BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 4

DCA2203

SYSTEM SOFTWARE

Unit 1

Introduction to Software Processor

Table of Contents

1. INTRODUCTION

The Unit introduces the concepts of design and implementation of system software. Software is a set of instructions or programs written to carry out the certain tasks on digital computers. It is classified into *system software* and *application software.*

In this unit, we are going to discuss system software and machine architecture. In the next session, we will discuss the Architecture of Intel 8086. Then we will discuss data and instruction format and addressing modes. In the last sessions, we will discuss Instruction sets and I/O, and programming.

1.1 Learning Objectives

After studying this unit, learners should be able to:

- ❖ *Describe system software and machine architecture*
- ❖ *Explain the Architecture of Intel 8086.*
- ❖ *Define data and instruction formats*
- ❖ *Describe various addressing modes.*
- ❖ *Explain instruction sets and I/O Programming.*

2. SYSTEM SOFTWARE AND MACHINE ARCHITECTURE

One characteristic in which most system software differs from application software is machine dependency.

System software – support operation and use of the computer.

Application software – solution to a problem.

System software

System software consists of a variety of programs that support the operation of a computer. This software makes it possible for the user to focus on an application or other problem to be solved, without needing to know the details of how the machine works internally. Examples are operating systems, compilers, assemblers, macro processors, loader or linker, debugger, text editors, database management systems, and, software engineering tools.

Application software

Application software focuses on an application or problem to be solved. System software consists of a variety of programs that support the operation of a computer.

Assembler

An assembler translates mnemonic instructions into machine code. The design of an assembler is directly affected by the instruction formats, addressing modes, etc. Similarly, *Compilers* must generate machine language code, considering hardware features like the quantity and type of registers and the available machine instructions . *Operating systems* are directly concerned with the management of all of the resources of a computing system.

There are aspects of system software that do not directly depend upon the type of computing system, general design and logic of an assembler, compiler and code optimization techniques which are independent of target machines.

Likewise, the process of linking together independently assembled subprograms does not usually depend on the computer being used.

The Simplified Instructional Computer (SIC)

A simplified Instructional Computer (SIC) is a hypothetical computer that includes the hardware features most often found on real machines. There are two versions of SIC, they are, standard model (SIC) and the extension version (SIC/XE) (extra equipment or extra expensive).

SIC Machine Architecture

The SIC Machine Architecture contains theMemory and Registers, Data Formats, Instruction Formats, Addressing Modes, Instruction Set, Input, and Output

Memory

The computer's RAM has 215 bytes, that is 32,768 bytes, It uses Little Endian format to store the numbers, 3 consecutive bytes form a word, and each location in memory contains 8-bit bytes.

Registers

There are five registers, every 24 bits in length. Their mnemonic, number, and use are given below.

Mnemonic Number Use:

A	0	Accumulator: used for arithmetic operations
X	1	Index register: used for addressing
L	2	Linkage register: JSUB
PC	8	Program counter
SW	9	Status word, including CC

Data Formats

Integers are stored as 24-bit binary number format, 2's complement representation is used for negative values, and characters are stored using their 8-bit ASCII codes, No floating-point hardware on the standard version of SIC.

Instruction Formats

Opcode(8) x Address (15)

All machine instructions on the standard version of SIC have the 24-bit format as shown above.

Addressing Modes

Addressing modes are the ways in which architectures specify the address of an object they want to access.

There are two addressing modes available, which are as shown below.

Mode	Indication	Target address calculation
1)	Direct x= 0	TA = address
2)	Indexed x = 1	TA = address + (x)

Parentheses are used to indicate the contents of a register or a memory location.

Instruction Set

SIC is used to load and store instructions (LDA, LDX, STA, STX, etc.). Integer arithmetic operations: (ADD, SUB, MUL, DIV, etc.). All arithmetic operation involves register A and a word in memory, with the result being left in the register. Two instructions are provided for subroutine linkage. COMP Compares the value in register A with a word in memory; this instruction sets a condition code CC to indicate the result. There are conditional jump instructions: (JLT, JEQ, JGT), these instructions test the setting of CC and jump accordingly. JSUB jumps to the subroutine by placing the return address in register L, and RSUB returns by jumping to the address contained in register L.

Input and Output

Input and Output are performed by transferring 1 byte at a time to or from the rightmost 8bits of register 'A' (accumulator). The Test Device (TD) instruction tests whether the addressed device is ready to send or receive a byte of data. Read Data (RD), and Write Data (WD) are used for reading or writing the data.

Data movement and Storage Definition

LDA, STA, LDL, STL, LDX, STX (A - Accumulator, L – Linkage Register, X – Index Register), all uses the 3-byte word. LDCH, STCH associated with characters uses 1-byte. There are no memory-memory move instructions Storage definitions are

WORD	–	ONE-WORD CONSTANT
RESW	–	ONE-WORD VARIABLE
BYTE	–	ONE-BYTE CONSTANT

RESB – ONE-BYTE VARIABLE

SIC/XE Machine Architecture

Memory

The maximum memory available on a SIC/XE system is 1 Megabyte (220 bytes)

Registers

Additional B, S, T, and F registers are provided by SIC/XE, in addition to the registers of SIC

Mnemonic	**Number**	**Special use**
B	3	Base register
S	4	General working register
T	5	General working register
F	6	Floating-point accumulator (48 bits)

Instruction Formats

The new set of instruction formats for SIC/XE machine architecture is as follows.

Format 1 (1 byte):

op {8}

Format 2 (2 bytes):

op {8} r1 {4} r2 {4}

Format 3 (3 bytes):

op {8} n i x b p e displacement {12}

Format 4 (4 bytes):

op {8} n i x b p e address {20}

Formats 3 & 4 introduce addressing mode flag bits:

(Note I will use TA for "target address" & disp for "displacement")

flags n & i:

- n=0 & i=1 *immediate addressing* - TA is used as an operand value (no memory reference)

- n=1 & i=0 *indirect addressing* - word at TA (in memory) is fetched & used as an address to fetch the operand from
- n=0 & i=0 *simple addressing* TA is the location of the operand
- n=1 & i=1 *simple addressing* same as n=0 & i=0

flag x:

x=1 *indexed addressing* add contents of X register to TA calculation

flag b & p (Format 3 only):

- b=0 & p=0 *direct addressing* displacement/address field contains TA (note Format 4 always uses direct addressing)
- b=0 & p=1 PC *relative addressing* - TA=(PC)+disp -2048<=disp<=2047)*
- b=1 & p=0 Base *relative addressing* - TA=(B)+disp (0<=disp<=4095)**

* note - in PC relative, dispis interpreted as a 12-bit signed integer in 2's complement

** note - in Base relative, dispis interpreted as a 12-bit unsigned integer

flag e:

e=0 use Format

3 e=1 use Format 4

Addressing modes & Flag Bits

Five possible addressing modes with the combinations are as follows.

Direct (x, b, and p all set to 0): operand address goes as it is. n and i are both set to the same value, either 0 or 1. Although typically that value is 1,, if set to 0 for format 3 can assume that the rest of the flags (x, b, p, and e) are used as a part of the address of the operand, to make the format compatible to the SIC format

Relative (either b or p equal to 1 and the other one to 0): the address of the operand should be added to the current value stored at the B register (if b = 1) or to the value stored at the PC register (if p = 1)

***Immediate* (i = 1, n = 0):** The operand value is already enclosed on the instruction (i.e. lies on the last 12/20 bits of the instruction)

***Indirect* (i = 0, n = 1):** The operand value points to an address that holds the address for the operand value.

***Indexed* (x = 1):** value to be added to the value stored at the register x to obtain the real address of the operand. This can be combined with any of the previous modes except immediate.

Instruction Set

SIC/XE provides the instructions that are available in the standard version. In addition to that the Instructions are used to load and store the new registers such as LDB, STB and so on. Floating-point arithmetic operations such asADDF, SUBF, MULF, DIVFand use Register move instruction such as RMO, Register-to-register arithmetic operations, ADDR, SUBR, MULR, DIVR and, Supervisor call instruction: SVC.

Input and Output

The I/O Channels are used to perform input and output while the CPU is executing other instructions. Allows overlap of computing and I/O, resulting in more efficient system operation. The instructions SIO, TIO, and HIO are used to start the test and halt the operation of I/O channels.

Comparison of CISC and RISC Architectures:

The architectures of CISC and RISC machines are introduced in the section that follows. Traditional machines are CISC machines. We also have more current RISC machines in addition to these.

CISC MACHINES:

Traditional (CISC) machines are just computers with a complex instruction set have a relatively large and complex instruction set, different instruction formats, different lengths, different addressing modes, and the implementation of hardware for these computers is complex.

Example: VAX and Intel x86 processors

RISC Machines

RISC means Reduced Instruction Set Computers. These machines are intended to simplify the design of processors. They have greater reliability, faster execution, and less expensive processors and also they have standard and fixed instruction lengths. The number of machine instructions, instruction formats, and addressing modes is relatively small. Example: Ultra SPARC Architecture and Cray T3E Architecture

SELF ASSESSMENT QUESTIONS – 1

1. _________translates mnemonic instructions into machine code.
2. Maximum memory available on a SIC/XE system is___________________.
3. In__________________addressing Mode, operand address goes as it is.

3. ARCHITECTURE OF INTEL 8086

The **8086** is a 16-bit microprocessor chip designed by Intel between early 1976 and mid-1978. 8086 has 16-bit ALU; this means 16-bit numbers are directly processed by 8086. It contains a 16-bit data bus, so it can read data or write data to memory or I/O ports either 16 bits or 8 bits at a time. It has 20 address lines, so it can address up to 220 i.e. 1048576 = 1M bytes of memory (words i.e. 16-bit numbers are stored in consecutive memory locations). Due to the 1M bytes memory size multiprogramming is made feasible as well as several multiprogramming features have been incorporated into the 8086 design.

Features of 8086

- 8086 is a 16bit processor. It's ALU, internal registers work with 16bit binary words.
- 8086 has a 16bit data bus. It can read or write data to a memory/port either 16bits or 8bits at a time.
- 8086 has a 20bit address bus which means, it can address up to 220 = 1MB memory location.
- The frequency range of 8086 is 6-10 MHz.
- 8086 processors employ parallel processing.
- It can support up to 64K I/O ports.
- It requires up to +5Volts power supply.
- The permissible range of the memory address is from 00000H to FFFFFH.
- 8086 operates in two modes, that are minimum mode and maximum mode.
- Memory is byte-addressable, a separate address is there for every byte.
- It can prefetch up to 6 bytes of instructions and queue them so that execution speed is increased.

Architecture of Microprocessor 8086

Microprocessor 8086 is a 40-pin 16-bit Intel IC made by using HMOS technology. It consists of 2900 transistors. The term 16-bit means that the arithmetic logic unit, its internal registers, and instructions are designed to work with 16-bit binary words. As stated earlier it has a 16-bit data bus and 20-bit address bus. The pipelined architecture of microprocessor 8086 is shown in figure 1.1.

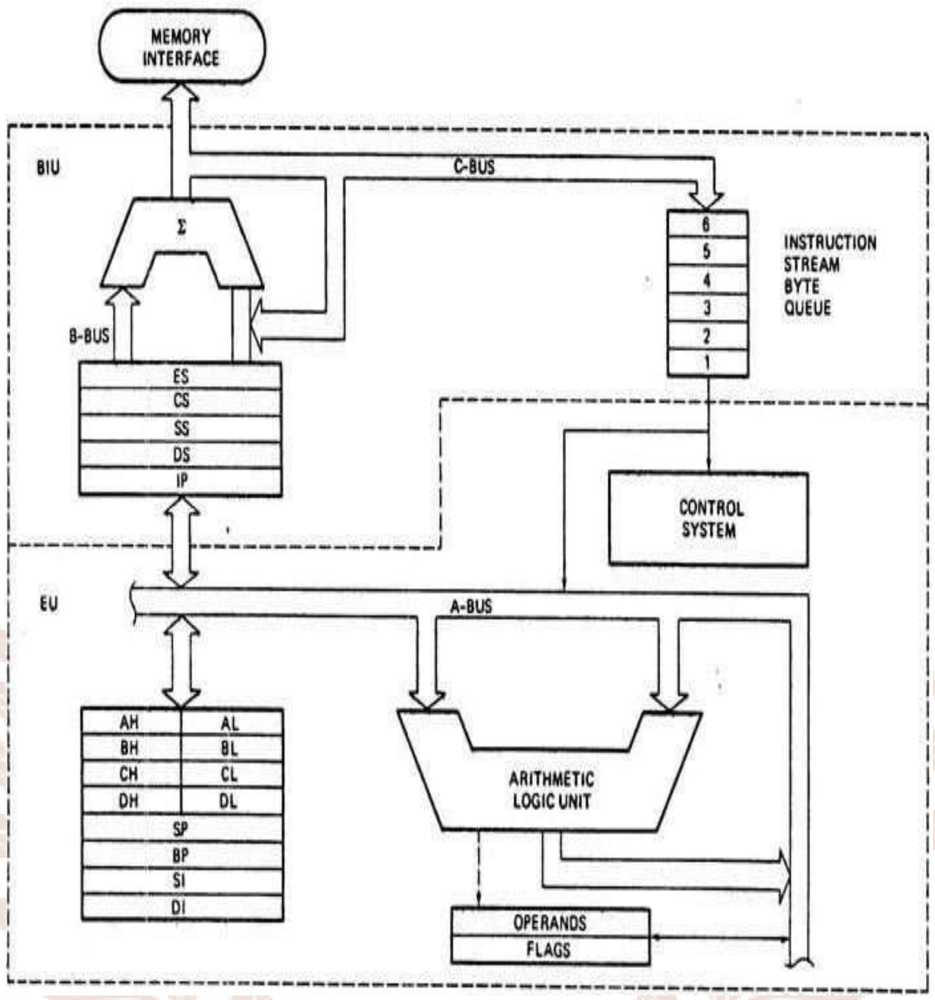

Fig 1.1: Block Diagram of Microprocessor 8086

The architecture is divided into two functional units as you can see in the above figure:

1. Bus Interface Unit(BIU)
2. Execution Unit(EU)

Bus Interface Unit (BIU): The BIU handles all the data and addresses on the buses for the execution unit (EU). It performs all bus operations such as instruction fetching, reading, and writing operands for memory and calculating the addresses of the memory operands. The instruction bytes are transferred to the instruction queue. The execution unit uses the instruction queue to execute the instructions. Both BIU and EU work asynchronously to execute instructions by using Pipelining mechanism which means overlapping of instruction fetch and execute mechanism. The pipelining results in efficient use of the system bus and increases system performance. Here will study pipelining mechanisms in detail in the coming section.

The various parts of BIU are as follows:

1. **Instruction Queue**

The prefetched instruction bytes are stored in a first in first out group of registers called an **instruction queue** for the Execution Unit (EU). When the EU is ready for its next instruction, it simply reads the instruction from this instruction queue. This is much faster than sending out an address to the system memory and sending back the next instruction byte. Fetching the next instruction while the current instruction executes is called **pipelining.** In 8086 pipelining is used to speed up the execution of the program as the instruction fetch and execution steps are overlapped. The 8086, a 6-byte instruction queue is presented at the Bus Interface Unit (BIU). It is used to prefetch and store a maximum of 6 bytes of instruction code from the memory.

2. **Segment Registers**

The BIU contains four 16-bit segment registers. These are:

- Extra segment (ES) register
- Code segment (CS) registers
- Data segment (DS) registers
- Stack segment (SS) registers

The upper bits of the starting address of each of the segments are held by these segment registers. The part of the segment starting address stored in the segment register is known as the segment base. The description of segment registers is as follows:-

i) *Code Segment Register:* It is a 16-bit register used for accessing instructions referenced by instruction pointer or we can say that it is used to address the memory location in the code segment, where the executable program is stored.

ii) *Stack Segment Register:* It is also a 16-bit register and contains the data used when the subprogram is executing. It contains the data pointed by Stack Pointer (SP) and base pointer(BP).

iii) *Data Segment:* It is a 16-bit register containing the data used by the program. The data used by pointed by general-purpose registers (AX, BX, CX, DX) and Index registers (SI, DI) is located in the data segment.

iv) *Extra Segment:* It is a 16-bit register used to hold extra destination data during some string manipulation operations. It contains the data pointed by the DI register.

v) *Instruction Pointer:* It holds the 16-bit address and holds the address of the next instruction to be executed.

Execution Unit (EU)

It decodes instructions fetched by BIU, generates control signals, and executes the instruction. The main parts of the control unit are the control system, arithmetic, and logic unit instruction decoder unit. The Control systems perform various internal operations. Arithmetic and the logical unit perform different arithmetic operations like increment, decrement operations, etc., and logical operations like AND, OR, NOT, etc. Decoder translates instruction fetched from memory to perform the necessary operation.

Flag Register

It shows the condition or changes produced by the execution of an instruction and these are modified by the CPU automatically after some mathematical or logical operation is performed. Nine flags are in microprocessor 8086 and are shown in figure 1.2.

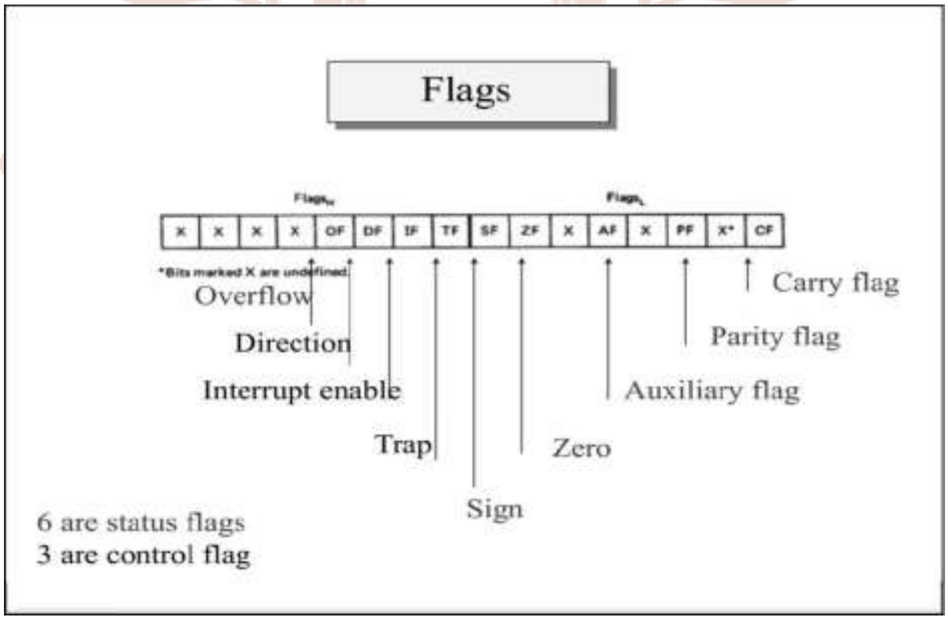

Fig 1.2: Flag register

The flags are categorized into two categories:

1. Conditional Flags
2. Control Flags

Conditional Flags: The result of the last arithmetic and logical instruction is represented by these flags.

The description of all conditional flags is as follows:

- Carry flag: If by any arithmetic operation a carry is produced at MSB (most significant bit position) then it is set to 1.
- Auxiliary flag: If an operation performed in ALU generates a carry/borrow from lower nibble (i.e. D0 – D3) to upper nibble (i.e. D4 – D7), the AF flag is set i.e. carry given by D3 bit to D4 is AF flag. This is not a general-purpose flag; it is used internally by the processor to perform Binary to BCD conversion.
- Parity flag: This flag is used to show the parity bits. If an even number of 1 bit are there then it is set to 1 else set to zero.
- Zero flag: If the result of the arithmetic or logical operation is 0 then it is set to 1 else set to 0.
- Sign flag: If the result of the operation is negative then this flag is set to one and when the result is positive it is set to zero. These flags take the value of MSB as in sign-magnitude representation MSB indicates a sign of the number.
- Overflow flag: When the signed numbers are added or subtracted and there is a signed overflow then this flag is set else reset. If the overflow flag is set then this indicates that the result has exceeded the capacity of the machine.

Control Flags: These flags are not automatically set or rest these flags are needed to be set or reset by the user to control certain operations.

Description of these flags is as follows:

- Trap Flag: When this flag is set then the user can execute one instruction at a time and able to debug.
- Interrupt flag: If this flag is set then interrupt (interruption of the program) is enabled if reset then the interrupt is dabbled.

- Direction Flag (DF): Accessing of string bytes is from higher to lower memory address if this flag is set and if reset then accessing of string bytes is from lower to higher memory address.

General Purpose Registers

In microprocessor 8086, there are eight general purpose registers, which are AL, BL, CL, DL, AH, BH, CH and DH.. All these registers can store up to 8 bits (I byte of data. Some pairs of registers are used to store 16 bits of data and these pairs are AL-AH, BL-BH, CL-CH, and DL-DH these pairs are called AX, BX, CX, and DX respectively.

AX (AL-AH) register: This register pair is used for 16-bit operation also known as Accumulator and is used to store operands for arithmetic operations.

BX (BL-BH) register: This register pair is used as a base register that is it is used to store the starting address of a data segment.

CX (CL-CH): It is known as counter register and is used in loop instructions. DX (DL-DH): It is used to store the I/O port addresses for I/O instructions.

Stack Pointer Register

It contains the offset of 16 bits from the start of the segment to the top of the stack. The top of the stack indicates the location where the word is most recently stored.

Other Pointer and Index Registers

Some other registers are also there in the execution unit. These are SI (Source Index), DI (Destination Index), and BP (Base Pointer) and are used to store the 16-bit data.

Pin diagram and description of 8086 Microprocessor

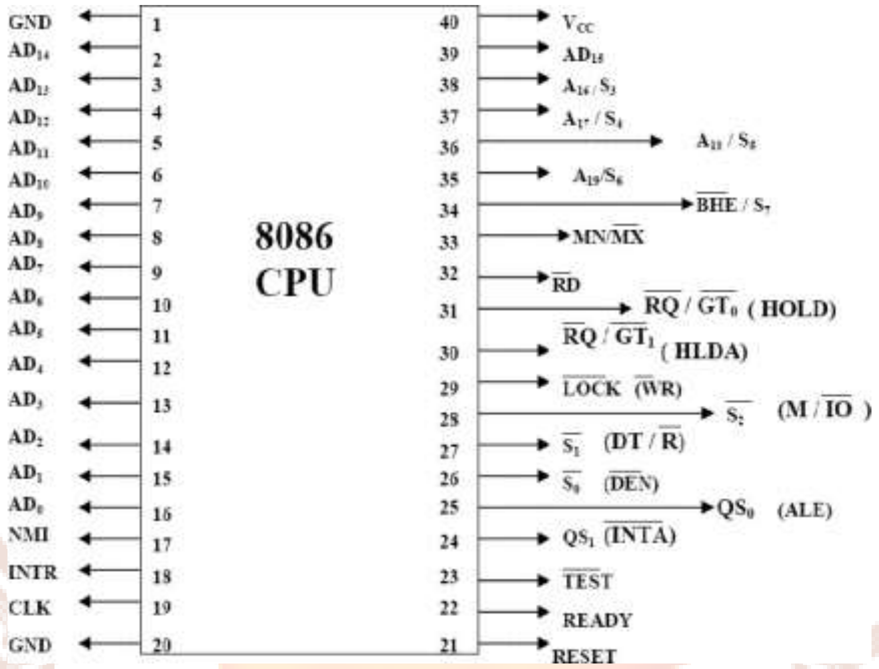

Fig 1.3: Pin Diagram of 8086

The Microprocessor 8086 is a 16-bit CPU available at different clock rates and packaged in a 40-pin CERDIP (CERamic Dual In-line Package) or plastic package. It operates in either single processor mode or multiprocessor mode so that it can achieve high performance.8086signal can be divided into three categories and these are as follows:-

- The signals have common functions in both modes.
- The signals that have a specific function are in minimum mode.
- And the signals that have specific functions in maximum mode.

Pin description of 8086 common to both the modes:

1. **AD_0-AD_{15}:** These are the time-multiplexed addresses and data lines. During T1 state address is there on the line and during T2, T3, TW, AND T4 state address data are there.
2. **A19/S6, A18/S4, A17/S4, A16/S3:** These lines are upper address lines and are multiplexed with status signals (S3-S6). These lines act as address lines during T states and show status for the remaining T states. These lines remain low during I/O operations.S4 and S3 indicate which segment is being used by the processor. and S6 is always has a low.

Table 1.1: Encoding of S4 and S3

S4	S3	Indications
0	0	alternate data
0	1	Stack
1	0	code or none
1	1	Data

3. **BHE/S7:** Bus High Enable/Status

BHE stands for bus high enable signal shows the data transfer over the higher-order (D15-D8) data bus. When BHE is low then data is transferred over D15-D18.

Table1.2: Encoding of BHE and A0

BHE	A0	Indication
0	0	Complete word (16 bit is accessed)
0	1	Upper byte is accessed
1	0	Lower byte is accessed
1	0	None

4. **RD:** Read

It is an active low output signal. When this signal is low, it indicates the processor is performing a read (memory or I/O read) operation. It is activated during T2, T3, and TW of any read machine cycle and in other T states, it is high.

5. **READY:** This is an active-high input signal and is used for slower peripherals. When a peripheral device is ready to receive or transmit the data then it sends the READY signal to the processor.
6. **INTR:** Interrupt Request: This is an active high-level triggered input signal. This signal is sampled during the last clock cycle of every instruction it is sampled to determine the availability of the request and if there is any request then the processor enters the interrupt acknowledgment cycle.
7. **TEST:** It is an active low input signal. Execution will continue if TEST=0 else the processor will be in an idle state.

8. **NMI:** Non-Maskable Interrupt It is an edge-triggered input and cannot be masked. And if there is a transition from low to high it indicates an interrupt response at the end of the current instruction.
9. **RESET:** It is an input signal. This signal results in the termination of current activities of the system and results in restarting of the system.
10. **CLK:** Clock Input

    The clock input provides the basic timing for processor operation and bus control activity.
11. **Vcc:** +5 volts power supply is connected to this pin.
12. **GND:** It is the GROUND pin.
13. **MN/MX:** It indicates in which mode the processor will operate that is minimum mode or maximum mode.

Pin Description of 8086 in minimum mode:

1. **M/IO (Status line):** This is an output signal which shows whether the processor is accessing memory or I/O.
2. **WR (Write):** It is an active low output signal, if high indicates that the write operation is being performed either from I/O or from memory.
3. **INTA:** It is an active low output signal. If this signal is low then this indicates that it means the acceptance of the interrupt signal by the processor.
4. **ALE:** It is an active high output signal and shows whether the address present on data or address bus is valid or not.
5. **DT/R:** it is the data transmit and receive signal and shows the direction of the data flow through transceivers. If it is high then data will be transmitted else not.
6. **DEN:** It is data enable signal and indicates the presence of valid data over address/data lines.
7. **HOLD and HLDA:** It means to hold and hold acknowledge. When the HOLD is high then it means that the bus is requested by some other external device.

When the HOLD signal is received by the processor it releases the bus and issues an HLDA signal as an acknowledgment.

Pin description of 8086 in maximum mode:

1. **S2, S2, S0:**To the bus controller 8288 these signals are applied to generate different control signals and in the maximum mode of the INTA. In the T1, T2, and T3 states these signals are active and passive during T3 and T4 states.

Table 1.3: S0,S1,S0 encoding

S2	S1	S0	Operation
0	0	0	Interrupt acknowledges
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1	0	0	Code Access
1	0	1	Read Memory
1	1	0	Write Memory
1	1	1	Passive

2. **RQ/GT$_0$, RQ/GT$_1$:** It is an I/O request grant. Other local buses use these signals to force the processor to release the local bus at the end of the current bus cycle of the processor. With RQ/GT0 every pin is bidirectional and has higher priority over EQ/GT1.
3. **LOCK:** It is an active low output signal. This signal does not allow other devices to take control over the system bus when LOCK=0. The LOCK instruction activates this signal and it is active till the completion of the next instruction.
4. **Qs1, Qs0:** It means queue status and gives the status of the code-prefetch queue. This signals give the status of 8086 queues to the maths coprocessor as these signals are interfaced with the signals of the math coprocessor.

Table1.4: Encoding of Qs1 and Qs2

QS1	QS0	Encoding
0	0	No operation
0	1	First byte of opcode from queue
1	0	Empty the queue
1	1	Subsequent byte from the queue

SELF ASSESSMENT QUESTIONS – 2

4. 8086 CPU has______________________bit data bus and___________bit address bus.
5. 8086 microprocessor consists of________________transistors.
6. The prefetched instruction bytes are stored in a first in first out group of registers called an _________________________
7. _________ register shows changes produced by execution of an instruction.
8. _________are time multiplexed address and data lines.

4. DATA AND INSTRUCTION FORMATS

The complete collections of instructions that are understood by a CPU are called instruction set.8086 is two addresses, registered to memory architecture. Operation and operands are the elements of instruction.

- **Operation:** It represents what is being done.
- **Operands:** It represents how data to use in the operation.

For example,

C = A + B.

- Operations: addition (+) and assignment (=).
- Operands: A, B & C.

Source operands provide values to use (i.e., Inputs, here A & B) and Destination operands receive results (i.e., output, here C). Consider the Symbolic representation,

E.g.: ADD A, B

Here, the Addition operation is performing, and the operands are A, B, and Opcode 'ADD'. This can be interpreted as A = A+B. Here, 'A' can be a register or memory address and 'B' can be a register, memory reference or a constant. But 'A' and 'B' cannot be memory addresses. There are some one address instructions, which leave the destination and first source implicit. In machine code each instruction has a unique bit pattern, for human consumption, a symbolic representation is used (for example, ADD, SUB, LOAD, etc.).

Instruction Types:

Three main classifications of Instruction types are the following:

- Data Transfer.
- Data Manipulation.
- Control Flow.

Data Transfer:

These types of instructions transfer data among variables (registers, memory, and I/O ports). They do not alter FLAGS. Transfer of 8 and 16-bit data is done using the MOV instruction. Either the source or destination must be a register. The other operand can come from another register, from memory, from immediate data (i.e., a value included in the instruction), or from a memory location "Pointed at" by the register BX. For example, If COUNT is the label of a memory location, then the following are possible assembly-language instructions:

a) Register: move contents of BX to AX
   MOV AX, BX
b) Direct: move the contents of AX to memory
   MOV COUNT, AX
c) Immediate: load CX with the value 0F0H
   MOV CX, 0F0H
d) Memory: load CX with the value at; address 0F0H
   MOV CX, [0F0H]
e) register indirect: move contents of AL; to a memory location in
   BX MOV [BX], AL

Data Manipulation:

These are instructions that modify variable values and are executed within the ALU data path. They modify the flags. Following are examples of data manipulation operations,

1) **Data Manipulation: ADD**

   ADD dest, src

Here, 'dest' is both a source and destination operand (semantics: dest = dest + src). This operation will modify flags as shown below,

ZF: =1, if result=0

SF: =1, if MS bit of result = 1 (sign = negative)

CF: =1, if carry out of MS bit

OF: =1, if result overflowed signed capacity.

2) **Data Manipulation: SUB and CMP**

SUB dest, src (i.e., the operation is dest=dest-src).

-(Zero Flag)ZF: =1, if result=0

SF: =1, if MS bit of result = 1 (sign = negative)

CF: =1, if borrow into MS bit

OF: =1, if result overflowed signed capacity

CMP dest, src (compare)

This modifies flags only to reflect dest – src.

3) **Data Manipulation: Logical Operations**

Syntax: BOOLEAN_MNEMONIC dest, src.

Semantics: dest= dest BOOLEAN_MNEMONIC src.

Example:

1) AND AL, 80h /* Doing AND operation with 80H and Contents of AL */
2) OR AL, BH /*Doing OR operation with the Contents of AL and BH registers */
3) XOR AX, BX /*Doing XOR operation with the Contents of AX and BX registers */
4) XOR AH, 0FFh/* Doing XOR operation with 0FFH and the Contents of AH register */

Control Flow:

These instructions determine the next instruction to execute. This allows non-sequential execution. Unconditional & Conditional are two types of switching execution to a different instruction sequence.

Syntax: MNEMONIC address

Example:

1) JMP 00FF // Jump to the address 00FF.
2) CALL 00FF // calls a subroutine and saves the return address on the stack

3) RET // returns from the subroutine to the main program
4) LOOP 00FF // returns from the subroutine to the main program
5) JC 00FF // jump if CF = 1
6) JNC 00FF // jump if CF = 0
7) JZ 00FF // jump if ZF = 1
8) JNZ 00FF // jump if ZF = 0

5. ADDRESSING MODES

Addressing mode indicates a way of locating data or operands. Depending upon the data types used in the instruction and the memory addressing modes, any instruction may belong to one or more addressing modes or some instruction may not belong to any of the addressing modes. Thus the addressing modes describe the types of operands and the way they are accessed for executing an instruction.

The addressing modes for sequential control transfer instructions are explained as follows:

1. **Immediate:** In this type of addressing, immediate data is a part of instruction, and appears in the form of successive bytes or bytes.

   Example: MOV AX, 0005H

   In the above example, 0005H is the immediate data. The immediate data may be 8-bit or 16-bit in size.

2. **Direct:** In the direct addressing mode, a 16-bit memory address (offset) is directly specified in the instruction as a part of it.

   Example: MOV AX, [5000H]

   Here, data resides in a memory location in the data segment, whose effective address may be computed using 5000H as the offset address and the content of DS as the segment address. The effective address, here, is 10H*DS+5000H.

3. **Register:** In register addressing mode, the data is stored in a register and it is referred to using the specific register. All the registers, except IP, may be used in this mode.

   Example: MOV BX, AX.

4. **Register Indirect:** Sometimes, the address of the memory location, which contains data or operand, is determined in an indirect way, using the offset registers. This mode of addressing is known as the register indirect mode. In this addressing mode, the offset address of data is in either BX or SI, or DI registers. The default segment is either DS or ES. The data is supposed to be available at the address pointed to by the content of any of the above registers in the default data segment.

   Example: MOV AX, [BX]

   Here, data is present in a memory location in DS whose offset address is in BX. The effective address of the data is given as 10H*DS+ [BX].

5. **Indexed:** In this addressing mode, the offset of the operand is stored in one of the index registers. DS and ES are the default segments for index register SI and DI respectively. This mode is a special case of the above-discussed register indirect addressing mode.

   Example: MOV AX, [SI]

   Here, data is available at an offset address stored in SI in DS. The effective address, in this case, is computed as 10H*DS+ [SI].

6. **Register Relative:** In this addressing mode, the data is available at an effective address formed by adding an 8-bit or 16-bit displacement with the content of any one of the registers BX, BP, SI, and DI in the default (either DS or ES) segment. The example is given before explains this mode.

   Example: MOV Ax, 50H [BX]

   Here, the effective address is given as 10H*DS+50H+ [BX].

7. **Based Indexed:** The effective address of data is formed, in this addressing mode, by adding the content of a base register (any one of BX or BP) to the content of an index register (any one of SI or DI). The default segment register may be ES or DS.

   Example: MOV AX, [BX] [SI]

Here, BX is the base register and SI is the index register. The effective address is computed as 10H*DS+ [BX] + [SI].

8. **Relative Based Indexed:** The effective address is formed by adding an 8-bit or 16-bit displacement with the sum of contents of any one of the base registers (BX or BP) and any one of the index registers, in a default segment.

   Example: MOV AX, 50H [BX] [SI]

   Here, 50H is an immediate displacement, BX is a base register and SI is an index register. The effective address of data is computed as 160H*DS+ [BX] + [SI] + 50H.For the **control transfer instructions,** the addressing modes depend upon whether the destination location is within the same segment or a different one. It also depends upon the method of passing the destination address to the processor. Basically, there are two addressing modes for the control transfer instructions, viz. inter-segment, and intra-segment addressing modes.

If the location to which the control is to be transferred lies in a different segment other than the current one, the mode is called inter-segment mode. If the destination location lies in the same segment, the mode is called intra-segment.

SELF ASSESSMENT QUESTIONS – 3

9. ________Instructions transfer data among variables.
10. 8086instructions may be categorized as________________and___________.
11. In_______________addressing mode, the data is stored in a register and it is referred using the particular register.
12. In_____________addressing modes, immediate data is a part of instruction, and appears in the form of successive byte or bytes.

6. INSTRUCTION SETS

An instruction is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions that a microprocessor supports is called **Instruction Set**. 8086 has more than **20,000** instructions

Classification of Instruction Set:

- Data Transfer Instructions
- Arithmetic Instructions
- Bit Manipulation Instructions
- Program Execution Transfer Instructions
- String Instructions
- Processor Control Instructions

Data Transfer Instruction:

These instructions are used to transfer data from source to destination. The operand can be a constant, memory location, register, or I/O port address.

- **MOV Des, Src:**

SRC operand can be a register, memory location, or immediate operand. DES can be a register or memory operand. Both Src and Des cannot be memory locations at the same time.

E.g.:

MOV CX, 037A H

MOV AL, BL

MOV BX, [0301 H]

- **PUSH Operand:**

It pushes the operand to the top of the stack.

E.g.: PUSH BX

- **POP Des:**

It pops the operand from the top of the stack to Des. Des can be a general-purpose register, segment register (except CS), or memory location.

E.g.: POP AX

- **XCHG Des, Src:**

This instruction exchanges Src with Des. It cannot exchange two memory locations directly.
E.g.: XCHG DX, AX

- **IN Accumulator, Port Address:**

It transfers the operand from the specified port to the accumulator register. E.g.: IN AX, 0028 H

- **OUT Port Address, Accumulator:**

It transfers the operand from the accumulator to the specified port. E.g.: OUT 0028 H, AX

- **LEA Register, Src:**

It loads a 16-bit register with the offset address of the data specified by the Src.
E.g.: LEA BX, [DI]
This instruction loads the contents of DI (offset) into the BX register

- **LDS Des, Src:**

It loads a 32-bit pointer from the memory source to the destination register and DS.

The offset is placed in the destination register and the segment is placed in DS. To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment.
E.g.: LDS BX, [0301 H]

- **LES Des, Src:**

It loads a 32-bit pointer from the memory source to the destination register and ES.

The offset is placed in the destination register and the segment is placed in ES.

This instruction is very similar to LDS except that it initializes ES instead of DS.

E.g.: LES BX, [0301 H]

- **LAHF:**

It copies the lower byte of the flag register to AH.

- **SAHF:**

It copies the contents of AH to the lower byte of the flag register.

- **PUSHF:**

Pushes flag register to the top of the stack.

- **POPF:**

Pops the stack top to flag register.

Arithmetic Instructions:

- **ADD Des, Src:**

It adds a byte to a byte or a word to word.

It effects AF, CF, OF, PF, SF, and ZF flags.

E.g.:

ADD AL, 74H

ADD DX, AX

ADD AX, [BX]

- **ADC Des, Src:**

It adds the two operands with CF. It effects AF, CF, OF, PF, SF, and ZF flags. E.g.:

ADC AL, 74H

ADC DX, AX

ADC AX, [BX]

- **SUB Des, Src:**

It subtracts a byte from another byte or a word from a word. It effects AF, CF, OF, PF, SF, and ZF flags. For subtraction, CF acts as a borrow flag.

E.g.:

SUB AL, 74H

SUB DX, AX

SUB AX, [BX]

- **SBB Des, Src:**

It subtracts the two operands and the borrows from the result. It effects AF, CF, OF, PF, SF, and ZF flags.

E.g.:

SBB AL, 74H

SBB DX, AX

SBB AX, [BX]

- **INC Src:**

It will increments the byte or word by one. The operand can be a register or memory location. It effects AF, OF, PF, SF, and ZF flags. CF is not affected.
E.g.: INC AX

- **DEC Src:**

It will decrements the byte or word by one. The operand can be a register or memory location. It effects AF, OF, PF, SF, and ZF flags. CF is not affected.
E.g.: DEC AX

- **AAA (ASCII Adjust after Addition):**

The data entered from the terminal is in ASCII format. In ASCII, 0 – 9 are represented by 30H – 39H. This instruction allows us to add the ASCII codes. This instruction does not have an operand.

- **Other ASCII Instructions:**

AAS (ASCII Adjust after Subtraction)

AAM (ASCII Adjust after Multiplication)

AAD (ASCII Adjust Before Division)

- **DAA (Decimal Adjust after Addition)**

It is used to make sure that the result of adding two BCD numbers is adjusted to be a correct BCD number. It only works on the AL register.

- **DAS (Decimal Adjust after Subtraction)**

It is used to make sure that the result of subtracting two BCD numbers is adjusted to be a correct BCD number. It only works on the AL register.

- **NEG Src:**

It creates a 2's complement of a given number. That means it changes the sign of a number.

- **CMP Des, Src:**

It compares two specified bytes or words. The Src and Des can be a constant, register, or memory location. Both operands cannot be in a memory location at the same time. The comparison is done simply by internally subtracting the source from the destination. The value of source and destination does not change, but the flags are modified to indicate the result.

- **MUL Src:**

It is an unsigned multiplication instruction. It multiplies two bytes to produce a word or two words to produce a double word.
AX = AL * Src
DX: AX = AX * Src

This instruction assumes one of the operands in AL or AX. Src can be a register or memory location.

- **IMUL Src:**

It is a signed multiplication instruction.

- **DIV Src:**

It is an unsigned division instruction. It divides word by byte or double word by word. The operand is stored in AX, the divisor is Src and the result is stored as:
AH = remainder AL = quotient

- **IDIV Src:**

It is a signed division instruction.

- **CBW (Convert Byte to Word):**

This instruction converts byte in AL to word in AX. The conversion is done by extending the sign bit of AL throughout AH.

- **CWD (Convert Word to Double Word):**

This instruction converts words in AX to double words in
DX: AX. The conversion is done by extending the sign bit of AX throughout DX.

Bit Manipulation Instructions:

These instructions are used at the bit level. These instructions can be used for:

Testing a zero bit

Set or reset a bit

Shift bits across registers.

- **NOT Src:**

It complements each bit of Src to produce 1's complement of the specified operand. The operand can be a register or memory location.

- **AND Des, Src:**

It performs AND operation of Des and Src. Src can be the immediate number, register, or memory location. Des can be a register or memory location. Both operands cannot be memory locations at the same time. CF and OF become zero after the operation. PF, SF, and ZF are updated

- **OR Des, Src:**

It performs OR operation of Des and Src. Src can be the immediate number, register, or memory location. Des can be a register or memory location. Both operands cannot be memory locations at the same time. CF and OF become zero after the operation. PF, SF, and ZF are updated.

- **XOR Des, Src:**

It performs the XOR operation of Des and Src. Src can be the immediate number, register, or memory location. Des can be a register or memory location. Both operands cannot be memory locations at the same time. CF and OF become zero after the operation. PF, SF, and ZF are updated.

- **SHL Des, Count:**

It shifts bits of byte or word left, by count. It puts zero(s) in LSBs. MSB is shifted into the carry flag. If the number of bits desired to be shifted is 1, then the immediate number 1 can be written in Count. However, if the number of bits to be shifted is more than1, then the count is put in the CL register.

- **SHR Des, Count:**

It shifts bits of a byte or word right, by count. It puts zero(s) in MSBs. LSB is shifted into the carry flag. If the number of bits desired to be shifted is 1, then the immediate number 1 can be written in Count. However, if the number of bits to be shifted is more than1, then the count is put in the CL register.

- **ROL Des, Count:**

It rotates bits of byte or word left, by count. MSB is transferred to LSB and also to CF. If the number of bits desired to be shifted is 1, then the immediate number 1 can be written in Count. However, if the number of bits to be shifted is more than1, then the count is put in the CL register.

- **ROR Des, Count:**

It rotates bits of a byte or word right, by count. LSB is transferred to MSB and also to CF. If the number of bits desired to be shifted is 1, then the immediate number 1 can be written in Count. However, if the number of bits to be shifted is more than1, then the count is put in the CL register.

Program execution Transfer Instructions:

These instructions cause a changes in the sequence of the execution of the instruction. This change can be through a condition or sometimes unconditional. The conditions are represented by flags.

- **CALL Des:**

This instruction is used to call a subroutine or function or procedure. The address of the next instruction after CALL is saved onto the stack.

- **RET**

It returns the control from the procedure to the calling program. Every CALL instruction should have a RET.

- **JMP Des:**

This instruction is used for the unconditional jump option from one place to another.

- **Jxx Des (Conditional Jump):**

All the conditional jumps follow some conditional statements or any instruction that affects the flag.

Mnemonic	Meaning	Jump Condition
JA	Jump if Above	CF = 0 and ZF = 0
JAE	Jump if Above or Equal	CF = 0
JB	Jump if Below	CF = 1
JBE	Jump if Below or Equal	CF = 1 or ZF = 1
JC	Jump if Carry	CF = 1
JE	Jump if Equal	ZF = 1
JNC	Jump if Not Carry	CF = 0
JNE	Jump if Not Equal	ZF = 0
JNZ	Jump if Not Zero	ZF = 0
JPE	Jump if Parity Even	PF = 1
JPO	Jump if Parity Odd	PF = 0
JZ	Jump if Zero	ZF = 1

- **Loop Des:**

This is a looping instruction. The number of times looping is required is placed in the CX register. With each iteration, the contents of CX are decremented. ZF is checked whether to loop again or not.

String Instructions:

String in assembly language is just sequentially stored bytes or words. There is a very strong set of string instructions in 8086. By using these string instructions, the size of the program is considerably reduced.

- **CMPS Des, Src:**

It compares the string bytes or words.

- **SCAS String:**

It scans a string. It compares the String with a byte in AL or with the word in AX.

- **MOVS / MOVSB / MOVSW:**

It causes the moving of a byte or word from one string to another. In this instruction, the source string is in Data Segment and the destination string is in Extra Segment. SI and DI store the offset values for the source and destination index.

- **REP (Repeat):**

This is an instruction prefix. It causes the repetition of the instruction until CX becomes zero.
E.g.: REP MOVSB STR1, STR2
It copies byte by byte contents. REP repeats the operation MOVSB until CX becomes zero.

Processor Control Instructions:

These instructions control the processor itself. 8086 allows controlling certain control flags that cause the processing in a certain direction, and processor synchronization if more than one microprocessor is attached.

- **STC:**

It sets the carry flag to 1.

- **CLC:**

It clears the carry flag to 0.

- **CMC:**

It complements the carry flag.

- **STD:**

It sets the direction flag to 1. If it is set, string bytes are accessed from a higher memory address to a lower memory address.

- **CLD:**

It clears the direction flag to 0. If it is reset, the string bytes are accessed from the lower memory address to the higher memory address.

7. I/O AND PROGRAMMING

The 8086 microprocessor is one of the families of 8086, 80286, 80386, 80486, Pentium, Pentium I, II, and III …. Also referred to as the X86 family. Learning any imperative programming language involves mastering a some of common concepts:

Variables

Declaration/definition

Assignment

assigning values to variables

Input/Output:

Displaying messages, Displaying variable values

Control flow:

If – then Loops

Subprograms:

Definition and Usage Programming in assembly language involves mastering the same concepts and a few other issues.

Variables

For the moment we will skip details of variable declaration and simply use the 8086 registers as the variables in our programs. Registers have predefined names and do not need to be declared.

The 8086 has 14 registers. Each of these is a 16-bit register. Initially, we will use four of them – the so-called general-purpose registers:

ax, bx, cx, dx

These four 16-bit registers can also be treated as eight 8-bit registers:

ah, al, bh, bl, ch, cl, dh, dl.

Assignment

Consider the assignment : x = 42 ;y = 24;z = x + y; In assembly language we carry out the same operation but we use an instruction to denote the assignment operator ("="). The above assignments would be carried out in 8086assembly language as follows

```
mov x, 42
mov y, 24
add z, x
add z, y
```

The mov Instruction carries out the assignment. It allows us to place a number in a register or in a memory location (a variable) i.e. it assigns a value to a register or variable.

The 8086 provides a variety of arithmetic instructions. For the moment, we only consider a few of them. To carry out arithmetic such as addition or subtraction, you use the appropriate instruction. In assembly language, you can only carry out a single arithmetic operation at a time. This means that if you wish to evaluate an expression such as:

$$z = x + y + w - v$$

You will have to use 3 assembly language instructions – one for each arithmetic operation. These instructions combine assignment with the arithmetic operation.

Example:

```
mov ax, 5      ; load 5 into ax
add ax, 3      ; add 3 to the contents of ax,
               ; ax now contains 8
inc ax         ; add 1 to ax
               ; ax now contains 9
dec ax         ; subtract 1 from ax
               ; ax now contains 8
sub ax, 6      ; subtract 4 from ax
               ; ax now contains 2.
```

The add instruction adds the source operand to the destination operand, leaving the result in the destination operand.

The destination operand is always the first operand is 8086 assembly language.

The inc instruction takes one operand and adds 1 to it. It is provided because of the frequency of adding 1 to an operand in programming. The dec instruction like inc takes one operand

and subtracts 1 from it. This is also a frequent operation in programming. The subinstruction subtracts the source operand from the destination operand leaving the result in the destination operand.

Input/output

Each microprocessor provides instructions for I/O with the devices that are attached to it, e.g. the keyboard and screen. The 8086 provides the instructions in for input and out for output. These instructions are quite complicated to use, so usually use the operating system to do I/O for us instead. In assembly language, we must have a mechanism to call the operating system to carry out I/O. In addition, must be able to tell the operating system what kind of I/O operation wish to carry out, e.g. to read a character from the keyboard, to display a character or string on the screen, or to do disk I/O. In 8086 assembly language, we do not call operating system subprograms by name; instead, we use a software interrupt mechanism. An interrupt signals the processor to suspend its current activity (i.e. running your program) and to pass control to an interrupt service program (i.e. part of the operating system).

A software interrupt is one generated by a program (as opposed to one generated by hardware). The 8086 **int** instruction generates a software interrupt. It uses a single operand which is a number indicating which MS-DOS subprogram is to be invoked. For I/O and some other operations, the number used is **21h.** Thus, the instruction **int 21h** transfers control to the operating system, to a subprogram that handles I/O operations. This subprogram handles a variety of I/O operations by calling appropriate subprograms. This means that you must also specify which I/O operation (e.g. read a character, display a character) you wish to carry out. This is done by placing a specific number in a register.

The ah register is used to pass this information.

For example, the subprogram to display a character is the subprogram number **2h.**

This number must be stored in the ah register. We are now in a position to describe character output.

When the I/O operation is finished, the interrupt service program terminates, and our program will be resumed at the instruction following int.

Character Output

The task here is to display a single character on the screen. There are three elements involved in carrying out this operation using the int instruction:

1) We specify the character to be displayed. This is done by storing the character's ASCII code in a specific 8086 register. In this case, we use the **dl** register, i.e. we use dl to pass a parameter to the output subprogram.
2) We specify which of MS-DOS's I/O subprograms we wish to use. The subprogram to display a character is subprogram number **2h**. This number is stored in the ah register.
3) We request MS-DOS to carry out the I/O operation using the int instruction. This means that we **interrupt** our program and transfer control to the MS-DOS subprogram that we have specified using the ah register.

Example:

Write a code fragment to display the character 'a' on the screen:

```
mov dl, 'a' ;     dl = 'a'
mov ah, 2h ;      character output subprogram
int 21h ;         call MS-DOS output character
```

As you can see, this simple task is quite complicated in assembly language.

Character Input

The task here is to read a single character from the keyboard. There are also three elements involved in performing character input:

1. As for character output, we specify which of MS-DOS's I/O subprograms we wish to use, i.e. the character input from the keyboard subprogram. This is MS-DOS subprogram number **1h.** This number must be stored in the ah register.
2. We call MS-DOS to carry out the I/O operation using the int instruction for character output.
3. The MS-DOS subprogram uses the al register to store the character it reads from the keyboard.

Example:

Write a code fragment to read a character from the keyboard:

```
mov ah, 1h ; keyboard input subprogram
int 21h; character input; character is stored in al
```

The following example combines the two previous ones, by reading a character from the keyboard and displaying it.

SELF ASSESSMENT QUESTIONS – 4

13. The interrupt generated by a program is called__________________.
14. In 8086____________________instruction generates a software interrupt.

8. SUMMARY

Let us recapitulate the important concepts discussed in this unit:

- System software is computer software designed to operate and control the computer hardware and to provide a platform for running application software.
- Software development tools like a compiler, debugger, and loader are examples of system software and software that allows users to do things like to create text documents, playing games, listening to music, or surfing the web is called application software.
- 8086 has 16-bit ALU; this means 16-bit numbers are directly processed by 8086. It has a 16-bit data bus, so it can read data or write data to memory or I/O ports either 16 bits or 8 bits at a time. It has 20 address lines, so it can address up to 220.
- The 8086is a 16-bit microprocessor chip designed by Intel between early 1976 and mid-1978.
- The entire group of instructions that a microprocessor supports is called Instruction Set. 8086 has more than 20,000 instructions.
- The 8086 microprocessor is one of the families of 8086, 80286, 80386, 80486, Pentium, Pentium I, II, and III …. Also referred to as the X86 family.

9. GLOSSARY

Address: A number, character, or group of characters that identifies a given device or a storage location that may contain a piece of data or a program step.

Architecture: The organizational structure of a system or component. See: component, module, subprogram, routine

ASCII: American Standard Code for Information Interchange.

Data: Representations of facts, concepts, or instructions in a manner suitable for communication, interpretation, or processing by humans or by automated means.

Instruction: A program statement that causes a computer to perform a particular operation or set of operations.

Software: Software is a set of instructions or programs written to carry out a certain tasks on digital computers.

10. TERMINAL QUESTIONS

Short Answer Questions

1. Describe the internal architecture of Intel 8086.
2. Describe the Pin diagram of 8086.
3. Explain three different instruction types.
4. List different addressing modes and explain each.
5. Explain different instruction sets in detail.

10.1 ANSWERS

A. Self-Assessment Questions

1. Assembler
2. 1 megabyte
3. Direct
4. 16-bit data, 20-bit address
5. 5. 2900
6. Instruction queue

7. Flag
8. AD0-AD15
9. Data Transfer
10. Sequential control flows instructions and control transfer instructions.
11. Register
12. Immediate
13. Software Interrupt
14. int

B. Short Answer Questions

1. 8086 architecture is divided into Execution unit (EU) and Bus Interface unit (BIU). (Refer section 3 for detail)
2. Refer section 3 Pin diagram of 8086 for detail
3. Instruction types are classified into three. (Refer section 4 for detail)
4. Refer section 5 addressing modes for detail.
5. Refer section 6 Instruction sets for detail.

11. SUGGESTED BOOKS

- Dhamdhere (2002). Systems programming and operating systems Tata McGraw-Hill.
- M. Joseph (2007). **System software**, Firewall Media.