers into the stack pointer register, the contents of the H register provide the high-order address and the contents of the L register provide the low-order address. The contents of the H and L registers are not altered.
Example: SPHL

Exchange H and L with top of stack
XTHL none

The contents of the L register are exchanged with the stack location pointed out by the contents of the stack pointer register. The contents of the H register are exchanged with the next stack location (SP+1); however, the contents of the stack pointer register are not altered.
Example: XTHL

Push register pair onto stack
PUSH Reg. pair

The contents of the register pair designated in the operand are copied onto the stack in the following sequence. The stack pointer register is decremented and the contents of the high-order register (B, D, H, A) are copied into that location. The

stack pointer register is decremented again and the contents of the low-order register (C, E, L, flags) are copied to that location.
Example: PUSH B or PUSH A

Pop off stack to register pair

POP	Reg. pair	The contents of the memory location pointed out by the stack pointer register are copied to the low-order register (C, E, L, status flags) of the operand. The stack pointer is incremented by 1 and the contents of that memory location are copied to the high-order register (B, D, H, A) of the operand. The stack pointer register is again incremented by 1. Example: POP H or POP A

Output data from accumulator to a port with 8-bit address

OUT	8-bit port address	The contents of the accumulator are copied into the I/O port specified by the operand. Example: OUT F8H

Input data to accumulator from a port with 8-bit address

IN	8-bit port address	The contents of the input port designated in the operand are read and loaded into the accumulator. Example: IN 8CH

ARITHMETIC INSTRUCTIONS

Opcode	Operand	Description

Add register or memory to accumulator

ADD	R M	The contents of the operand (register or memory) are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. Example: ADD B or ADD M

Add register to accumulator with carry

ADC	R M	The contents of the operand (register or memory) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. Example: ADC B or ADC M

Add immediate to accumulator

ADI	8-bit data	The 8-bit data (operand) is added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. Example: ADI 45H

Add immediate to accumulator with carry

ACI	8-bit data	The 8-bit data (operand) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. Example: ACI 45H

Add register pair to H and L registers

DAD	Reg. pair	The 16-bit contents of the specified register pair are added to the contents of the HL register and the sum is stored in the HL register. The contents of the source register pair are not altered. If the result is larger than 16 bits, the CY flag is set. No other flags are affected. Example: DAD H

Subtract register or memory from accumulator

SUB	R M	The contents of the operand (register or memory) are subtracted from the contents of the accumulator, and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction. Example: SUB B or SUB M

Subtract source and borrow from accumulator

SBB	R M	The contents of the operand (register or memory) and the Borrow flag are subtracted from the contents of the accumulator and the result is placed in the accumulator.If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction. Example: SBB B or SBB M

Subtract immediate from accumulator

SUI	8-bit data	The 8-bit data (operand) is subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtraction.

Example: SUI 45H

Subtract immediate from accumulator with borrow

SBI 8-bit data

The 8-bit data (operand) and the Borrow flag are subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtracion.
Example: SBI 45H

Increment register or memory by 1

INR R
M

The contents of the designated register or memory) are incremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers.
Example: INR B or INR M

Increment register pair by 1

INX R

The contents of the designated register pair are incremented by 1 and the result is stored in the same place.
Example: INX H

Decrement register or memory by 1

DCR R
M

The contents of the designated register or memory are decremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers.
Example: DCR B or DCR M

Decrement register pair by 1

DCX R

The contents of the designated register pair are decremented by 1 and the result is stored in the same place.
Example: DCX H

Decimal adjust accumulator

DAA none

The contents of the accumulator are changed from a binary value to two 4-bit binary coded decimal (BCD) digits. This is the only instruction that uses the auxiliary flag to perform the binary to BCD conversion, and the conversion procedure is described below. S, Z, AC, P, CY flags are altered to reflect the results of the operation.

If the value of the low-order 4-bits in the accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low-order four bits.

If the value of the high-order 4-bits in the accumulator is greater than 9 or if the Carry flag is set, the instruction adds 6 to the high-order four bits.

Example: DAA

BRANCHING INSTRUCTIONS

Opcode	Operand	Description
Jump unconditionally		
JMP	16-bit address	The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. Example: JMP 2034H

or JMP XYZ Jump conditionally

Operand: 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW as described below.
Example: JZ 2034H or JZ XYZ

Opcode	Description	Flag Status
JC	Jump on Carry	CY = 1
JNC	Jump on no Carry	CY = 0
JP	Jump on positive	S = 0
JM	Jump on minus	S = 1
JZ	Jump on zero	Z = 1
JNZ	Jump on no zero	Z = 0
JPE	Jump on parity even	P = 1
JPO	Jump on parity odd	P = 0

Unconditional subroutine call

CALL 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. Before the transfer, the address of the next instruction after CALL (the contents of the program counter) is pushed onto the stack.
Example: CALL 2034H or CALL XYZ

Call conditionally

Operand: 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW as described below. Before the transfer, the address of the next instruction after the call (the contents of the program counter) is pushed onto the stack.
Example: CZ 2034H or CZ XYZ

Opcode	Description	Flag Status
CC	Call on Carry	CY = 1
CNC	Call on no Carry	CY = 0
CP	Call on positive	S = 0
CM	Call on minus	S = 1
CZ	Call on zero	Z = 1
CNZ	Call on no zero	Z = 0
CPE	Call on parity even	P = 1
CPO	Call on parity odd	P = 0

Return from subroutine unconditionally

RET none

The program sequence is transferred from the subroutine to the calling program. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address.
Example: RET

Return from subroutine conditionally

Operand: none

The program sequence is transferred from the subroutine to the calling program based on the specified flag of the PSW as described below. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address.
Example: RZ

Opcode	Description	Flag Status
RC	Return on Carry	CY = 1
RNC	Return on no Carry	CY = 0
RP	Return on positive	S = 0
RM	Return on minus	S = 1
RZ	Return on zero	Z = 1
RNZ	Return on no zero	Z = 0
RPE	Return on parity even	P = 1
RPO	Return on parity odd	P = 0

Load program counter with HL contents

PCHL none

The contents of registers H and L are copied into the program counter. The contents of H are placed as the high-order byte and the contents of L as the low-order byte.

Example PCHL

Restart

RST 0-7

The RST instruction is equivalent to a 1-byte call instruction to one of eight memory locations depending upon the number. The instructions are generally used in conjunction with interrupts and inserted using external hardware. However these can be used as software instructions in a program to transfer program execution to one of the eight locations. The addresses are:

Instruction	Restart Address
RST 0	0000H
RST 1	0008H
RST 2	0010H
RST 3	0018H
RST 4	0020H
RST 5	0028H
RST 6	0030H
RST 7	0038H

The 8085 has four additional interrupts and these interrupts generate RST instructions internally and thus do not require any external hardware. These instructions and their Restart addresses are:

Interrupt	Restart Address
TRAP	0024H
RST 5.5	002CH
RST 6.5	0034H
RST 7.5	003CH

LOGICAL INSTRUCTIONS

Opcode	Operand	Description
Compare register or memory with accumulator		

CMP	R
	M
		The contents of the operand (register or memory) are compared with the contents of the accumulator. Both contents are preserved . The result of the comparison is shown by setting the flags of the PSW as follows:
		if (A) < (reg/mem): carry flag is set
		if (A) = (reg/mem): zero flag is set
		if (A) > (reg/mem): carry and zero flags are reset
		Example: CMP B or CMP M

Compare immediate with accumulator		

CPI	8-bit data	The second byte (8-bit data) is compared with the contents of the accumulator. The values being compared remain unchanged. The result of the comparison is shown by setting the flags of the PSW as follows:
		if (A) < data: carry flag is set
		if (A) = data: zero flag is set
		if (A) > data: carry and zero flags are reset
		Example:
		CPI 89H

Logical AND register or memory with accumulator		
ANA	R M	The contents of the accumulator are logically ANDed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set. Example: ANA B or ANA M
Logical AND immediate with accumulator		
ANI	8-bit data	The contents of the accumulator are logically ANDed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set. Example: ANI 86H
Exclusive OR register or memory with accumulator		
XRA	R M	The contents of the accumulator are Exclusive ORed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a

memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset.
Example: XRA B or XRA M

Exclusive OR immediate with accumulator

XRI	8-bit data	The contents of the accumulator are Exclusive ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. Example: XRI 86H

Logical OR register or memory with accumulaotr

ORA	R M	The contents of the accumulator are logically ORed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. Example: ORA B or ORA M

Logical OR immediate with accumulator

ORI	8-bit data	The contents of the accumulator are logically ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. Example: ORI 86H

Rotate accumulator left

RLC	none	Each binary bit of the accumulator is rotated left by one position. Bit D_7 is placed in the position of D_0 as well as in the Carry flag. CY is modified according to bit D_7. S, Z, P, AC are not affected. Example: RLC

Rotate accumulator right

RRC	none	Each binary bit of the accumulator is rotated right by one position. Bit D_0 is placed in the position of D_7 as well as in the Carry flag. CY is modified according to bit D_0. S, Z, P, AC are not affected. Example: RRC

Rotate accumulator left through carry

RAL	none	Each binary bit of the accumulator is rotated left by one

position through the Carry flag. Bit D_7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D_0. CY is modified according to bit D_7. S, Z, P, AC are not affected.
Example: RAL

Rotate accumulator right through carry

RAR none — Each binary bit of the accumulator is rotated right by one position through the Carry flag. Bit D_0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D_7. CY is modified according to bit D_0. S, Z, P, AC are not affected.
Example: RAR

Complement accumulator

CMA none — The contents of the accumulator are complemented. No flags are affected.
Example:
CMA

Complement carry

CMC none — The Carry flag is complemented. No other flags are affected.

Example:
CMC

Set Carry

STC none — The Carry flag is set to 1. No other flags are affected.

Example: STC

CONTROL INSTRUCTIONS

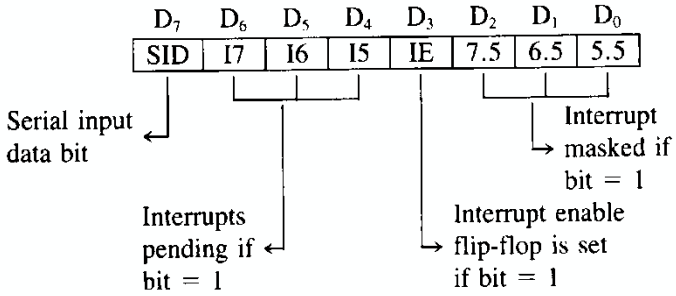
Opcode	Operand	Description
No operation		
NOP	none	No operation is performed. The instruction is fetched and decoded. However no operation is executed. Example: NOP
Halt and enter wait state		
HLT	none	The CPU finishes executing the current instruction and halts any further execution. An interrupt or reset is necessary to exit from the halt state. Example: HLT
Disable interrupts		
DI	none	The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled. No flags are affected. Example: DI
Enable interrupts		
EI	none	The interrupt enable flip-flop is set and all interrupts are enabled. No flags are affected. After a system reset or the acknowledgement of an interrupt, the interrupt enable flip-flop is reset, thus disabling the interrupts. This instruction is necessary to reenable the interrupts (except TRAP). Example: EI

Read interrupt mask

RIM none

This is a multipurpose instruction used to read the status of interrupts 7.5, 6.5, 5.5 and read serial data input bit. The instruction loads eight bits in the accumulator with the following interpretations.

Example: RIM

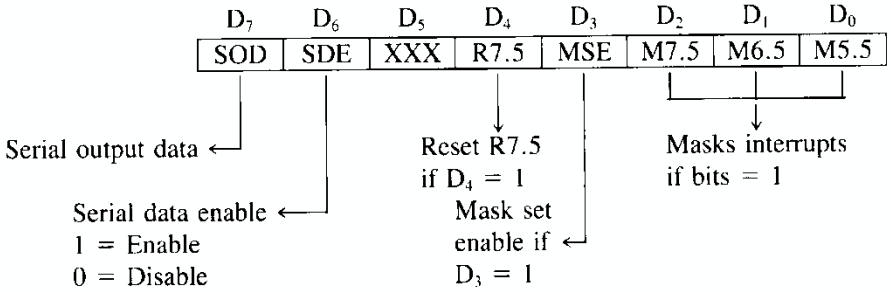


Set interrupt mask

SIM none

This is a multipurpose instruction and used to implement the 8085 interrupts 7.5, 6.5, 5.5, and serial data output. The instruction interprets the accumulator contents as follows.

Example: SIM



- SOD — Serial Output Data: Bit D_7 of the accumulator is latched into the SOD output line and made available to a serial peripheral if bit $D_6 = 1$.
- SDE — Serial Data Enable: If this bit = 1, it enables the serial output. To implement serial output, this bit needs to be enabled.
- XXX — Don't Care
- R7.5 — Reset RST 7.5: If this bit = 1, RST 7.5 flip-flop is reset. This is an additional control to reset RST 7.5.
- MSE — Mask Set Enable: If this bit is high, it enables the functions of bits D_2, D_1, D_0. This is a master control over all the interrupt masking bits. If this bit is low, bits D_2, D_1, and D_0 do not have any effect on the masks.
- M7.5 — D_2 = 0, RST 7.5 is enabled.
  = 1, RST 7.5 is masked or disabled.
- M6.5 — D_1 = 0, RST 6.5 is enabled.
  = 1, RST 6.5 is masked or disabled.
- M5.5 — D_0 = 0, RST 5.5 is enabled.
  = 1, RST 5.5 is masked or disabled.

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**

8086 INTERNAL ARCHITECTURE

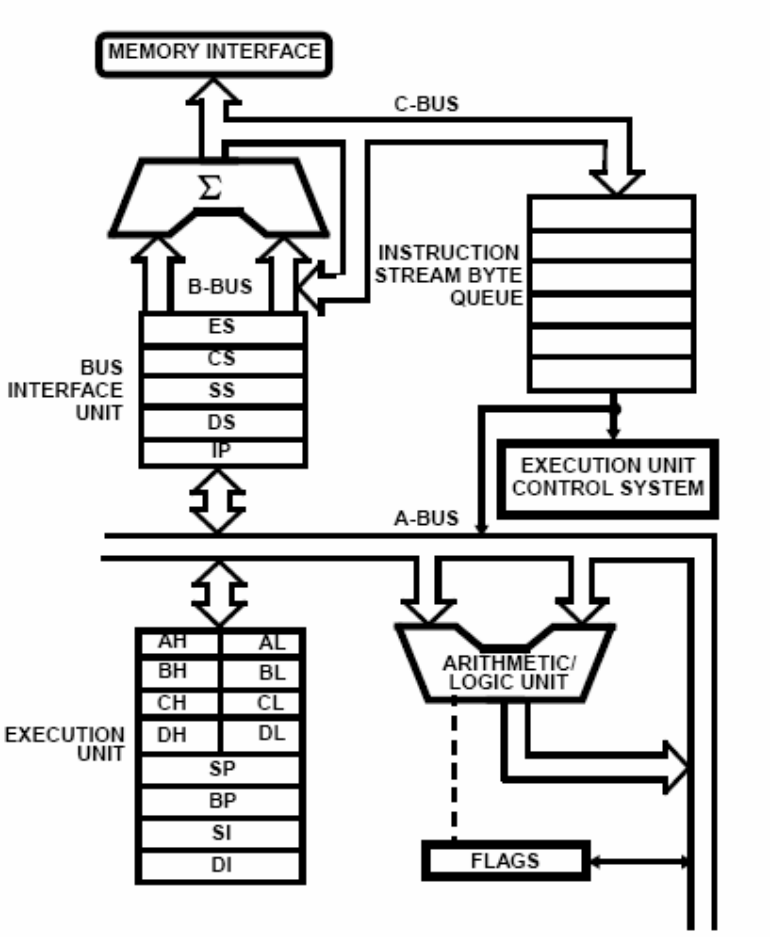


Figure: Internal Architecture of 8086

- **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.**

A. THE EXECUTION UNIT

- **The EU contains control circuitry which directs internal operations.**
- **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 = AccumulatorRegister

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

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]
OF = OVERFLOW FLAG

CONTROL FLAG

TF = SINGLE STEP TRAPFLAG
IF = INTERRUPT ENABLE FLAG
DF = STRING DIRECTION 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.**

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 pipelining**

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.**

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]**

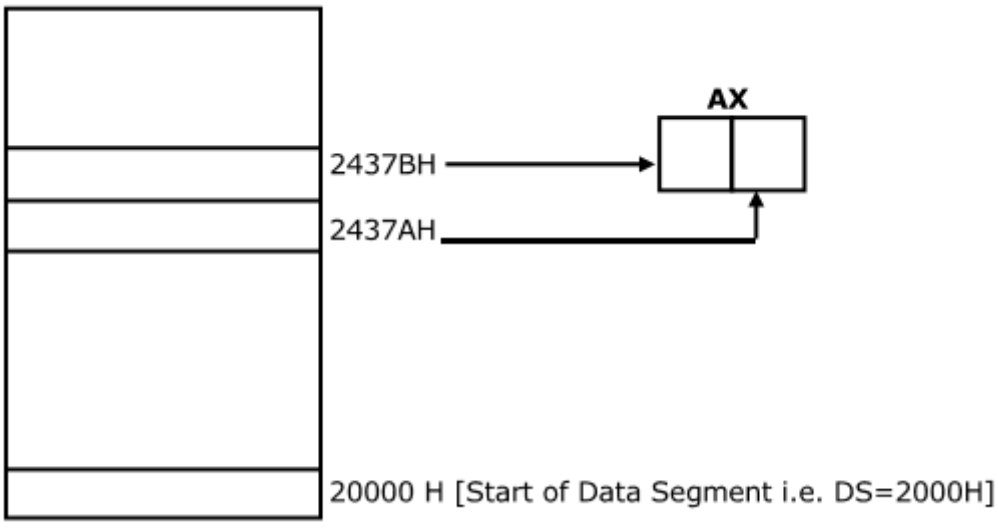
  Assume DS=2000H

- **To Compute the physical address Add**

  20000 H and 437AH 20000 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.
- 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 348A0 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
SS [STACK SEGMENT]	SP [STACK POINTER] BP [BASE POINTER]	ADDRESS
DS [DATA SEGMENT]	BX [BASE REGISTER] DI [DESTINATION INDEX] SI [SOURCE INDEX] 8 BIT NUMBER 16 BIT NUMBER	STACK ADDRESS
ES [EXTRA SEGMENT]		

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

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 10000 H + 1234 H 11234H	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 10000 H + 0300 H 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.

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 10000H + 0300H + 0200H 10500H MEMORY LOCATION

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 10000H + 0300H + 4H 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.

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 10000H + 0300H + 0200H +05H 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

INSTRUCTION SETS

1. DATA TRANSFER INSTRUCTIONS

1.1 GENERAL PURPOSE BYTE OR WORD TRANSFER INSTRUCTIONS

INSTRUCTIONS	COMMENTS
MOV MOV Destination,Source	Copy byte or word from specified source to specified destination.
PUSH PUSH Source	Copy specified word to top of stack.
POP POP Destination	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.**

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.

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.**

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.

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,Sourc e 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.

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)

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.

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.

UNIT 5

ASSEMBLY LANGUAGE PROGRAMMING

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	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.**

8085 PROGRAMMING EXAMPLES

1.Write a program to perform the following :

a). Load the no: 1BH in D

b) Load the no. B5H in B

C). Increment the content of B by 1.

d). Decrement the content of D by 1.

e). Subtract the content of D from the content of B.

f). Display the result at OUT port 1.

MVI D,1BH
MVI B,B5H
INR B
DCR D
MOV A, B
SUB D
OUT PORT 1
HLT

2 Wap to load the data byte in the register C. Mask the high- order bits (D7-D4) and display the low order bits (D3-D0) at outport. Exclusive-OR the result with 57H and display at OUT PORT2.

Solution: MVI C, A8H

MOV A, C

ANI OFH

OUT PORT1

XRI 57H

OUT PORT 2

HLT

3. Wap to load the byte 8EH in register D and F7H in register E. Mask the higher order bits (D7 – D4) from both the data bytes , EX-OR the low order bit (D3-D0) and display the answer.

Solution:

MVI D, 8EH

MOI A, D

ANI

MVI D, 8EH

MVI E, F7H

MOV A, D

ANI OFH

MOV D, A

MOV A, E

ANI OFH

XRA D

OUT PORT 1

HLT

4. Write a program to load two unsigned nos in register B and C respectively . Subtract c from B. If the result in 2's complement convert the result in absolute magnitude and display it port 1. Otherwise display the result.

Solution:

MVI B, byte 1

MVI C, byte 2

MOV A, B

SUB C

JNC label 1

CMA

ADI 01H

Label 1 OUT PORT 1

HLT

5. Write an ALP to do the following:

a) Load A with byte 1.

b) Load B with byte 2.

c) Compare the equality of the contents of A and B

d) If two nos . are equal , display 01 otherwise display 00H at port 1.

Solution:

MVI A, byte 1

MVI B, byte 2

SUB B

JNZ loop

MVI A, 01H

OUT PORT 1

HLT

Loop MVI A, 00H

OUT PORT

HLT

6.The following block of data is stored in memory location from CO55 to C05A H. Transfer the entire block of data to the locations C080 to C085 H in reverse order. Data: 22, A5, B2, 99, 7F, 37

Solution:

```
LXI H, C055 H

LXI D, C085 H
MOV B, 06H

Next MVI A, M
STAX D
INX H
DCX D
DCR B
JNZ next
 HLT.
```

7. Write a program to find larger of two nos. 1st no in C001 and 2nd no in C002 and result in C003 H.

Solution:

LXI H, C001 H

MOV A, M

INX H

CMP M

JNC loop

MOV A, M

Loop STA C003 H

HLT

8. Write an ALP to find the smallest no in a data array. Data from location C000H to C005 H.

Solution:

```
LXI H, C000H
MVI C, 06H
MOV A, M
DCR C
Loop INX H
CMP M
JC loop1
MOV A, M
Loop1 DCR C
JNZ loop
STA C0C0 H
HLT.
```

9. Write an ALP to multiply two nos: eg 05 H × 08 H

Solution:

```
MVI A, 00H
MVI B, 08H
MVI C, 05H
Loop  ADD B
DCR C
JNZ loop
STA C000H
HLT
```

10.Write an ALP for the following addition 12+22+32+42+52+62+72+82+92

Solution:

```
 MVI A, 00H
MVI B, 09H
Loop 1 MOV C, B
```

Loop 2 ADD B

DCR C

JNZ loop2

DCR B

JNZ loop 1

OUT PORT 1

HLT

11.Write an ALP to count the no of 1 in the given string '10100110' and display the result at COCOH

Solution:

MVI A, A6 H

MVI B, 00H

MVI C, 08H

Loop1 RAL

JNC loop2

INR B

Loop2 DCR C

JNZ loop 1

MOV A, B

STA COCOH

HLT

12.The following datas are stored in memory location starting from C0B0 to C0B9 H . Take a test no. 48. Find out how many times the no 48 is repeated. Display the result at C0C0H .

DADA: 12, 23, 34, 45, 48, 56, 48, 67, 48, 89.

Solution:

LXI H, C0B0 H

MVI B, 00H

MVI C, 0A H

Loop 1 MOV A, M

CPI 48 H

JNZ loop2

INR B

Loop 2 INX H

DCR C

JNZ loop1

MOV A,B

STA COCO H

HLT

13.8 bit multiplication , product is 16 bit . The multiplicand is loaded in the two consecutive memory locations 2501 and 2502 H . The multiplier is stored in 2053 H. Store the

product in 2504 and 2505 H.

Solution:

LHLD 2501 H

XCHG

LDA 2503 H

LXI H, 0000

MVI C, 08

Loop1 DAD H

RAL

JNC loop2

DAD D

Loop 2 DCR C

JNZ loop1

SHLD 2504 H

HLT

14.Write an ALP to divide two nos. The dividend is in C001 and divisor is in C002. Store the quotient in C0C0 H and remainder in C0C1.

Solution:

```
LXI H, C001 H
MOV A,M
INX H
MOV B,M
MVI C, 00H
Loop1 CMP B
JC Loop2
INR C
SUB B
JNZ Loop 1
Loop2 STA COC1 H
MOV A, C
STA COCO H
HLT
```

15.To arrange 54 , EB, 85, A8 & 99 in descending order. These numbers are stored in the memory location 2501 to 2505 H. The count = 05 is restored in 2500 H. Results are to be stored in 2601 to 2605 H.

Solution:

```
LXI D, 2601 H
LXI H, 2500 H
 MOV B, M
Start CALL Subroutine 1
STAX D
CALL Subroutine 2
```

INX D

DCR B

JNZ start

HLT

Subroutine 1:

LXI H, 2500 H

SUB A

Loop1: INX H

CMP M

JNC loop 2

MOV A, M

Loop2: DCR C

JNZ Loop 1

RET

Subroutin2: LXI H, 2500H

MOV C, M

Loop1 INX H

CMP M

JZ Loop2

DCR C

JNZ Loop1

Loop2 MVI A, 00H

MOV M, A

RET

Note: Subroutine 1 gives the largest number of array.

Subroutine 2 find the largest number and replace it by 00.

Counter and delay:

Timing delay using one register:

MVI C, FFH ---------- 7 T state

Loop DCR C ---------- 4 T state

JNZ Loop --------10 or 7 T state

Consider a micro computer with 2 MHZ frequency

Clock period , T = 1/f = ½ = 0.5 µ sec

Delay for inst. Outside the loop To = No of Ts state × T

= 7 × 0.5

= 3.5 µ sec

Delay for inst inside the loop, TL = No of T state × T * (N10)

= (14× 0.5 × 10-6 ×255)

= 1785 µ sec

Now TLA = TL-3× 0.5

= 1785 – 1.5

= 1.7835 ms

TD = T0 + TL

= 3.5 + 1785

= 1788.5

= 1.7885 ms

<u>Time delay for register pair:</u>

LXI B, 2384 H ……. 10 T state

Loop DCX B ……. 6 Tstate

MOV A, C …….. 4 T state

ORA B …… 4 T state

JNZ loop ……. 10/7

Delay calculation:

T0 = T state * T

= 10 × 0.5 × 10-3

= 109 ms

TL = No of T state * T * (N10)

= 24 * 0.5 * 90.92

Total Delay = T0 + TL

=

<u>Time delay using a loop within a loop:</u>

MVI B, 38 H …….. 7 T

Loop2 MVI C, FFH ……7 T

Loop 1 DCR C……..4 T

JNZ loop1 ------10/7 T

DCR B ------ 4 T

JNZ loop2 ------ 10/7 T

<u>Delay calculation:</u>

To = 7 * 0.5 = 3.5 µ sec

TL1 = 14 * 0.5 * 255 – 3*0.5

TL2 = (TL1 + 21*0.5) N10 – 3

Total dealy = To + TL2

Q. Write a program to count continuously in hexadecimal from FFH to 00H in a system with a 0.5 µ s clock period. Use register C to set up a 1 ms delay between each count and display the number at one of the o/p ports.

MVI B, 00H

Next DCR B

MVI C, count

Delay DCR C

JNZ delay

MOV A, B

OUT port 1

JMP Next

Delay calculation:

TL = T state * T* (T10)

= 14 × 0.5 × count

= 7 × count μ s

To = 35 × T

= 35 × 0.5

= 17.5 μ s

TD = TL + To

1 ms = 7 * count * 10-6 + 17.5 * 10-6

Count = 140.35

= 8C H

Q. Write a program to generate a continuous square wave with the period 500 μ sec. Assume the system clock period is 325 μ sec and use bit Do to output the square wave.

Solution:

MVI D, AA

ROTATE

MOV A, D

RLC

MOV D, A

ANI 01 H

OUT PORT 1

MVI B, count

Delay DCR B

JNZ delay

JMP ROTATE

Delay calculation:

TL = 14 * 325 * 10-9 * count or 14 * 325 * (count - 1) + 11 T-state * 325

T0 = 46 * 325 * 10-9

TD = TL + T0

250 = (52.4)10 = 34 H

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.**

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]

……

……

……

……

……

[ENDDIRECTIVES]

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
```

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.**

 # 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]

#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.**

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 "Hello
  World"
  DB "St.Xavier's College"

2. NUMERIC CONSTANTS

#BINARY : VAL1 DB 10101010B

#DECIMAL : VAL1 DB 230

#HEXADECIMAL : VAL1 DB 23H

```
;Program to Print Hello World in ALP

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

.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



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

```
;Program to Print the Sum of Two 8 Bit Numbers

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

.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

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

```
;16 Bit Binary Content of Ax is Converted to 4 Digit ASCII

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

.MODEL SMALL

.STACK

.DATA

        VAL DW 256

.CODE

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


MAIN PROC

        MOV AX,@DATA

        MOV DS,AX     ;DATA SEGMENT INITIALIZATION


        MOV AX,VAL

        XOR DX,DX    ;DX IS CLEARED

        MOV CX,100   ;DIVISOR IS PASSED INTO CX REGISTER


        DIV CX         ;DX:AX PAIR DIVIDED BY CX

                       ;QUOTIENT  IN AX

                       ;REMAINDER IN DX


         AAM             ;QUOTIENT IS ADJUSTED TO UNPACKED BCD

        ADD AX,3030H   ;QUOTIENT IS CONVERTED TO ASCII

        XCHG AX,DX       ;DX AND AX ARE SWAPPED

        AAM              ;REMAINDER IS ADJUSTED TO UNPACKED BCD

        ADD AX,3030H   ;REMAINDER IS CONVERTED TO ASCII


         ;;;;;;;;;;;;;;;;;;;DISPLAY OPERATION
```

```
        ;;DISPLAY QUOTIENT
        ;;DISPLAY REMAINDER



        ;;;;;;;;;;;;;;;;;;;;END DISPLAY



        MOV AX,4C00H ;END REQUEST
        INT 21H       ;DOS INTERRUPT FUNCTION

MAIN ENDP
END MAIN

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

```
;Program to Print the Difference of Two 8 Bit Numbers

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

.MODEL SMALL

.STACK

.DATA

        VAL1 DB 89

        VAL2 DB 10

        MSG DB 'DIFFERENCE OF 2 NUMBERS: $'

.CODE

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


MAIN PROC

    MOV AX,@DATA

    MOV DS,AX


    MOV AX,0

    MOV AL,VAL1

    SUB AL,VAL2



    ;;;;; AAM Converts Binary Value to Unpacked BCD.

    ;;;;; AAM Only Works with AX Register

    AAM


    ;;;;; AX is Added with 3030H to Obtain ASCII Value.

    ADD AX,3030H


    PUSH AX
```

```
    ;;;;; DISPLAY MESSAGE With AH=09H
    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

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

```
;Program to Print the Sum of Two 16 Bit Numbers
;-----------------------------------------------------------------
------
.MODEL SMALL
.STACK
.DATA
        VAL1 DW 2010
        VAL2 DW 2050
 .CODE
;-----------------------------------------------------------------
------
MAIN PROC
      MOV AX,@DATA
      MOV DS,AX    ;INITIALIZE THE DATA SEGMENT

      MOV AX,VAL1
      ADD AX,VAL2  ;AX Holds 16 bit Value
      ;; 16 Bit Answer Splitting Strategy.
      ;; 16 Bit Division is Carried out
      ;; 32 Bit Divident (DX:AX) and 16 Bit Divisor is Required.

      XOR DX,DX    ;Register DX is Cleared
      MOV CX,100
      DIV CX       ;DX:Ax Divided by CX.
                   ;Remainder in DX and Quotient in AX

      AAM          ;Quotient is Converted to Unpacked BCD
      ADD AX,3030H ;Ready  For Display i.e Converted to ASCII
      MOV BX,AX    ;Store the Quotient to BX

      XCHG AX,DX   ;Exchange the Contents of AX and DX
      AAM          ;Remainder is Converted to Unpacked BCD
```

```
      ADD AX,3030H ;Remainder Converted to ASCII
      PUSH AX      ;Remainder Pushed to Stack
      ;;;;;;;;;;;;;;Quotient Ready in BX and Remainder Ready in
AX
      ;;;;;;;;;;;;;;Display   Quotient   First   and   then   the
Remainder


      ;;;;;;;;;;;;;;Display Operation Started
      MOV DL,BH
      MOV AH,02H
      INT 21H
      MOV DL,BL
      MOV AH,02H
      INT 21H
      POP AX
      MOV DL,AH
      MOV DH,AL
      MOV AH,02H
      INT 21H
      MOV DL,DH
      MOV AH,02H
      INT 21H
      ;;;;;;;;;;;;;; Display Operation End


      MOV AX,4C00H ; End Request AH=4CH
      INT 21H       ; Dos Interrupt Function
MAIN ENDP
END MAIN
;-----------------------------------------------------------------
------
```

```
;Program to Display Numbers From 0 To 9.[0 1 2 3 4 5 6 7 8 9 ]

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

.MODEL SMALL

.STACK

.DATA

        VAL DB '0'

.CODE


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

SPACE MACRO

        MOV DL,' '

        MOV AH,02H

        INT 21H

ENDM

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


MAIN PROC

        MOV AX,@DATA

        MOV DS,AX


        MOV CX,26

        MOV DL,VAL

        TOP:

                MOV AH,02H

                INT 21H ;DISPLAY THE NUMBER

                INC DL  ;DL IS INCREMENTED BY 1

                PUSH DX ;PUSH THE CONTENT OF DX TO STACK
```

```
                ;;;;;;;;; INVOKED SPACE MACRO
                SPACE
               ;;;;;;;;; END OF SPACE MACRO


                POP DX  ;POP THE CONTENT FROM STACK TO DX
               DEC CX  ;DECREMENT THE CONTENT OF COUNT BY 1.
               JZ LAST ;JUMP TO LABEL LAST IF CX=0
               JMP TOP ;UNCONDITIONAL JUMP TO LABEL TOP

        LAST:
               MOV AH,4CH
               INT 21H
MAIN ENDP
END MAIN

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

```
;Program to Display Numbers From 0 To 9 With Line Feed

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

.MODEL SMALL

.STACK

.DATA

        VAL DB 0

.CODE

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

LFEED MACRO

        MOV AH,06H

        MOV DL,0AH

        INT 21H

        MOV DL,0DH

        INT 21H

ENDM

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


MAIN PROC

        MOV AX,@DATA

        MOV DS,AX


        MOV CX,10


        MOV AH,00H

        MOV AL,VAL

        TOP:
```

```
                MOV BL,AL
                AAM
                ADD AX,3030H
                MOV DL,AL
                MOV AH,02H
                INT 21H

                 ;;;;;;;; INVOKED LFEED MACRO FOR LINE FEED
                LFEED
                 ;;;;;;;; END OF LFEED MACRO

                INC BL
                MOV AL,BL
                DEC CX
                JZ LAST
                JMP TOP
        LAST:
                MOV AH,4CH
                INT 21H
MAIN ENDP
END MAIN

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

```
;Program to Display Alphabets From A To Z. [A B C D E F ……….. Z]


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

.MODEL SMALL

.STACK

.DATA

        VAL DB 'A'

.CODE

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


SPACE MACRO

        MOV DL,' '

        MOV AH,02H

        INT 21H

ENDM


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


MAIN PROC

        MOV AX,@DATA

        MOV DS,AX


        MOV CX,26

        MOV DL,VAL

        TOP:

                MOV AH,02H

                INT 21H

                INC DL

                PUSH DX
```

```
                ;;;;;;;;; INVOKED SPACE MACRO
                SPACE
                ;;;;;;;;; END OF SPACE MACRO

                POP DX
                DEC CX
                JZ LAST
                JMP TOP
        LAST:
                MOV AH,4CH
                INT 21H
MAIN ENDP
END MAIN

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

```
;Program to Print the Sum of Natural Nos From 1 To 10.[1+2+3….+10]


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

.MODEL SMALL

.STACK

.DATA

        VAL DB 1

.CODE

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


MAIN PROC

        MOV AX,@DATA

        MOV DS,AX


        MOV CX,10

        MOV AL,VAL

        MOV DL,0


        TOP:

                ADD DL,AL ;DL=DL+AL

                INC AL    ;INCREMENT THE CONTENT OF AL

                DEC CX    ;DECREMENT THE CONTENT OF CX

                JZ LAST   ;JUMP TO LAST IF CX=0

                JMP TOP   ;UNCONDITIONAL JUMP TO LABEL TOP



        LAST:


                MOV AL,DL  ;FINAL SUM IN DL IS PASSED TO AL
```

```
                                ;AAM ALWAYS WORKS WITH AX REGISTER

                  ;;;;;;;;;;;;DISPLAY OPERATION STARTED
                AAM
                ADD AX,3030H
                MOV DL,AH
                MOV DH,AL
                MOV AH,02H
                INT  21H
                MOV DL,DH
                MOV AH,02H
                INT 21H

                  ;;;;;;;;;;;;END DISPLAY

                MOV AH,4CH
                INT 21H
MAIN ENDP
END MAIN
;-----------------------------------------------------------------
------;Program to Demonstrate Multiplication Table of a Given
Number


;-----------------------------------------------------------------
------
.MODEL SMALL
.STACK
.DATA
       VAL DB 7

.CODE
```

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

GODOWN MACRO
      MOV DL,0DH
      INT 21H
      MOV DL,0AH
      INT 21H
ENDM


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


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


      MOV AX,0  ; Make AX as 0
      MOV CX,10 ;Initialize the Counter
      MOV BL,1


      TOP:
            MOV AL,BL
            MUL VAL ; Multiply the Content of Val and AL : Answer
in AL
            AAM
            ADD AX,3030H
            MOV DL,AH
            MOV DH,AL
            MOV AH,02H
            INT 21H
            MOV DL,DH
            MOV AH,02H
```

```
            INT 21H
            GODOWN ; Invoked GoDOWN Macro
            MOV AX,0
            INC BL
            DEC CX
            JZ LAST
            JMP TOP
      LAST:
      MOV AX,4C00H
      INT 21H
MAIN ENDP
END MAIN

;-----------------------------------------------------------------
------
               ;Program to Calculate Factorial of Given Number
;-----------------------------------------------------------------
------
.MODEL SMALL
.STACK
.DATA
        VAL DW 7

.CODE
;-----------------------------------------------------------------
------
MAIN PROC
      MOV AX,@DATA
      MOV DS,AX  ; Initialize Data Segment

      MOV AX,VAL ; AL is Passed with the Content of DI
      MOV CX,AX  ; Counter to  no given in val
```

```
      DEC CX     ; Decrement the Content of CL to make n-1

      TOP:
            MUL CX         ; Multiply the content of AL and CL
answer in AL
            DEC CX      ; Decrement the content of cl
            JZ LAST     ; Jump if Zero to Last
            JMP TOP     ; JMP to Top Unconditionally

      LAST:
            MOV CX,100
            DIV CX      ; Quotient in AX and Remainder in DX
            AAM         ; Adjust Quotient to Unpacked BCD
            ADD AX,3030H
            MOV BX,AX   ; Prserve the Content of AX to BX
            XCHG AX,DX
            AAM
            ADD AX,3030H
            PUSH AX     ; Preserve the Content of AX into Stack

            ;;;;;;;;;;;;; Display Operation Started
            MOV DL,BH
            MOV AH,02H
            INT 21H
            MOV DL,BL
            MOV AH,02H
            INT 21H     ; Quotient Printing is Finished
            POP AX
            MOV DL,AH
            MOV DH,AL
            MOV AH,02H
```

```
            INT 21H
            MOV DL,DH
            MOV AH,02H
            INT 21H     ; Remainder Printing is Finished
            ;;;;;;;;;;;;; Display Operation Finished

            MOV AX,4C00H
            INT 21H     ; Normal End Request
MAIN ENDP
END MAIN

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

```
;Program to Print the Fibonaci Series.[1 2 3 5 8 13 21 ..]

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

.MODEL SMALL

.STACK

.DATA

        VAL1 DB 0

        VAL2 DB 1

.CODE

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


MAIN PROC

        MOV AX,@DATA

        MOV DS,AX


        MOV AL,VAL1 ; A = VAL1

        MOV BL,VAL2 ; B = VAL2

        MOV CX,10

        MOV AH,00H


        TOP:

                ADD AL,BL ; P= A+B

                MOV BH,BL ; TEMP1=B

                PUSH AX   ; CONTENT OF A IS PUSHED TO STACK


                 ;;;;;;;;;;; DISPLAY OPERATION STARTED

                AAM

                ADD AX,3030H

                MOV DL,AH
```

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

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

                MOV DL,' '
                MOV AH,02H
                INT 21H
                ;;;;;;;;;;; DISPLY OPERATION END

                POP AX    ; CONTENT OF STACK IS POPED TO AX
                MOV BL,AL ; B=P
                MOV AL,BH ; A=B

                LOOP TOP  ; DECREMENT THE CONTENT OF CX BY1
                            ; JUMP TO LABEL TOP UNTIL CX > 0
                MOV AX,4C00H
                INT 21H

MAIN ENDP
END MAIN

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

```
;Program To Demonstrate File Handles For Input and Output
;-----------------------------------------------------------------
------
.MODEL SMALL
.STACK
.DATA
        KEYDISP DB 'Enter Character/Number[10]: $'

        KEYINP  DB 20 DUP(' ') ;Additional 2 Characters For 0DH
& 0CH
                               ;0DH : Enter
                               ;0CH : Line Feed
        KEYOUT  DB 'Output : $'
.CODE
;-----------------------------------------------------------------
------
DISPLAY MACRO A
        LEA DX,A
        MOV AH,09H
        INT 21H
ENDM
;-----------------------------------------------------------------
------
MAIN PROC
        MOV AX,@DATA
        MOV DS,AX
                              ; KEYDISP IS A PARAMETER TO MACRO
        DISPLAY KEYDISP     ; Invoked DISPLAY MACRO
        CALL READ
                              ; KEYOUT IS A PARAMETER TO MACRO
         DISPLAY KEYOUT       ; Invoked DISPLAY MACRO
        MOV CX,20
```

```
        TOP :
                MOV DL,[BX]
                MOV AH,02H
                INT 21H
                INC BX
                DEC CX
                JZ LAST
                JMP TOP
         LAST:
                MOV AX,4C00H
                INT 21H
MAIN ENDP
;-------------------------------------------------------------
------
;READ PROCEDURE FOR KEYBOARD INPUT USING FILE HANDLES
;AH=FUNCTION 3FH
;BX=00H INDICATES INPUT
;CX=MAXIMUM NO OF CHARACTERS TO ACCEPT
;DX=ADDRESS OF AREA FOR ENTERING CHARACTERS
READ PROC
        MOV AH,3FH
        MOV BX,00H
        MOV CX,20
        LEA DX,KEYINP
        INT 21H
        MOV BX,DX
        RET
READ ENDP
;-------------------------------------------------------------
------
END MAIN
```

```
;Program to Print the Input String In Reverse Order.

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

.MODEL SMALL

.STACK

.DATA

.CODE

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


MAIN PROC

      ;;;;;;;;;;;;;;;;;;;;Data Segment Initialization Started


      MOV AX,@DATA

      MOV DS,AX


      ;;;;;;;;;;;;;;;;;;;;Data Segment Initialization Ended



      MOV CX,0                ;Initialize the Counter as 0


      READ_CHAR:

            MOV AH,01H        ;Put AH=01H to Read Character From
Keyboard

            INT 21H           ;Reads Character and Puts it in AL

            CMP AL,0DH        ;0DH is the ASCII Code For Enter
Key

                              ;Check if Enter Key is Pressed


           JE END_OF_LINE   ;If Enter Pressed Go to END_OF_LINE

            PUSH AX           ;Push Character into Stack

            INC CX            ;Increment Counter CX
```

```
            JMP READ_CHAR    ;Unconditional Jump to READ_CHAR


      END_OF_LINE:
            POP DX              ;Pop Character From Stack into DX

            MOV AH,02H          ;Code for Displaying character on
VDU
            INT 21H             ;DOS Interrupt Function
            LOOP END_OF_LINE   ;Loop Until CX=0

            MOV AX,4C00H        ;End Request
            INT 21H             ;Dos Interrrupt Function


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

```
;Program To Clear the Screen With Bios Interrupt Function.
;-----------------------------------------------------------------
------
.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
;-----------------------------------------------------------------
------

;Program to Set the Cursor in Desired Location

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

MAIN PROC
        MOV AX,@DATA
        MOV DS,AX

        ;;;;;;;;;;;;;;;;;;;;
        ;INT 10H Function 02H: Set Cursor Position
        ;AH=Request Cursor with AH=02H
        ;BH=Page Number [Default 00H]
        ;DH=Row
        ;DL=Column
        ;;;;;;;;;;;;;;;;;;;;


        MOV AH,02H ;Request Set Cursor
        MOV BH,00  ;Page Number to 0
        MOV DH,08  ;Row 8
```

```
        MOV DL,50  ;Column 50
        INT 10H    ;Bios Interrupt Function


        MOV DL,'C' ;Character to be Displayed in DL
        MOV AH,02H ;Character Mode
        INT 21H    ;Dos Interrupt Function


        MOV AX,4C00H
        INT 21H
MAIN ENDP
END MAIN
;-------------------------------------------------------------------
-----;Progam to Display Character with Attributes


;-------------------------------------------------------------------
------
.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
;-----------------------------------------------------------------
------



;Program to Demonstrate Screen Scroll Down


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

```
MAIN PROC
        MOV AX,@DATA
        MOV DS,AX

        ;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;INT 10H Function 07H : Scroll Down Screen
        ;AH=Function 07H
        ;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,0702H  ;AH=07,AL=00 for Full Screen
        MOV BH,71H    ;White background(7),Blue foreground(1)
        MOV CX,0C19H  ;Upper left row:column
        MOV DX,1236H  ;lower right row:column
        INT 10H       ;Bios Interrupt Function

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

```
;-------------------------------------------------------------
------
;Program to Plot Pixels in Graphics Mode

;-------------------------------------------------------------
------
.MODEL SMALL
.STACK
.DATA
.CODE
;-------------------------------------------------------------
------
MAIN PROC
        MOV AX,@DATA
        MOV DS,AX

        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;
        ;INT 10H: Get Current Video Mode
        ;AH=0FH
        ;AL=Returns Current Video Mode
        ;AH=Number of Screen Columns
        ;BH=Active Video Page

        MOV AH,0FH
        INT 10H
        PUSH AX ;PUSH the Current Mode to Stack

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

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

```
;INT 10H : Set Video Mode
;AH=00H
;AL=Required Mode
MOV AH,00H
MOV AL,13H
INT 10H
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;INT 10H : Write Pixel Dot
;AH=0CH
;AL=Color of the Pixel
;BH=Page Number
;CX=Column
;DX=ROW
MOV BH,00
MOV DX,10
MOV CX,10
MOV BL,00
TOP:
        MOV AH,0CH
        MOV AL,BL
        INT 10H
        INC DX
        INC BL
        CMP DX,100
        JNE TOP
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
```

```
        ;KEYBOARD INPUT
        ;AH=01H
        ;AL=Returns Key IN ASCII

        MOV AH,01H
        INT 21H
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;



        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;
        ;Return to Previous Graphics Mode

        POP AX
        MOV AH,00H
        INT 10H
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;



        MOV AX,4C00H
        INT 21H

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

```
;Program to Print the String in Reverse Order


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

.MODEL SMALL

.STACK

.DATA

        STRING DB '!EMOC-LEW'

        REVERSE DB 9 DUP(' ')

.CODE

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


MAIN PROC

        MOV AX,@DATA

        MOV DS,AX



         LEA SI,STRING   ;Load Effective Address of STRING into
      SI

        LEA DI,REVERSE ;Load Effective Address of REVERSE into
DI

        ADD DI,9       ;Add the Address of DI With 9

        MOV CX,9   ;Initialize Counter as 9


        TOP:

                MOV AL,[SI] ;Content of SI to AL

                MOV [DI],AL ;Content of AL to Content of DI

                INC SI         ;Increment SI

                DEC DI         ;Decrement DI

                LOOP TOP    ;Loop Until  CX!=0
```

```
        ADD DI,10   ;Add DI With 10 to Locate to End
        MOV AL,'$'    ;Add String Termination Character
        MOV [DI],AL

        MOV AH,09H     ;AH=09 Specifies String
        LEA DX,REVERSE ;Load Effective Address of REVERSE into
DX
        INT 21H    ;DOS Interrupt Function

        MOV AH,4CH
        INT 21H

MAIN ENDP
END MAIN

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

```
;Program To Change The String Into Toggle Case
;-------------------------------------------------------------
------
.MODEL SMALL
.STACK
.DATA
        STRING DB 'Welcome'
        CASE DB 7 DUP(' ')
.CODE
;-------------------------------------------------------------
------
MAIN PROC
```

```
MOV AX,@DATA
MOV DS,AX
LEA SI,STRING ;Load Effective Address of STRING into SI
LEA DI,CASE   ;Load Effective Address of CASE with DI
MOV CX,7   ;Load Counter with 7
TOP:
        CMP CX,0000H ;Compare CX with 0
        JE EXIT        ;If CX=0 then Goto Exit
        MOV AH,[SI]  ;Load Content of SI into AH
         CMP AH,60H   ;Compare AH with 60H i.6 96
        JA ISSMALL
        CMP AH,5AH   ;Compare AH with 5AH i.e 90
        JB ISCAP

        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;TO UPPERCASE
         ISSMALL:
                AND AH,11011111B ;Mask With 11011111B
                MOV [DI],AH
                INC SI
                INC DI
                DEC CX
                JMP TOP
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


       ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;TO LOWERCASE
         ISCAP:
                OR AH,00100000B ;Mask With 00100000B
                MOV [DI],AH
```

```
                        INC SI
                        INC DI
                        DEC CX
                        JMP TOP
                    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
               EXIT:
                      MOV AL,'$'   ;Add String Terminator.
                     MOV [DI],AL  ;Pass the Content of AL to DI.
                     MOV DX,OFFSET CASE
                     MOV AH,09H
                     INT 21H
                     MOV AH,4CH
                     INT 21H
MAIN ENDP
END MAIN
;-------------------------------------------------------------------
------
```

UNIT 6

BASIC I/0, MEMORY R/W AND INTERRUPT OPERATIONS

Direct Memory Access(DMA)

Introduction:

- **DMA is an I/O technique commonly used for high speed data transfer from one location to another. For example: Data transfer between system memory and a pen drive or other storage device.**
- **Or it can also be defined as feature of modern computers that allows certain hardware subsystems within the computer to access system memory independently of the central processing unit (CPU).**
- **In DMA, MPU releases the control of the buses to a device called DMA controller. This DMA controller is a device or say, a processor, capable only of copying data in high speed from one memory location to another.**
- **It uses the following two signals available in 8085 microprocessor:**
    1. **HOLD: DMA sends hold request to 8085 microprocessor requesting the use of address bus and data bus. After receiving the HOLD request, MPU relinquishes *(surrenders)* the buses in the following machine cycle. All buses are tri-stated and HLDA (hold acknowledge) signal is sent out. MPU regains the power of buses after HOLD goes low.**
    2. **HLDA: This is an 8085 output signal indicating that MPU is relinquishing control of buses. DMA controller uses these signals as if it's a peripheral device requesting the control bus. Once the controller has gained control, it acts as processor for data transfer. This process is called switching from slave to master mode.**

- ✓ ***To serve as a data transfer processor, DMA controller should have*:**
    a. **A data bus**
    b. **An address bus**
    c. **Read/Write control signals**
    d. **Control signal to disable its role as a peripheral and to enable it as a processor**

Advantages:

- ➢ **Quick data transfer because a dedicated piece of hardware transfers data from one computer location to another and only one or two bus read/write cycles are required per piece of data transferred.**
- ➢ **Minimizes latency in servicing a data acquisition device because the dedicated hardware responds more quickly than interrupts and transfer time is short.**
- ➢ **Minimizes latency reduces the amount of temporary storage (memory) required on an I/O device.**
- ➢ **Processor is not used for holding the data transfer activity and is available for other processing activity.**
- ➢ **Also in systems where the processor primarily operates out of its cache, data transfer actually occurring in parallel, thus increasing overall system utilization.**

Application:

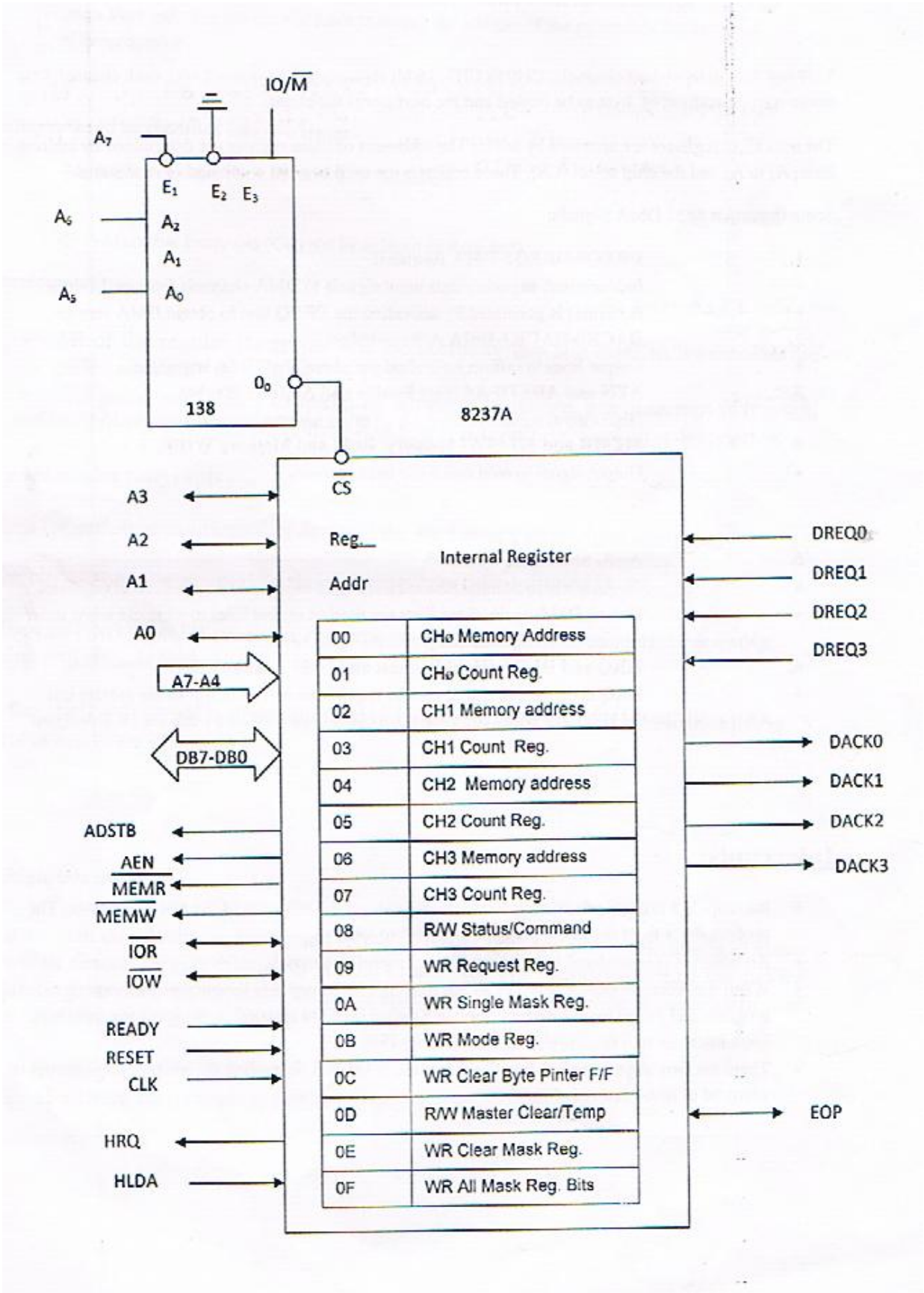
- ✓ **Extensively used for computer-based data acquisition applications including streaming data to disk, real-time screen data display and continuous data acquisition applications.**
- ✓ **DMA was used for floppy disk I/O in the original PC and for hard disk I/O in later versions.**
- ✓ **PC-based DMA technology, along with high-speed bus technology, is driven by data storage, communications and graphics needs-all of which**

require the highest rates of data transfer between system memory and I/O devices.

- ✓ **Data acquisition applications have the same needs and therefore can take advantage of the technology developed for larger markets.**

DMA Controller 8237 Interfacing:

- **Is a programmable DMA controller**
- **40 pin package**
- **It must interface with two types of devices-MPU and peripherals (eg: floppy disk, pen drive, etc)**
- **Following is the logical pin diagram for 8237 DMA controller:**



8237 has four independent channels, CH_0 to CH_3. 16 bit registers are associated with each channel: One stores

starting address of byte to be copied and the next store the count.

The next eight registers are accessed by MPU. The addresses of these register are determined by address lines, A_3 to A_0 and the chip select (CS). These registers are used to write command or read status.

Below are some important DMA Signals:

1. **DREQ0-DREQ3-DMA Request:**
   - **Independent, asynchronous input signals to DMA channels from peripherals**
   - **A request is generated by activating the DREQ line to obtain DMA service**

2. **DACK0-DACK3-DMA Acknowledge:**
   - **Output lines to inform individual peripheral that DMA is granted.**

3. **AEN and DSTB-Address Enable and Address Scribe:**
   - **High output signals to latch a high order address bytes to generate 16-bit address**

4. **MEMR and MEMW – Memory Read and Memory Write:**
   - **Output signal is read and write from memory**

5. **A_3-A_0 and A_7-A_4-Address**
   - **A_3-A_0 are bidirectional address lines used as input to access control registers.**
   - **During DMA cycle, these lines are used as output lines to generate a low order address that combined with the remaining address lines A_7-A_4**
6. **HRQ and HLDA (Hold Request and Hold Acknowledge):**

- **HRQ is the output signal used to request the MPU control of the system bus.**
- **After receiving the HRQ, the MPU completes bus cycle in process and issue the HLDA signal.**

System Interface

The DMA is used to transfer data bytes between I/O (such as floppy disk) and system memory (or from memory to memory) at high speed. It includes eight data lines, four control signals ($\overline{IOR}$, $\overline{IOW}$, $\overline{MEMR}$, and $\overline{MEMW}$), and eight address lines (A_7-A_0). However, it needs 16 address lines to access 64K bytes. Therefore, an additional eight lines must be generated as shown in figure 15.34.

When a transfer begins, the DMA places the low-order byte on the address bus and the high-order byte on the data bus and asserts AEN (Address Enable) and ADSTB (Address Strobe). These two signals are used to latch the high-order byte from the data bus: thus, it places the 16-bit address on the system bus. After the transfer of first byte, the latch is updated when the lower byte generates a carry (or borrow). Figure 15.34 shows two latches: one latch (373 #1) to latch a high-order address from the data bus by using the AEN and ADSTB signals, and the second latch (373 #2) to demultiplex the 8085 bus and generate the low-order address bus by using the ALE (Address Latch Enable from the 8085) signal. The $\overline{AEN}$ signal is connected to the OE signal of the second latch to disable the low-order address bus from the 8085 when the first latch is enabled to latch the high-order byte of address.

Programming The 8237

To implement the DMA transfer, the 8237 should be initialized by writing into various control registers discussed earlier in the DMA

channels and interfacing section. To initialize the 8237, the following steps are necessary:

1. **Write a control word in the Mode register that selects the channel and specifies the type of transfer (Read, Write or Verify) and the DMA mode (block, single-byte, etc.).**

2. **Write a control word in the Command Register that specifies parameters such as priority among four channels. DREQ and DACK active levels, and timing, and enables the 8237.**
3. **Write the starting address of the data block to be transferred in the channel Memory Address Register (MAR).**
4. **Write the count (the number of the bytes in the data block) in the channel Count register.**

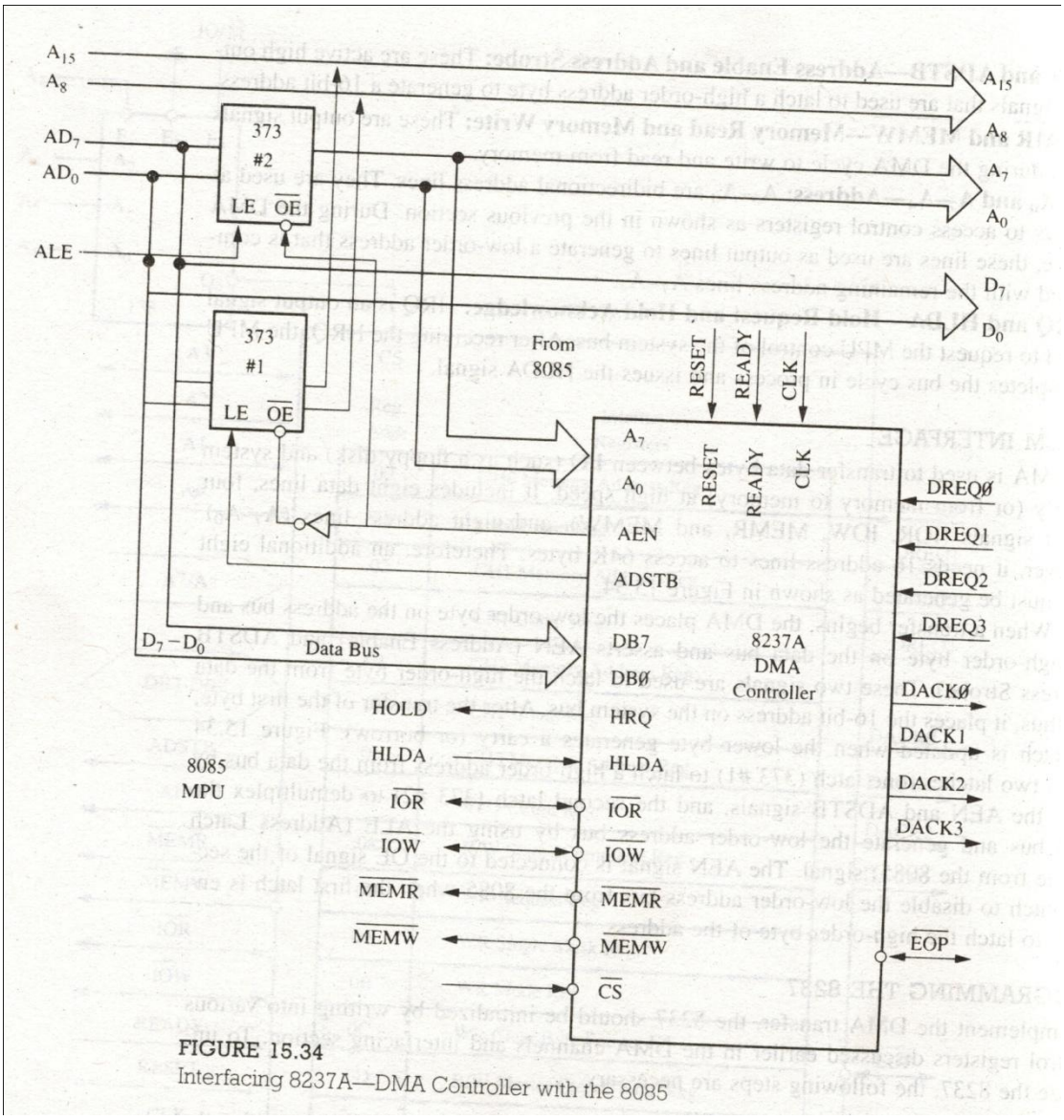


FIGURE 15.34
Interfacing 8237A—DMA Controller with the 8085

These steps are illustrated in the following example:

Example 15.8

Write initialization instructions for the DMA controller in the Figure 15.33 to meet the following specifications. Use the same port (register) address (00 to 0FH) as in Figure 15.33.

1. **Disable the DMA controller and begin writing initialization instructions.**
2. **Initialize Channel #3 (CH3) to transfer 1K of bytes from the system memory to the floppy disk assigned to CH3.**
3. **The starting address of the data block is 4075H and subsequent data bytes have memory addressess in increasing order**
4. **The command parameters should be normal timing, fixed piority late write. DREQ and DACK are both active low.**
5. **Set up the demand mode whereby the DMA can complete the data transfer without any interruption.**

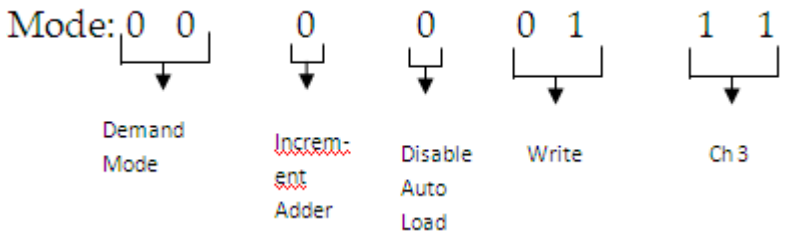
The initialization instructions to set up the DMA controller are as follows:

MVI A, 00000111B ;Command: 0 0 0 0 0 1 0 0

Disable DMA

OUT 08H ;Send to Command Register

MVI A, 00000111B ;



OUT 0BH **;Send to Mod Reg.**

MVI A, 75H **;Low-order byte of starting address**

OUT 06H **;Output to CH3 Memory Address Register**

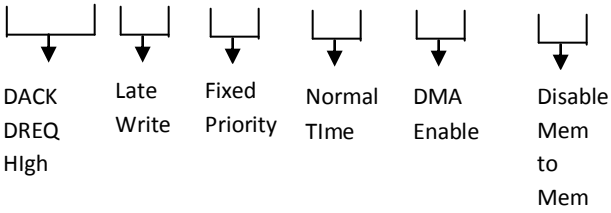
MVI A, 40H **;High-order byte of starting address**

OUT 06H **;Output to CH3 Memory Address Register**

MVI A, FFH **;Low-order byte of the count 03FFH**

OUT 07H **;Output to CH3 Count Register**

MVI A, 10000000B **;Command:** **1 0 0 0 0 0 0 0**



OUT 07H **;Send to Command Flag**

DMA EXECUTION

The process of data transfer from the peripheral to the system memory under the DMA controller can be classified under two modes: the slave mode and the master mode.

Slave Mode: In the slave mode, the DMA controller is treated as peripheral, using the following steps:

1. **The MPU selects the DMU controller through Chip Select.**

2. **The MPU writes the control words as illustrated in the Example 15.8 in channel registers and command/status registers by using control signals $\overline{IOW}$ and $\overline{IOR}$.**

Master Mode: After the initialization, the 8237 in the master mode keeps checking for a DMA request and the steps in the data transfer can be listed as follows:

1. **When the pheripheral is ready it send high signal to DRQ.**
2. **When the DRQ has been received and the channel enabled, the control logic sets HRQ (Hold Request) high. (HRQ is connected to the HOLD signal of the 8085.)**
3. **In the next cycle, the MPU relinquishes the buses and send the HLDA (Hold Acknowledge) signal to 8237.**
4. **After receiving the HLDA signal, the DMA asserts AEN (Address Enable) signal high. The high AEN signal diables 373 Latch #2, thus disconnecting the demultiplexed bus A_7-A_0 of the MPU and enables 373 Latch #1 through an inverter. Next, the DMA asserts ADSTB (Address Strobe) high that is connected to Latch Enable (LE) of 373 Latch #1 and places the contents of data bus, which is a high-order byte of the starting address, on A_{15}-A_8. At the same time, the DMA also outputs the low order address A_7-A_0 on the low-order address bus.**
5. **When the entire address A_{15}-A_0 is available on the address bus, the DMA sends DACK to the peripheral.**
6. **The DMA controller continues the data transfer by asserting the necessary control signals ($\overline{IOR}$, $\overline{IOW}$, $\overline{MEMR}$, and $\overline{MEMW}$) until DACK remains high.**
7. **At the end of the data transfer, the DMA asserts $\overline{EOP}$ (End of Process) signal low that can be used to inform the peripheral that the data transfer is complete. The DMA data transfer can also be terminated by sending a low signal to $\overline{EOP}$ from outside.**

Interrupt:

An interrupt is a signal that a peripheral board sends to the central processor in order to request attention. In response to an interrupt, the processor stops what it is currently doing and executes a service routine. When the execution of the service routine is terminated, the original process may resume its previous operation. The interrupt is initiated by an external device and is asynchronous, meaning that it can be initiated at any time without reference to system clock. However, the response to an interrupt request is directed or controlled by the microprocessor.

Interrupts are primarily issued on:

- ✓ **initiation of I/O operation.**
- ✓ **completion of I/O operation.**
- ✓ **occurrence of hardware or software errors.**

Process of interrupt operation:

1. The I/O unit issues an interrupt signal to the processor. An interrupt signal from I/O is the request for exchange of data with the processor.

2. The processor finishes execution of the current instruction before responding to the interrupt.

3. The processor testes for an interrupt, determines that there is one, and sends an acknowledgement signal to the device that issued the interrupt. After receiving this acknowledgement, the device retains its interrupts signal.

4. The processor now begins to transfer the control to the routine which serves the interrupt request from the device. This routine is

called 'Interrupt service routine' and it resides at a specified memory location.

For this process, the CPU needs to save information needed to reassume the current program at the point of interrupt. The minimum information required is i) the status of the processor, which contained by the processor status word (PSW) and ii) the location of the next instruction to be executed which is contained by the program counter (PC), these all are pushed onto the stack.

5. The processor then loads the program counter with the entry location of the interrupt service routine that will respond to this interrupt. Once the program counter has been loaded, the control is transferred to the interrupt handler program.

6. The fundamental requirement of the interrupt service routine is that it should begin by saving the contents of all the registers on the stack(as state of the main program should be safe). Suppose the user program is interrupted after the instruction at location N. The contents of all the registers plus the address of the next instruction are saved on the stack. The stack pointed is updated and the programs counter is updated to point to the beginning of the interrupt service routine.

7. The interrupt handler now proceeds to process the interrupt. This will include an examination of status information relating to the I/O operation or the other event that caused an interrupt. It may also involve sending additional commands or acknowledgement to the I/O unit.

8. When interrupt processing is complete the saved register's value (of the main program) are retrieved from the stack and restored to the register.

9. The final function is to restore the PSW and program counter values from the stack. As a result the next instruction to be executed will be from the previously interrupted main program.

Types of interrupt:

There are three major types of interrupts that cause a break in the normal execution of a program. They can be classified as:

1. External interrupts

2. Internal interrupts

3. Software interrupts.

External interrupts: External interrupts are initiated via the microprocessor's interrupt pins by external devices such I/O devices, timing device, circuit monitoring the power supply etc. Causes of these interrupts may be; I/O device requesting transfer of data, I/O device finished transfer of data, elasped time of an event, or power failure. Timeout interrupt may result from a program that is an endless loop and thus exceeded its time allocation. Power failure interrupt may have as its service routine a program that transfers the complete state of the CPU into a non-destructive memory in few milliseconds before power ceases.

External interrupts can be further divided into two types:

I. Maskable interrupt.

II. Non-maskable interrupt.

Maskable interrupt: A maskable interrupt is one which cab be enabled or disabled by executing instructions such as EI (enable interrupts)and DI (Disable interrupt). If the microprocessor's 'interrupt enable flip flop' is disabled, it ignores a maskable interrupt.

In 8085, the 1 byte instruction EI sets the interrupt enable flip flop and enables the interrupt process. Similarly the 1 byte instruction DI resets the interrupt enable flip flop and disables the interrupt process. No maskable interrupts are recognized by the processor when the interrupt is disabled.

Non maskable interrupt: This type of interrupt cannot be enabled or disabled by instructions. This types has higher priority over the maskable interrupt. This means that if both the makable and non maskable interrupts are activated at the same time, then the processor will service the non-maskable interrupt first. In 8085 TRAP is an example of non maskable interrupt.

Internal interrupt: Internal interrupt arise form illegal or erroneous use of an instruction or data. Cause of this interrupt may be: register overflow, attempt to divide by zero, an invalid operation code, stack overflow etc. These error conditions usually occur as a result of premature
termination of the instruction execution. These are even termed as exceptions.

The difference between internal and external interrupt is that the internal interrupt is initiated by some exceptional conditions caused by the program itself rather than by an external events. Internal interrupts are synchronous with the program, while external interrupts are asynchronous. If the program is return, the internal interrupt will occur in the same place each time. External interrupts depends on external conditions that are independent of the program being executed at the time.

Software interrupt: **External and internal interrupts are initiated from signal that occur in the hardware of the cpu. A software interrupt is initiated by executing an instruction. Software interrupt is a special call instruction that behaves like an interrupt rather than a subroutine call. It can be used by the programmer to initiate an interrupt procedure at any desired point in the program. The most common use of software interrupt is associated with a supervisor call instruction. This instruction provides means for switching from a CPU user mode to the supervisor mode. Certain operations in the computer may be assigned to thesupervisor mode only, as for example, a complex input oroutput transfer procedure. In 8085 the instruction like RST0,RST1, RST2, RST3…..etc. causes a software interrupt.**

Interrupt priority: **Data transfer between the CPU and an I/O device is initiated by the CPU. However , the CPU cannot start the transfer unless the device is ready to communicate with the CPU. The readiness of the device can be determined from an interrupt signal. The CPU responds to the interrupt request by storing the return address form PC into a memory stack and then the program branches to a service routine that process the required transfer. In micro-computer a number of I/O device are attached to the processor, with each device being able to originate an interrupt request. The first task of the interrupt system is to indemnify the source of the interrupt. There is also the possibility that several sources will request service simultaneously. In this case the system must also decide which device to service first. An interrupt priority is a system that established a priority over the various sources to**

determine which condition is to be serviced first when two or more request arrive simultaneously. The system may also determine which conditions are permitted to interrupt the computer while another interrupt is being serviced. Higher-priority interrupt levels are assigned to request which, if delayed or interrupted, could have serious consequences. Device with higher speed transfers such as magnetic disks are given high priority, and slow devices such as keyboards receive low priority. When two devices terrupt the processor at the same time, the processor service the device, with the higher priority first.

There are mainly two ways of servicing multiple interrupts.

These are:

1. Polled interrupt.

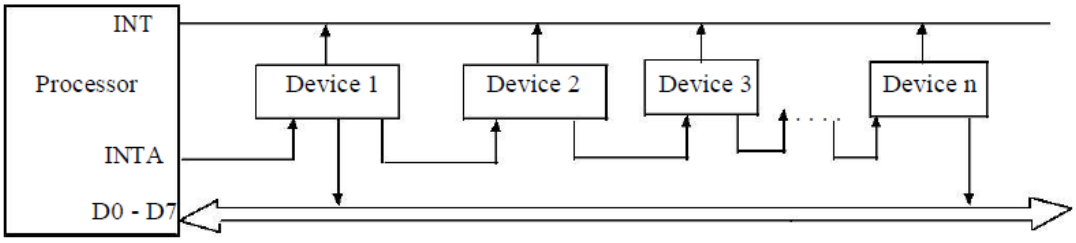
2. Chained (Vectored) interrupt.

Polled interrupt: Polled interrupt are handled using software and are therefore slower compared to vectored (hardware) interrupts. In this method there is one common branch address for all interrupts. The program that takes care of interrupts begins at the branch address and polls the interrupts sources in sequence. The order in which they are tested determines the priority of each interrupt. The highest priority source is tested first, and if its interrupt signal is on, control branches to a service routine for this source. Other wise the next lower priority source is tested, and so on. Thus, the initial service routine for all interrupts consists of a program that test the interrupt sources in sequence and branches to one of many possible service routines.

Polled interrupts are very simple . But or large number of devices, the time required to poll each device may exceed the to service

the device. In such case, the faster mechanism called chained interrupt is used.

Chained interrupt: This is hardware concept of handling the multiple interrupts. In this technique, the devices are connected in a chain fashion as shown in figure below for setting up the priority system.



Here the device with he highest priority placed in the first position, followed by lower priority devices. Suppose that one or more devices interrupt the processor at a time. In response, the processor saves its current status and then generates an interrupt acknowledge (INTA) signal to the highest priority device, which is device 1 in our case. If this device has generated the interrupt it will accept the INTA signal from the processor; otherwise, it will pass INTA on to
the next device until the INTA is accepted by the interrupting device.

Once accepted, the device provides a means to the processor for finding the interrupt address vector using external hardware. Usually the requesting device responds by placing a word on the data lines. With the help of hardware it generates interrupts vector address. This word is referred to as vector, which the processor used as a pointer to the appropriate device service routine.

This avoids the need to execute a general interrupt service routine first. So this technique is also referred to as vectored interrupts.

Interrupts of 8085: The 8085 has five interrupts:

i. TRAP

ii. RST 7.5

iii. RST 6.5

iv. RST 5.5

v. INTR

The four interrupts TRAP, RST 7.5, 6.5,5.5 are automatically vectored (transferred) to specific locations on without any external hardware. They do not require INTA signal or an input port; the necessary hardware is already implemented inside the 8085. These interrupts and their call

locations are:

INTERRUPT	CALL LOCATIONS
TRAP	0024 H
RST 7.5	003 CH
RST 6.5	0034 H
RST 5.5	002C H.

The TRAP has the highest, followed by RST 7.5, 6.5, 5.5 and INTR. Figure below shows the schematic diagram of 8085 interrupts.

INTR:

This interrupt is maskable. It can be enabled by instruction EI and can be disabled by instruction DI. The INTR interrupt requires external hardware to transfer program sequence to specific CALL locations. There are 8 numbers of CALL-Locations for INTR interrupt. The hardware circuit generate RST codes for this purpose and places that on the data bus externally.

When the microprocessor is executing a program, it checks the INTR line (when interrupt enable flip flop is enabled using EI instruction) during the execution of each instruction. If the line is high and the interrupt is enabled, the microprocessor completes the currents instruction, disabled the interrupt enable flip flop and sends a INTA signal. The processor does not accept any interrupt requests until the interrupt flip flop is enabled again.

The signal INTA is used to insert a Restart (RST) instruction, (it saves the memory address of the next instruction to the stack. The program is transferred to the cal location.). The RST instruction and their call locations are :

Instruction	Hex-code	Call location
RST 0	C7	0000
RST 1	CF	0008
RST 2	D7	0010
RST 3	DF	0018
RST 4	E7	0020
RST 5	EF	0028
RST 6	F7	0030
RST 7	FF	0038

-bit D3 is a control bit and should be 1 for bits D0 , D1, and D2 to be effective.

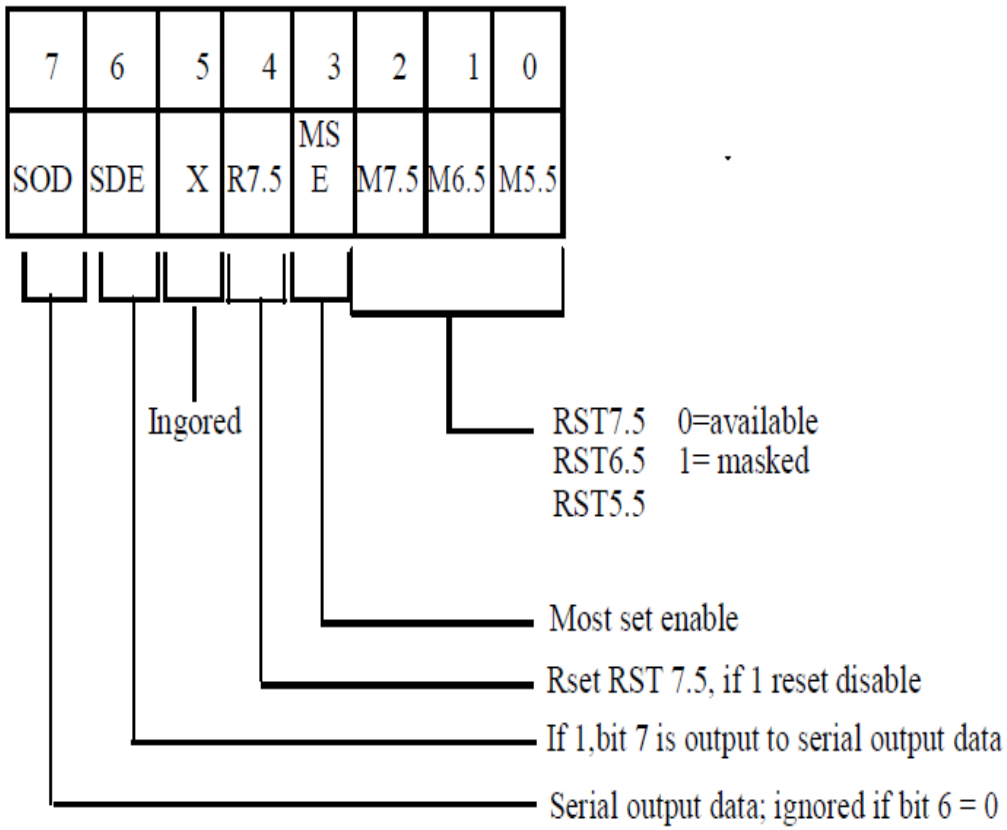
-Logic 0 on D0, D1, and D2 will enable the corresponding interrupts and logic 1 will disable the

interrupts.

ii) The second function is to reset RST 7.5 flip flop. Bit D4 is additional control for RST 7.5

-If D4 =1, RST 7.5 is reset. This is used to ignore RST 7.5 without servicing it.

iii) The third function is to implement serial I/O. Bit D7 and D6 are used for serial I/O and do not effect the interrupts.



Assuming that the task to be performed is written as a subroutine at the specified location the processor performs a task. This service routine includes the instruction EI to enable the interrupt again and RE-instruction to retrieve the memory address where the program has interrupted. Then the execution goes to the main program again.

TRAP:

It is a non maskable interrupt. It has the highest priority among the interrupt signal. It need not be enabled and it cannot be disabled. When this interrupt is triggered the program control is transferred to the location 0024 H without any external hardware or the interrupt enable instruction. TRAP is generally used for such critical events as power failure and emergency shut off. RST 7.5, 6.5, 5.5: These interrupts are maskable and are enabled by software using instructions EI and SIM (set interrupt mask). The execution of the instruction SIM enables/disables the interrupts according to the bit pattern of the accumulator.

SET Interrupt Mask (SIM) instruction:

-This is 1 byte instruction.

-Can be used for three different functions:

i). One function is set mask for RST 7.5, 6.6 and 5.5 interrupts. This instruction reads the content of the accumulator and enables or disables the interrupts.

Pending interrupts:

When one interrupt request is being served, other interrupt may occur resulting in a pending request. When more than one interrupts occur. Simultaneously the interrupts having higher priority is served and the interrupts with lower priority remain pending. The 8085 has an instruction RIM using which the programmer can know the current status of pending interrupts. This instructions gives the current status of only maskable interrupts.

Instruction RIM:

- Read interrupt Mask.

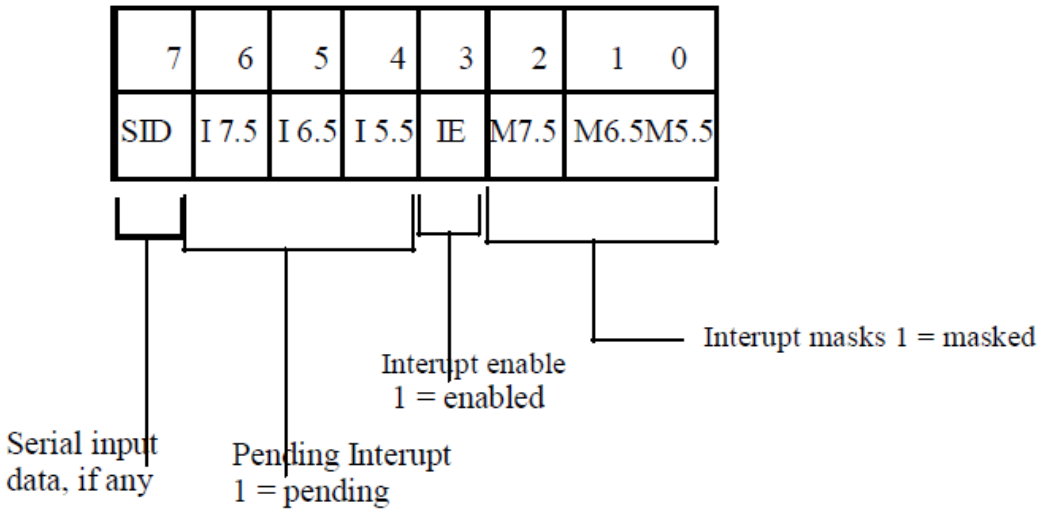
- 1 byte instruction.

- Can be used for the followings.

a. To read interrupt mask. This instruction loads the accumulator with 8-bits indicating the current status of the interrupts.

b. To identify the pending interrupts. Bits D4, D5, and D6 identify the pending interrupts .

c. To receive serial data. Bit D7 is used to receive serial data.



THE 8259A PROGRAMMABLE INTERRUPT CONTROLLER

The 8259A programmable interrupt controller designed to work with Intel microprocessors 8085, 8086 and 8088. The 8259A interrupt controller can

1. Manage eight interrupts according to the instructions written into its control registers. This is equivalent to proving eight interrupt pins on the processor in place of one INTR (8085) pin.

2. Vector can interrupt request anywhere in the memory map. However, all eight interrupts are spaced at the interval of either four or eight locations. This eliminates all the major drawback of the 8085 interrupts in which all interrupts are vectored to memory locations on page 00H

3. Resolve eight levels of interrupt priorities in a variety of modes, such as fully nested mode, automatic rotation mode, and specific rotation mode.

4. Mask each interrupt request individually.

5. Read the status of pending interrupts, in-service interrupts, and masked interrupts.

6. Be set up to accept either the level-triggered or the edge-triggered interrupt request

7. Be expanded to 64 priority levels by cascading additional 8259As.

8. Be set up to work with either the 8085 microprocessor mode or the 886/8088 microprocessor mode.

The 8259A is upward-compatible with its predecessor, the 8259. The main difference between the two is that the 8259A can be used with Intel's 8086/88 16-bit microprocessor. It also includes additional features such as the level-triggered mode, buffered mode, and automatic-end-of interrupt mode. To simplify the explanation of the 8259A, illustrative examples will not include the cascade mode or the 8086/88 mode and will be limited to modes continuously used with the 8085.

15.5.1 BLOCK DIAGRAM OF THE 8259A

Figure 15.29 shows the internal block diagram of the 8259A. It includes eight blocks: control logic, Read/Write logic, data bus buffer, three registers (IRR, ISR and IMR), priority resolver, and cascade buffer. This diagram shows all the elements of a programmable device, plus additional blocks. The functions of some of these blocks need explanation, which is given below:

READ/WRITE LOGIC

This is a typical Read/Write control logic. When the address line A_0 is at logic 0, the controller is selected to write a command or read a status. The Chip Select logic and A_0 determine the port address of the controller.

CONTROL LOGIC

This block has two pints: INT (Interrupt) as an output, and $\overline{\text{INTA}}$ (Interrupt Acknowledge) as an input. The INT is connected to the interrupt pin of the MPU. Whenever a valid

interrupt is asserted, this signal goes high. The $\overline{\text{INTA}}$ in the Interrupt Acknowledge signal from the MPU.

INTERRUPT REGISTERS AND PRIORITY RESOLVER

The interrupt Request Register (IRR) has eight input lines (IR_0-IR_7) for the interrupts. When these lines go high, the requests are stored in the register. The In-Service Register (ISR) stores all the levels that are currently being serviced, and the Interrupt Mask Register (IMR) stored the masking bits of the interrupt lines to be masked. The Priority Resolver (PR) examines these three registers and determines whether INT should be sent to the MPU.

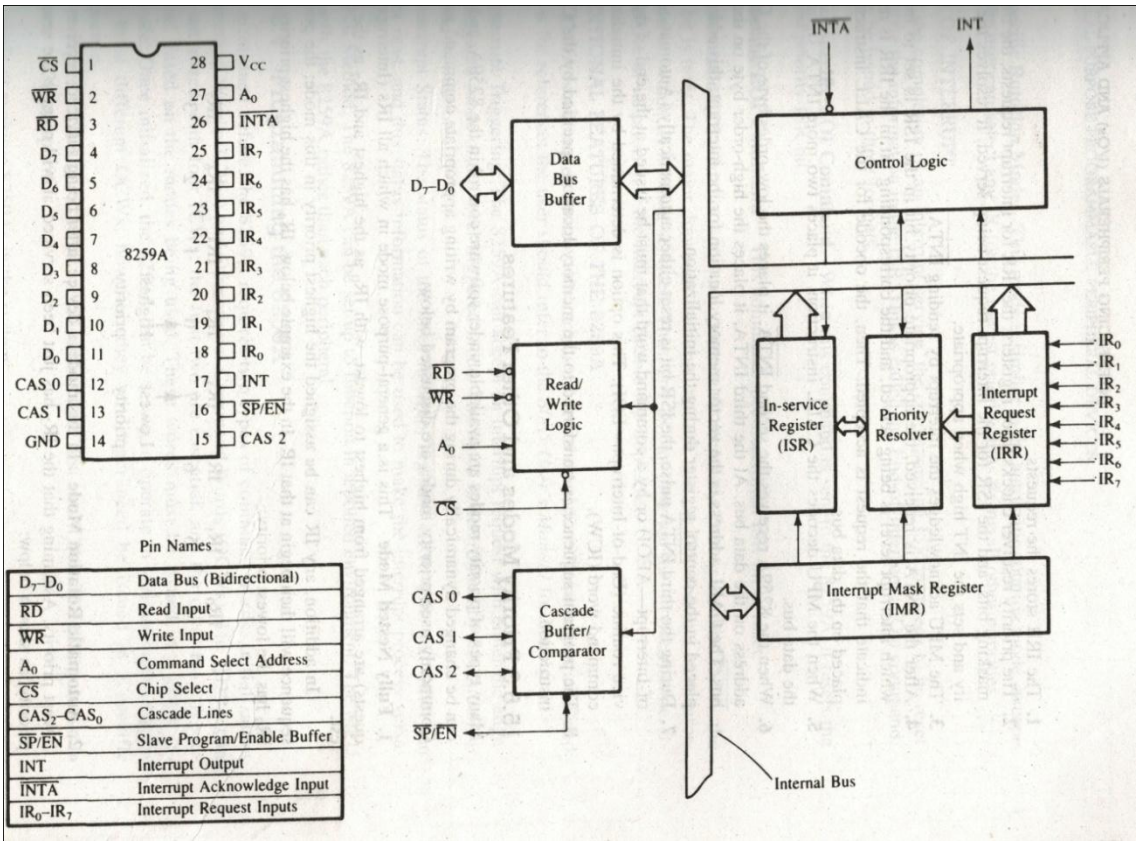
CASCADE BUFFER/COMPARATOR

This block is used to expand the number of interrupt levels by cascading two or more 8259As. To simplify this discussion, this block will not be mentioned again.

15.5.2 Interrupt Operation

To implement interrupts, the Interrupt Enable flip-flop in the microprocessor should be enabled by writing the EI instruction, and the 8259A should be initialized by writing control words in the control register. The 8259A requires two types of control words: Initialization Command Words (ICWs) and Operational Command Words (OCWs). The ICWs are used to set up the proper conditions and specify RST vector addresses. The OCWs are used to perform functions such as masking interrupts, setting up status-read operations, etc. After the 8259A is initialized, the following sequence of events occurs when one or more interrupt request lines go high:

Pin Configuration Block Diagram



1. The IRR stores the requests.
2. The priority resolver checks three registers: the IRR for interrupt requests, the IMR for masking bits, and the ISR for the interrupt request being served. It resolves the priority and sets the INT high when appropriate.
3. The MPU acknowledges the interrupt by sending $\overline{\text{INTA}}$.
4. After the $\overline{\text{INTA}}$ is received, the appropriate priority bit in the ISR is set to indicate which interrupt level is being served, and the corresponding bit in the IRR is reset to indicate that the

request is accepted. Then, the opcode for the CALL instruction is placed on the data bus.

5. When the MPU decodes the CALL instruction, it places two or more $\overline{\text{INTA}}$ signals on the data bus.
6. When 8259A receives the second $\overline{\text{INTA}}$, it places the low-order byte of the CALL address on the data bus. At the third $\overline{\text{INTA}}$, it places the high-order byte on the data bus. The CALL address is the vector memory location for the interrupt: this address is placed in the control register during the initialization.
7. During the third $\overline{\text{INTA}}$ pulse, the ISR bit is reset either automatically (Automatic-End-of-Interrupt—AEOI) or by a command word that must be issued at the end of the service routine (End-of-Interrupt—EOI). This option is determined by the initialization command word (ICW).
8. The program sequence is transferred to the memory location specified by the CALL instruction.

15.5.3 Priority Modes and Other Features

Many types of priority modes are available under software control in the 8259A, and they can be changed dynamically during the program by writing appropriate command words. Commonly used priority modes are discussed below:

1. Fully Nested Mode: This is a general-purpose mode in which all IRS (interrupt Requests) are arranged from highest to lowest, with IR_0 as the highest and IR_7 as the lowest.

In addition, any IR can be assigned the highest priority in this mode; the priority sequence will then begin at that IR. In the example below, IR_4 has the highest priority, and IR_3 has the lowest priority:

IR_0	IR_1	IR_2	IR_3	IR_4	IR_2	IR_3	IR_4
4	5	6	7	0	1	2	3
			↑	↑			
			Lowest Priority	Highest Priority			

2. Automatic Rotation Mode: In this mode, a device, after being serviced, receives the lowest priority. Assuming that the IR_2 has just been serviced, it will receive the seventh priority, as shown below:

IR_0	IR_1	IR_2	IR_3	IR_4	IR_2	IR_3	IR_4
1	6	7	0	1	2	3	4

3. Specific Rotation Mode: This mode is similar to the automatic rotation mode, except that the user can select any IR for the lowest priority, thus fixing all other priorities.

END OF INTERRUPT

After the completion of an interrupt service, the corresponding ISR bit needs to be reset to update the information in the ISR. This is called the End-of-Interrupt (EOI) command. It can be issued in three formats.

1. Nonspecific EOI Command When this command is sent to 8259A, it resets the highest priority ISR bit.

2. Specific EOI Command This command specifies which ISR bit to reset.

3. Automatic EOI In this mode, no command is necessary. During the third $\overline{\text{INTA}}$, the ISR bit is reset. The major drawback with this mode is that the ISR does not have information on which IR is being serviced. Thus, any IR can interrupt the service routine irrespective of its priority, if the Interrupt Enable flip-flop is set.

ADDITIONAL FEATURES OF THE 8259A

The 8259A is a complex device with various modes of operation. These modes are listed below for reference; the user should refer to the 8085 User's manual for details.

- Interrupt Triggering: The 8259A can accept an interrupt request with either the edge triggered mode or the level-triggered mode. This mode is determined by the initialization instructions.

- Interrupt Status: The status of the three instruction registers (IRR, ISR and IMR) can be read, and this status information can be used to make the interrupt process versatile.

- Poll Method: The 8259A can be set up to function in a polled environment. The MPU polls the 8259A rather than each peripheral.

UNIT 7

INPUT/OUTPUT INTERFACES

Methods of Communication

Parallel Communication

Parallel Data Transfer: When a word of n bits is to be transmitted in parallel each bit is transmitted on a separate line along with a common ground line with respect to which the status of each line is measured. Thus, a channel comprises of (n+1) lines.

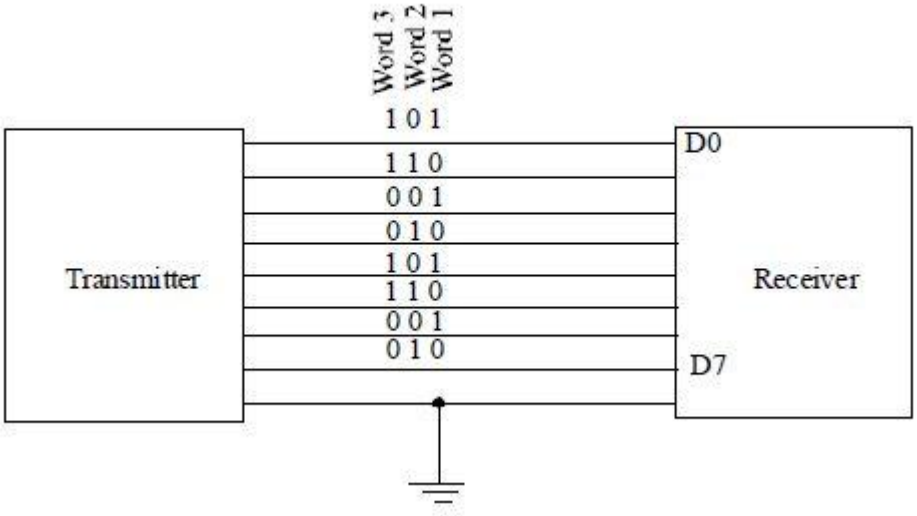


Fig. Parallel Transfer data transfer

Here, the time required to transfer one word is equal to the time taken to transmit a bit. Parallel data transmission is impractical over long distances because of prohibitive cost of installing a large number of lines.

Serial Communication

In serial data transfer, each bit of the word is sent in succession, one at a time over a single pair of wires. A parallel to serial converters is

used to convert the incoming parallel data to serial form and then the data is sent out with the lest significant bit Do first and most significant bit D7 coming last of all. If the bit rate is retained after the parallel to serial conversion, the time taken to transmit a word in serial data transmission will be n times more than the time taken in parallel data transmission. If the word in the above example were to sent serially, the data on the channel will appear as in figure below.

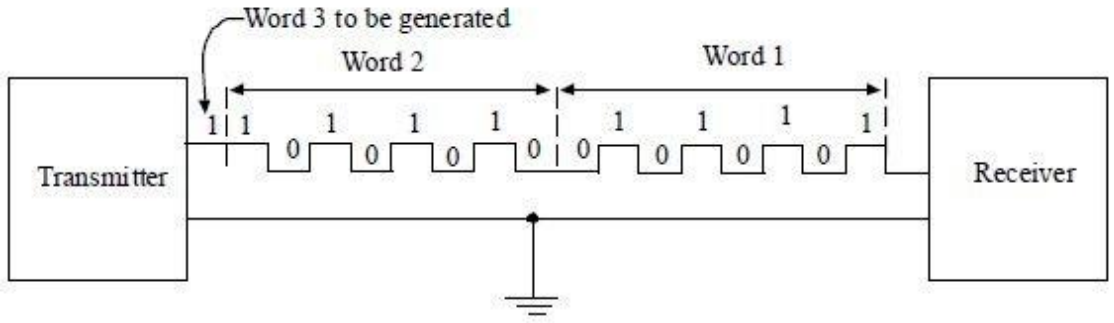


Fig. Serial Data Transfer

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.

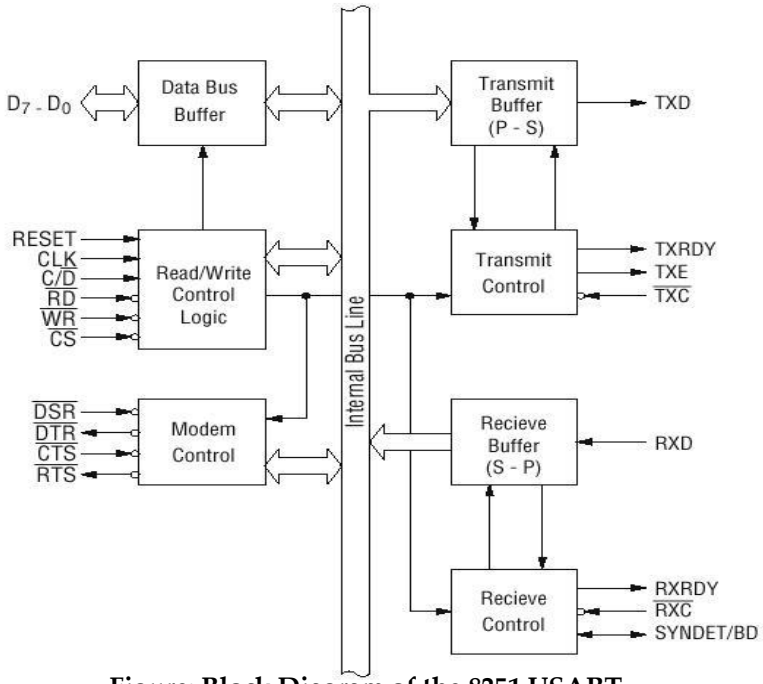


Figure: Block Diagram of the 8251 USART

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

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 "m ode instruction."

Items set by m ode instruction are as follows:

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

The bit configuration of m ode instruction is shown in Figures 2 and 3. I n the case of synchronous m ode, it is necessary t o write one- or two byte sync characters. I f sync characters were written, a function will be set because the writing of sync characters constitutes part of m ode instruction.

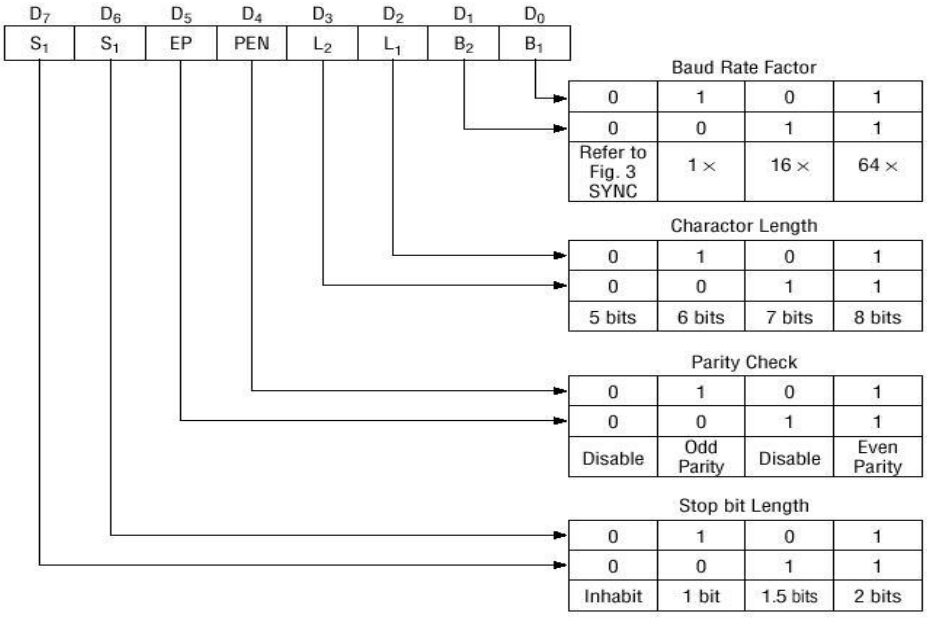


Fig. 2 Bit Configuration of Mode Instruction (Asynchronous)

2. COMMAND 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.

I tem s to be set by com m and 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)

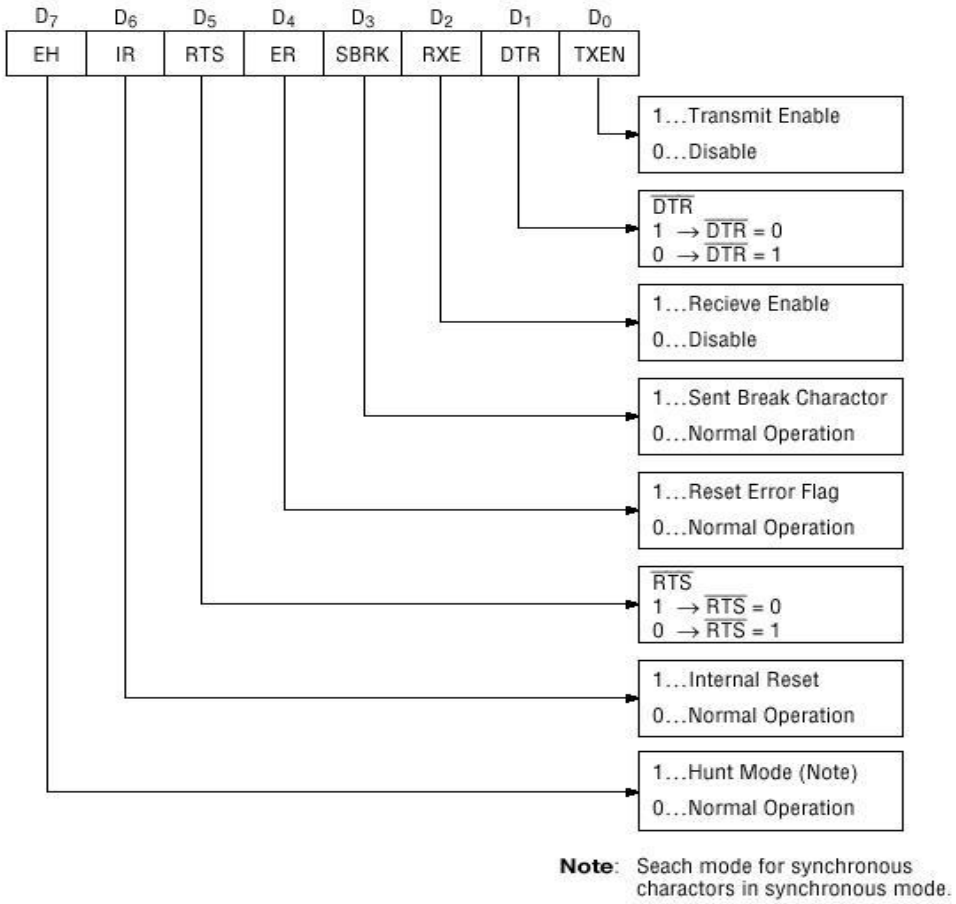


Fig. 4 Bit Configuration of Command

3. STATUS W ORD

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.

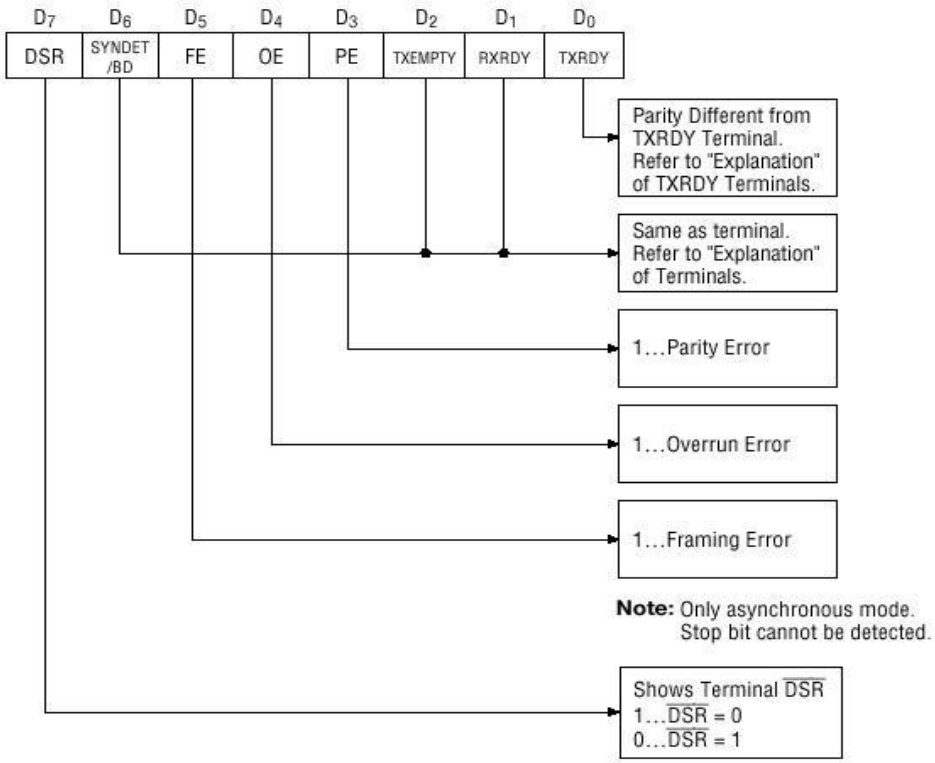


Fig. 5 Bit Configuration of Status Word

PIN DESCRIPTION

D_0 to D_7 [I/O TERMINAL]

This is bidirectional data bus which receives control words and transmits its 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 writing of "mode instruction". The m in reset width is six clock inputs during the operating status of CLK.

CLK [INPUT TERMINAL]

CLK signal is used t o generate internal device tim ing. CLK signal is independent of RXC or TXC. However, the frequency of CLK m ust 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" terminal which receives a signal for writing transmitted data and control words from the CPU into the 8251.

RD [INPUT TERMINAL]

This is the "active low" terminal which receives a signal for reading transmitted data and control words from the CPU into 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 8251 is 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 status" by a command

Note: TXRDY status word indicates disable and that transmit data character is receiveable of CTS or command.

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 transmitted data 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 com m and, 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 m ode," the baud rate will be the same as the frequency of TXC. I n "asynchronous m ode", it is possible to select the baud rate factor by m ode inst ruction. I t can be 1, 1/ 16 or 1/ 64 the TXC. The falling edge of TXC sifts t he 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. I f 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 RXC "synchronous mode" the baud rate is the same as the frequency of "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 t he 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 [I NPUT TERMINAL]

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

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. I t is possible to set the status RTS by a command.

PROGRAMMING 8251A

- ✓ According to the datasheet, the 8251A requires a worst case recovery time of 16 cycles.
- ✓ The initialization sequence of 8251A is hence lengthy.
- ✓ The second reason is that, the 8251A does not respond correctly to a hardware reset.
- ✓ This means that you have to delay 16 processor clock cycles on power up.
- ✓ Therefore a series of software command must be sent to the device to **t**he 8251A distinguishes a command word from a mode word by the order in which make sure it reset properly before the desired mode and command word are sent.
- ✓ They 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 > 1 Clock Cycles
LOOP Last > 5 Clock Cycles

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

MOV  DX,  0FFF2H  ;  Com  m  and  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

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

There are two types of serial transfers. They are:

1. Asynchronous serial data transfer.

2. Synchronous data transfer.

Asynchronous transfer:

In this type of transmission, the receiving device does not need to be synchronized with the transmitting device. The transmitting device can send one or more data units when it is ready to send. Each data unit must be formatted. In other words, each data unit must contain 'a start bit' and 'stop bit (or bits)' indicating the beginning and the end of each data unit. In asynchronous transmission the data message is sent one word at a time.

-When no datas are sent over the time it is maintained at an idle value; a logic 1.

-Start bit is a logic 0.

-The stop bit is a logical 1.

Both the transmitter and receiver are given separate clocking signals. It is not essential that they both be of exactly the same frequency. The data is sent out of the transmitter synchronous with its own clock input and the data is similarly received and assembled at the receiver.

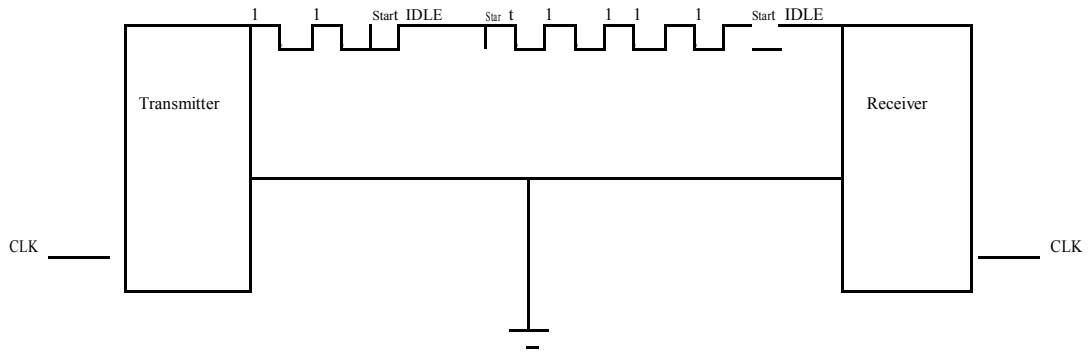


Fig. Asynchronous communication.

When a character is to be transmitted, the transmitter first sends out a low bit. This transition is perceived by the receiver and it gets ready to receive the data. The transmitter then sends out the word bit by bit, one after the other, synchronous with its clock. The receiver assembles the data one by one synchronous with its clock. The receiver must known beforehand the transmission 'Baud Rate' (bits per seconds) for proper assembling at its end. After all the bit of a character are sent, the transmitter sends out a stop bit which is the idle value (logic 1) to indicate the end of transmission.

Asynchronous communication is a start-stop type of communication and is used where the source of data may not be providing a steady stream of new characters. The data thus comes to the receiver at unevenly spaced intervals without reference to a master clock, hence the name Asynchronous.

Asynchronous transmission: Synchronous communication is used for transferring large amount of data at a stretch without frequent start or stops. In synchronous systems to , the line is maintained at the idle value when no data is being transmitted. The transmission begins with a block header

which is a predetermined pattern of bits. The receiver identifies the pattern and gets ready to receive the characters.

The transmitter sends the data character by character, bit by bit. After sending all the characters, the transmitter sends another pattern of bits to indicate the end of transmission.

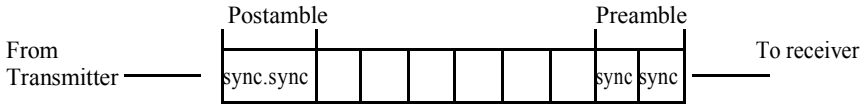


Fig. Transmission format for synchronous transfer

This format is generally used for high speed transmission (more than 20 kbps). Here the receiver and transmitter are synchronized by a master clock i.e. both function depending upon the same clock signal.

Baud Rate: The rate at which the bits are transmitted (bits per seconds) is called a baud; technically, however it is defined as the number of signal changes/ second. Each piece of equipment has its own band requirements.

Basic Concept of Synchronous and Asynchronous Modes

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]

METHODS OF PARALLEL DATA TRANSFER

1. SIMPLE I/O

- 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. 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".**

8255A And Its Working

INTRODUCTION

- The 8255A PPI provides three 8 bit input/output ports in one 40 pin package.
- The chip can be interfaced directly to the data bus of the processor allowing its function to be programmed.
- In one application, a port may appear as an output, but in another by reprogramming it an be as input.

DESCRIPTION

- The 8255A is a general purpose parallel I/O interfacing device.
- It provides 24 I/O lines organized as three 8 bit input/output ports labeled A,B and C.
- Each of the ports A or B can be programmed as an 8 bit input or output port.
- Port C can be divided in half, with the topmost or bottommost four bits programmed as inputs or outputs.
- Port C can be used as an 8 bit input or output port as two 4 bit ports or to produce handshake signals for Port A and B.
- The 8255A is a very versatile device.
- It can be programmed to look like
    - Three simple I/O ports (called MODE 0)
    - Two handshaking I/O ports (called MODE 1)
    - Port A as a bidirectional I/O port with 5 handshaking signals (called MODE 2) The modes can also be intermixed.
    - For example, Port A can be programmed to operate in MODE 2 while Port B in MODE 0.

PIN CONFIGURATION OF 8255 PPI

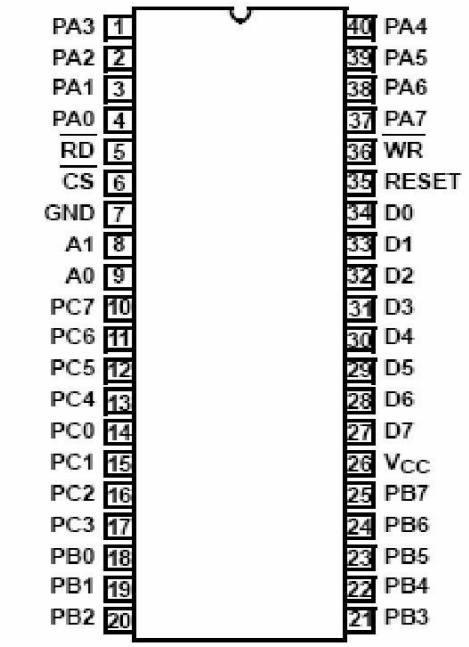


Figure: Pin Configuration of 82C55A PPI

PIN NAMES WITH DESCRIPTIONS

D7-D0	DATA BUS [BIDIRECTIONAL]
RESET	RESET INPUT
CS	CHIP SELECT
RD	READ INPUT
WR	WRITE INPUT
A0,A1	PORT ADDRESS
PA7-PA0	PORT A
PB7-PB0	PORT B
PC7-PC0	PORT C
VCC	+5V
GND	GROUND

INTERNAL BLOCK DIAGRAM OF 8255A PPI

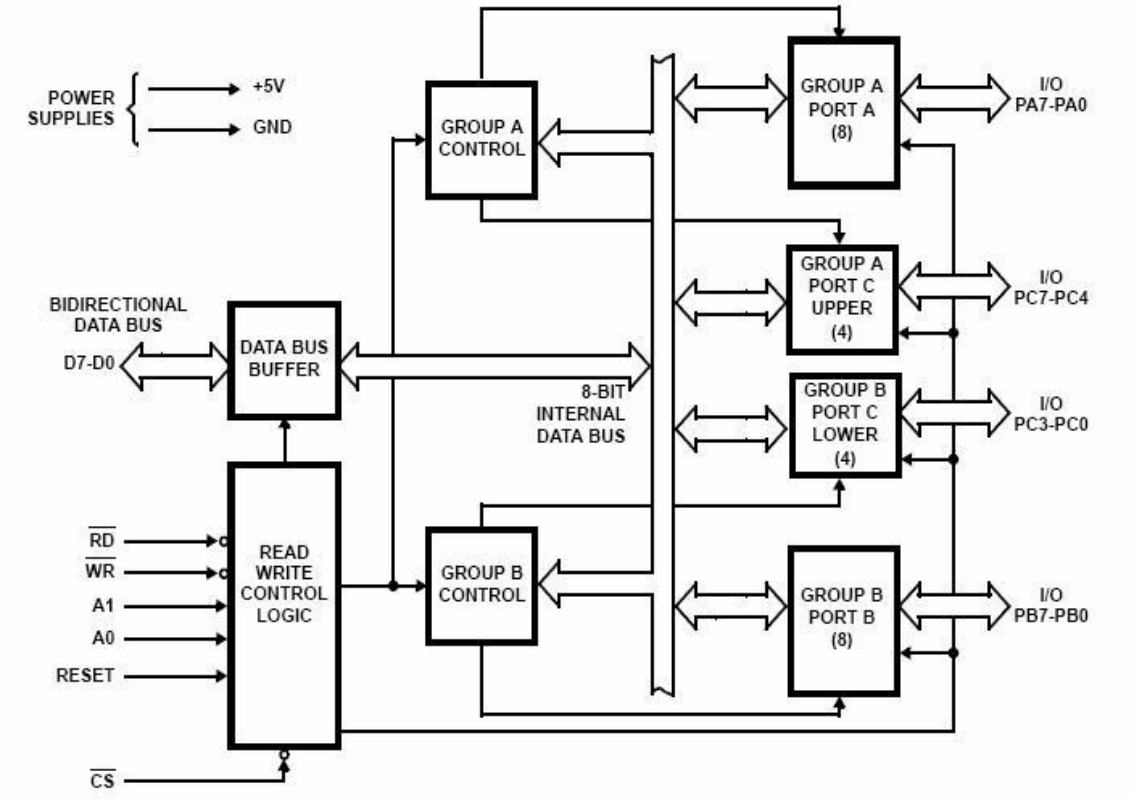


Figure: 8255A Internal Block Diagram

- The address inputs A0, A1 allow you to selectively access one of the three ports or the control register.
- The internal addresses for the devices are tabulated below.

DESCRIPTION	A1	A0
PORT A	0	0
PORT B	0	1
PORT C	1	0
CONTROL	1	1

- The CS input of 8255A enables it for reading or writing
- This input is driven by an address decoder.

- The RD and WR input pins determine the direction of data flow over the chips 8 bit bidirectional data bus.
- The RESET input of 8255A is connected to the system reset line so that, when the system is reset all the port lines are initialized as input lines. This is done to prevent destruction of circuitry connected to port lines.

TRUTH TABLE FOR THE 8255A PPI

1. INPUT OPERATION

A1	A0	RD	WR	CS	INPUT OPERATION
0	0	0	1	0	PORT A->DATA BUS
0	1	0	1	0	PORT B->DATA BUS
1	0	0	1	0	PORT C->DATA BUS

2. OUTPUT OPERATION

A1	A0	RD	WR	CS	INPUT OPERATION
0	0	1	0	0	DATA BUS->PORT A
0	1	1	0	0	DATA BUS->PORT B
1	0	1	0	0	DATA BUS->PORT C
1	1	1	0	0	DATA BUS ->CONTROL

3. DIS FUNCTION

A1	A0	RD	WR	CS	INPUT OPERATION
X	X	X	X	1	DATA BUS TRISTATE
1	1	0	1	0	ILLEGAL CONDITION
X	X	1	1	0	DATA BUS TRISTATE

8255A OPERATIONAL MODES AND INITIALIZATION

1. MODE 0

- When programmed for MODE 0, the PPI offers three simple I/O ports with no handshaking signals.
- This mode is appropriate for I/O devices that do not need special synchronizing signals to exchange data with the processor.
- A common example is a keyboard used for data entry.
- When used as O/Ps, the PORT c lines can be individually set or reset by sending a special control word to control register address.
- Two halves of PORT C are independent so one half can be initialized as input and the other half as output.

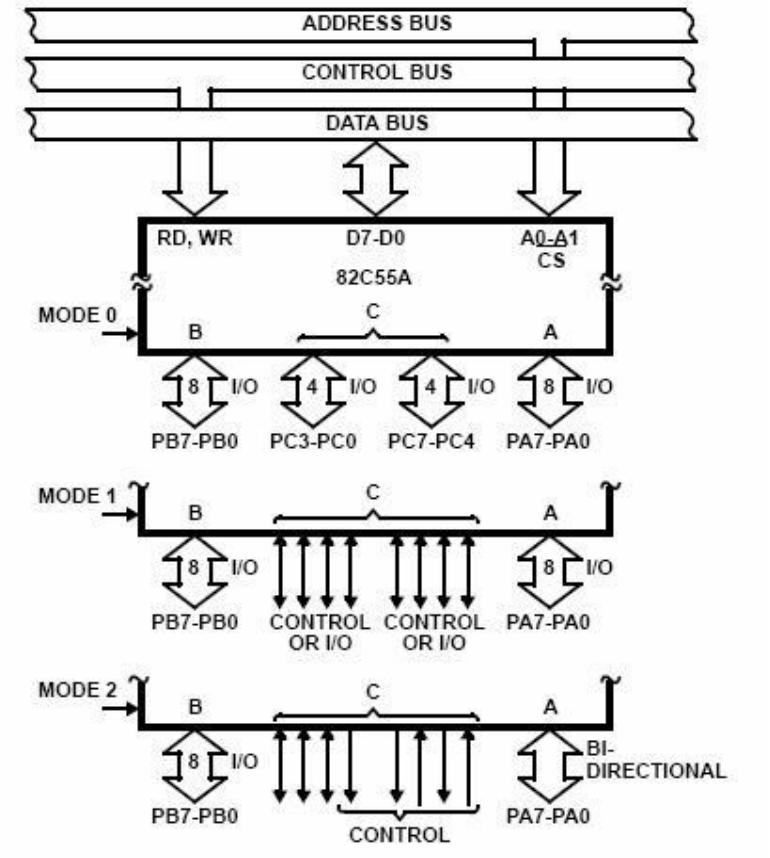
2. MODE 1

- When programmed for MODE 1, the PPI offers PORT A or PORT B for a handshake input/output operation.
- Pins PC0,PC1 and PC2 function as handshake lines for PORT B
- Pins PC3,PC4 and PC5 function as handshake signal for PORT A (input).
- Pins PC6 and PC7 are available for use as input/output lines for PORT A
- If PORT A is initialized as handshake O/P port, then PORT C pins.

- PC3, PC6 and PC7 function as handshake signals.
- PORT C pins PC4 and PC5 are used as input/output lines for PORT A

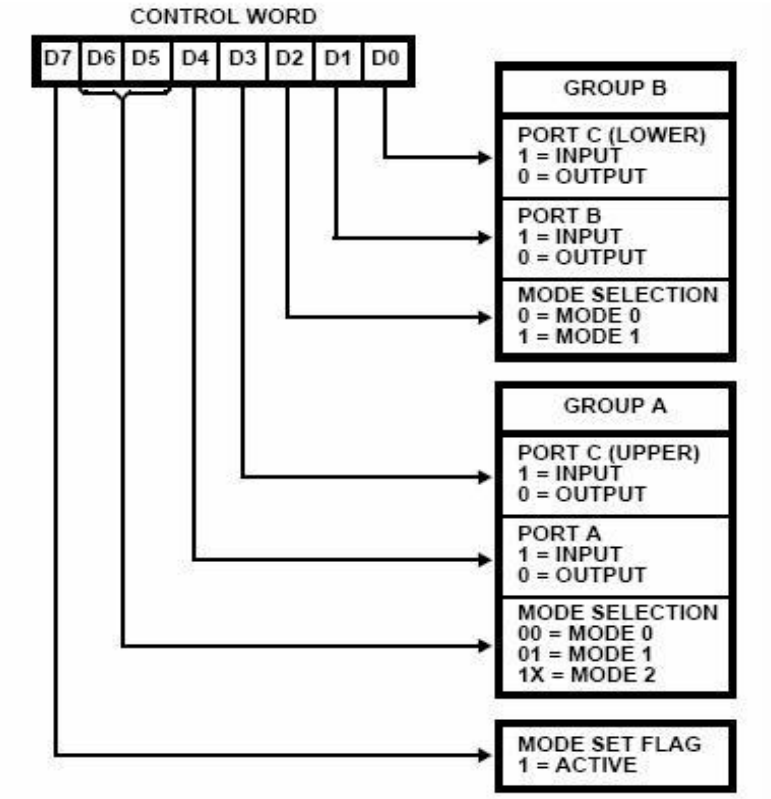
3. MODE 2

- Only PORT A can be initialized in MODE 2.
- In MODE 2, PORT A can be used as bidirectional handshake data transfer.
- This means that data can be outputted/inputted from same eight lines.
- If PORT A is initialized in MODE 2, then pins PC3 through PC7 are used as handshake lines for PORT A.
- The other three pins PC0 through PC2 can be used for I/O port if PORT B is in MODE 0.
- The same 3 pins will be used for PORT B handshake signals if PORT B is initialized in MODE 1.

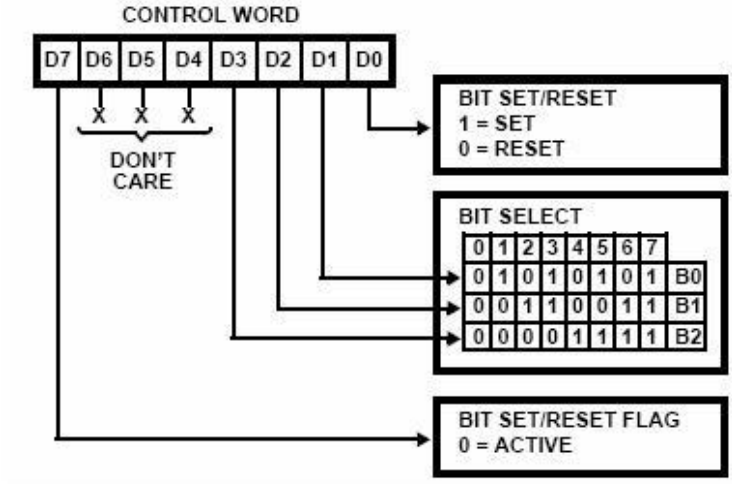


CONTROL WORDS

MODE SET CONTROL WORD



PORT C BIT SET/RESET CONTROL WORD



PROBLEM 1

Write the 80x86 initialization routine required to program the 8255A for mode 0, with PORT A as an output port and PORT B and C as an input ports.

SOLUTION

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	1	0	1	1

D7 : 1 -> Mode set
D6 D5 : 00 -> Mode 0
-> Port A
D4 : 0 Output
D3 : 1 -> Port C Upper Input
-> Mode
D2 : 0 0
D1 : 1 -> Port B Input
D0 : 0 -> Port C Lower Input

Assume the port address of control port is 0FFH

```
MOV AL,8BH
OUT 0FFH,AL
```

PROBLEM 2

Write an 80x86 program to input a byte from PORT B of PPI chip and output this byte to PORT A of the same chip. Assume it is already initialized.

SOLUTION

Assume the port address of control port is 0FDH
Assume the port address of PORT B as 0FDHH

```
IN AL,0FDH   ;Read from PORT B
OUT 0FCH,AL ; Write to PORT B
```

PROBLEM 3

Write instruction to initialize 8255 to configure PORT A as simple

output port, PORT B as simple input port, PORT C upper as output and port c lower as an input port.

PROBLEM 4

Write a program to take input from the 8 switches connected to PORT B and display the status of the switches thus red in the 8 LEDS connected to PORT A. Show how you derive the control word.

PROBLEM 5

Write instruction to initialize 8255 to configure PORT A and display the status of the switches thus read in the 8 LEDS connected to PORT B. show how do you derive the control word.

RS 232C

This interface standard is most widely used standard for serial communication between microcomputers and peripheral devices. The interface, defined by EIA, relates essentially to two types of equipment. The first is known as data terminal equipment. While the second is referred as data communication equipment (DTE). The data terminal equipment (e.g a microcomputer) is capable of sending and/or receiving data via the serial interface. The data communication equipment on the other hand is generally through of as a device which can facilitate serial data communications (e.g modems).

- RS 232C works in a negative logic. The standard specifies that 'the logic one' level is a voltage between -3 and -15 v and 'the logic zero' is a voltage between +3 and +15 V. The commonly used voltages are +12v and -12V.

- The transmission line normally used a twisted pair of shielded wire with a line capacitance of more than 1200PF and no less than 300Pf. The standard specifies the line length to 50 feets only.
- The standard describes the function of 25 signal and hansake pins for serial data transfer. It also specifies that the DTE connector should be male and the DCE connector should be female. Usually 9 pin and 25 pin connectors are available.

Among the 25 pins of RS232C the independent pin numbers are:

Pin no. **Signals Functions**

2 Transmit data, TXD Output , transmit data from DTE to DCE

3 Receive data RXD Input, DTE receive data from DCE

4 Request to send, RTS General Purpose output from DTE

5 Clear to send, CTS Input to DTE, used as handshake

6 Data set ready, DSR Input to DTE, indicate that DCE is ready

7 Signal ground, GND Common reference bet. DTE and DCE

8 Data carrier detect, DCD Used by DTE to disable data reception

20 Data terminal ready, DTR Output means DTE is ready.

The main problem with –RS-232C is that it can only transfer data reliably about 50ft (16.4m) at its maximum rate of 20k band. If longer lines are used the transmission rate has to be drastically reduced. For higher rate of transfers and for longer distance we have another standard defined.

Standards Speed Distances Voltage Range

RS 232C 20k band 50ft ±15V

RS-422A 10Mbaud at 40ft 4000ft ±7

100 Kbaud at 4000ft

RS-485A 100K baud at 30ft

1 kbaud at 4000ft 4000ft ±12V

RS 232C interface with DTE and DCE: The figure below shows a interfacing with minimum lines.

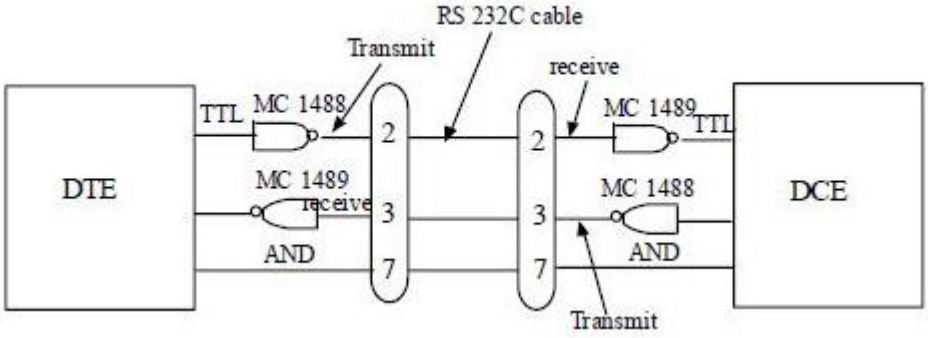


Fig. RS 232C interface

The signaling in RS-232C is not compatible with the TTL logic level. For TTL 0 v to 0.2V is considered a logic 0 and 3.4 v to 5v as logic 1. But RS-232C works in a negative logic -3 to -15v considered as logic 1 and +3 to +15v as logic 0. Because of this incompatibility of the data lines with the TTL logic, voltage translators called line drivers and line receivers are required to interface TTL logic with RS-232C signals. The line driver MC 1488 converts logic 1 into approx -9V and logic 0 into +9v. Before it is

received by the DCE it is again converted by the line receiver MC 1489 into TTL-Compatible logic.

The minimum interface required both a computer and a peripheral device requires three lines; pin 2,3 and 7. These lines are defined in relation to the DTE; the terminal transmits on pin 2 and receives as pin 3. On the other hand the DCE transmits on pin 3 and receives on pin 1. Pin 7 is ground pin.

Keyboard and Display Controller: Introduction to 8279

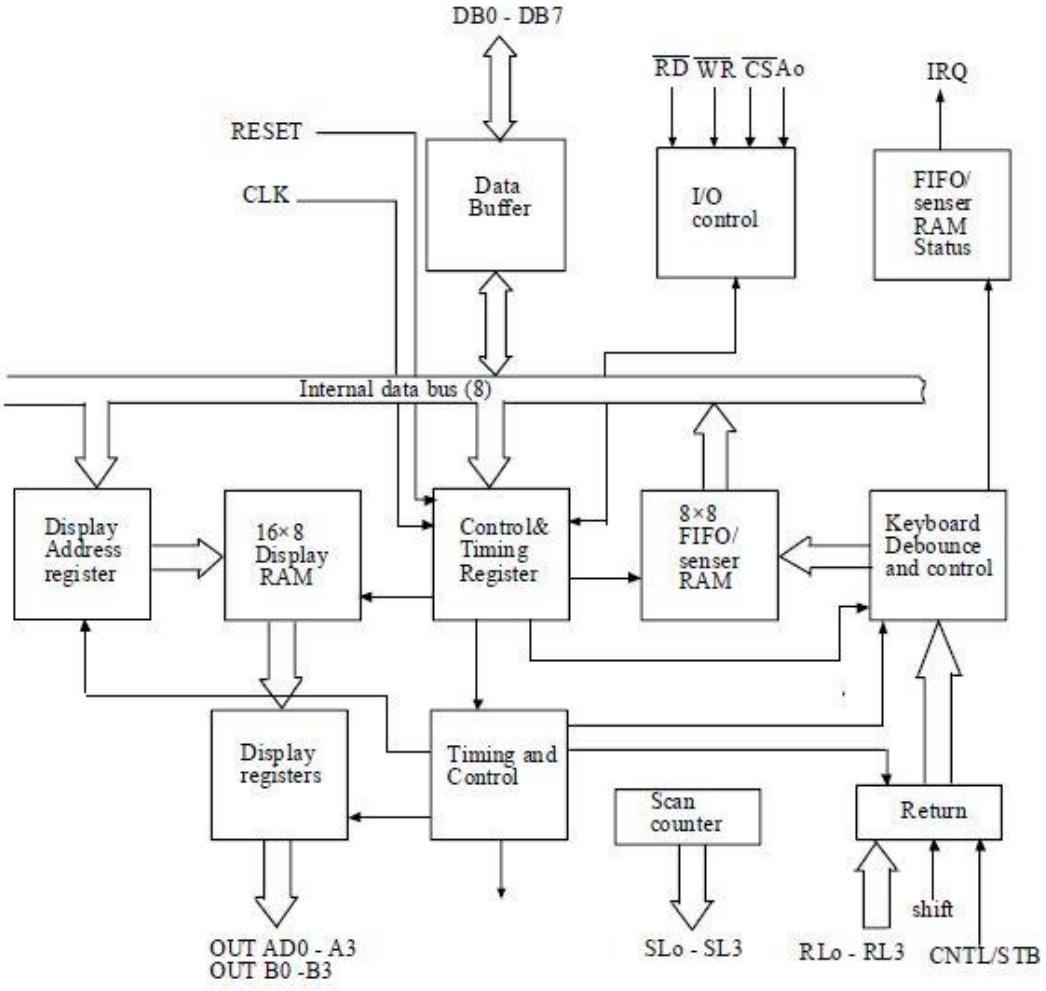


Fig. 8279 block diagram

Keyboard Section: This section has eight lines (RL0 – RL7)that can be connected to 8 columns of a keyboard, plus two additional lines shift and

CNTL/STB (control/strobe). The keys are automatically debouched and the keyboard can operate in two modes; two-key lockout or N-key rollover. In two-key lockout mode, if two keys are pressed almost simultaneously only the first key is recognized. In the N-key rollover mode, simultaneous keys are recognized and their codes are stored in the internal buffer. It can also be set up so that no key is recognized until only one key remains pressed.

This section also includes 8 × 8 FIFO RAM, that store keyboard entries and provides IRQ (interrupt request). Signal when FIFO is not empty.

Scan Section: The scan section has scan counter and 4 scan lines (SL0 – SL3). These 4 scan lines can be decoded using a 4 to 16 decoder to generate 16 lines for scanning.

Display Section: The display section has eight output lines divided into two groups A0 – A3 and B0 –B3. These lines can be used, either as a group of eight lines or as two groups of four, in conjunction with the scan line, for a multiplexed display. The display can be blanked by using the BD line. This section includes 16 × 8 display RAM.

MPU Interface Section: This section includes eight bidirectional data lines (DB0 –DB7), one interrupt request (IRQ) line, and 6 lines for interfacing , including the buffer address line (A0) When A0 is high, signals are interpreted as control words or status; when A0 is low, signals are

interpreted as data. The IRQ line goes high whenever data entries are stored in the FIFO indicating the availability of data.