MUFFAKHAM JAH COLLEGE OF ENGINEERING AND TECHNOLOGY

(Affiliated to Osmania University and Recognized by AICTE)

Mount Pleasant, 8-2-249, Road No. 3, Banjara Hills, Hyderabad, Telangana 500034.

MUFFAKHAM JAH
COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE (CS&AI)

COMPUTER ORGANIZATION AND MICROPROCESSOR LAB MANUAL (PC451AD)
B.E IV SEM (2021-2022)

INDEX

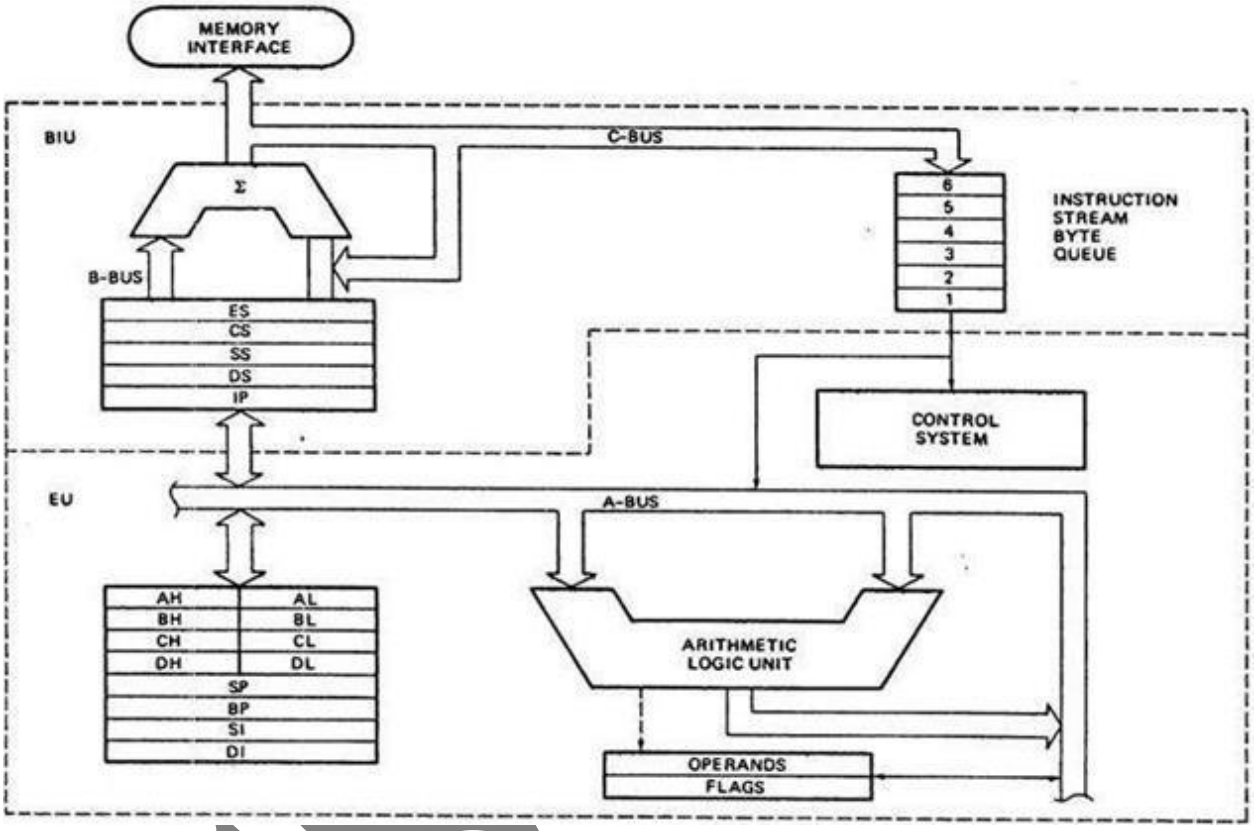
Content Beyond Syllabus Programs		
23	Program to find number of 1's in an 8 bit number	36
24	Program to find 2's complement of a number	37
25	Program to find negative of a number using NEG	38

OVERVIEW OF 8086 MICROPROCESSOR

ARCHITECTURE OF 8086:

As shown in the below figure, the 8086 CPU is divided into two independent functional parts. Dividing the work between these two units' speeds up processing.

- o Bus Interface Unit(BIU)
- o Execution Unit(EU)



The Execution Unit (EU):

- The execution unit of the 8086 tells the BIU where to fetch instructions or data from, decodes instructions,and executes instructions.
- The EU contains control circuitry, which directs internal operations. A decoder in the EU translates instructions fetched from memory into a series of actions, which the EU carries out.
- The EU has a 16-bit arithmetic logic unit (ALU) which can add, subtract, AND, OR, OR, increment, decrement, complement or shift binary numbers.
- The main functions of EU are:
  - o Decoding of Instructions
  - o Execution of instructions

Steps:

- EU extracts instructions from top of queue in BIU
- Decode the instructions

- Generates operands if necessary
- Passes operands to BIU & requests it to perform read or write bus cycles to memory or I/O
- Perform the operation specified by the instruction on operands

Bus Interface Unit (BIU):

- The BIU sends out addresses, fetches instructions from memory, reads data from ports and memory, and writes data to ports and memory.
- In simple words, the BIU handles all transfers of data and addresses on the buses for the execution unit.

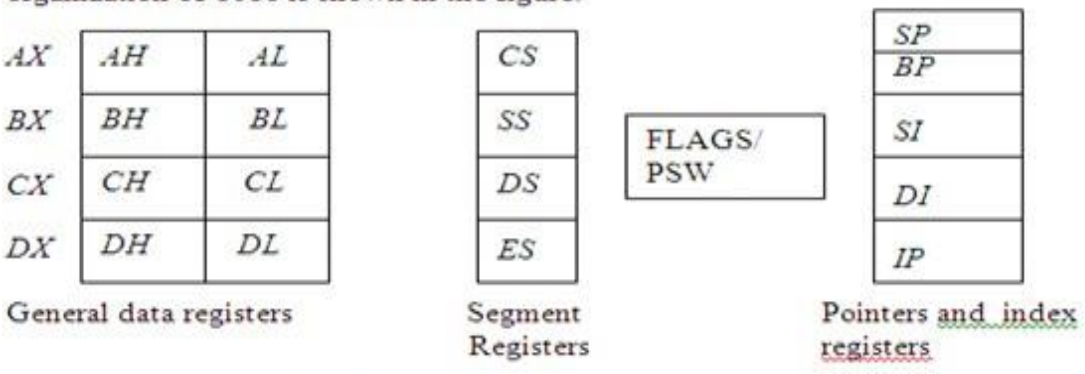
8086 has Pipelining Architecture:

- 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 instruction bytes for the following instructions.
- The BIU stores these pre-fetched bytes in a first-in-first-out register set called a queue. When the EU is ready for its next 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 byte or bytes.
- Except in the case of JMP and CALL instructions, where the queue must be dumped and then reloaded starting from a new address, this pre-fetch and queue scheme greatly speeds up processing.
- Fetching the next instruction while the current instruction executes is called pipelining.

REGISTER ORGANIZATION:

- 8086 has a powerful set of registers known as general purpose registers and special purpose registers. All of them are 16-bit registers.
- The 8086 registers are classified into the following types:
  - o General Data Registers
  - o Segment Registers
  - o Pointers and Index Registers
  - o Flag Register

The register set of 8086 can be categorized into 4 different groups. The register organization of 8086 is shown in the figure.



Register organization of 8086

General Data Registers:

- The registers AX, BX, CX and DX are the general purpose 16-bit registers.
- AX is used as 16-bit accumulator. The lower 8-bit is designated as AL and higher 8-bit is designated as AH. AL can be used as an 8-bit accumulator for 8-bit operation.
- All data register can be used as either 16 bit or 8 bit. BX is a 16 bit register, but BL indicates the lower 8-bit of BX and BH indicates the higher 8-bit of BX.
- The register BX is used as offset storage for forming physical address in case of certain addressing modes.
- The register CX is used default counter in case of string and loop instructions
- DX register is a general purpose register which may be used as an implicit operand or destination in case of a few instructions.

Segment Registers:

- The 8086 architecture uses the concept of segmented memory. 8086 able to address a memory capacity of 1 megabyte and it is byte organized. This 1 megabyte memory is divided into 16 logical segments. Each segment contains 64 kbytes of memory. There are 4 segment registers.
  - **Code Segment Register(CS):** is used for addressing memory location in the code segment of the memory, where the executable program is stored.
  - **Data Segment Register(DS):** points to the data segment of the memory where the data is stored.
  - **Extra Segment Register(ES):** also refers to a segment in the memory which is another data segment in the memory.
  - **Stack Segment Register(SS):** is used for addressing stack segment of the memory. The stack segment is that segment of memory which is used to store stack data.
- While addressing any location in the memory bank, the physical address is calculated from two parts:

  Physical address= segment address + offset address

- The first is segment address, the segment registers contain 16-bit segment base addresses, related to different segments. The second part is the offset value in that segment.

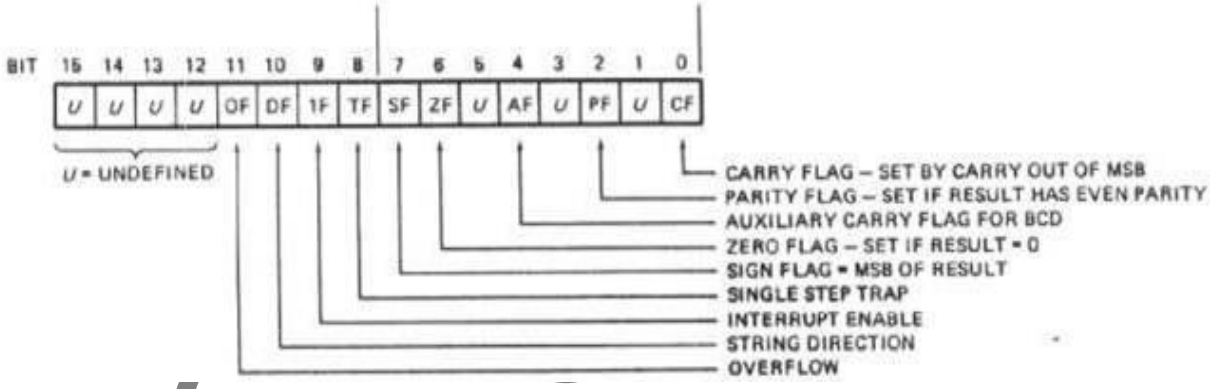
Pointers and Index Registers:

- The pointers registers contain offset within the particular segments. The index and pointer registers are given below:
- **IP(Instruction pointer)-** store memory location of next instruction to be executed. The pointer register IP contains offset within the code segment.
- **BP(Base pointer)-** The pointer register BP contains offset within the data segment.
- **SP(Stack pointer)-** The pointer register SP contains offset within the stack segment.
- The index registers are used as general purpose registers as well as for offset storage in case of indexed, base indexed and relative base indexed addressing modes

- **SI(Source index)-** The register SI is used to store the offset of source data in data segment.
- **DI(Destination index)-** The register DI is used to store the offset of destination in data or extra segment.

8086 flag register and its functions:

- The 8086 flag register contents indicate the results of computation in the ALU. It also contains some flag bits to control the CPU operations.
- A 16 bit flag register is used in 8086. It is divided into two parts .
  - o Condition code or status flags
  - o Machine control flags
- The condition code flag register is the lower byte of the 16-bit flag register. The condition code flag register is identical to 8085 flag register, with an additional overflow flag.
- The control flag register is the higher byte of the flag register. It contains three flags namely direction flag (D), interrupt flag (I) and trap flag (T).



- **SF- Sign Flag:** This flag is set, when the result of any computation is negative. For signed computations the sign flag equals the MSB of the result.
- **ZF- Zero Flag:** This flag is set, if the result of the computation or comparison performed by the previous instruction is zero.
- **PF- Parity Flag:** This flag is set to 1, if the lower byte of the result contains even number of 1's.
- **CF- Carry Flag:** This flag is set, when there is a carry out of MSB in case of addition or a borrow in case of subtraction.
- **AF(Auxiliary Carry Flag):** This is set, if there is a carry from the lowest nibble, i.e, bit three during addition, or borrow for the lowest nibble, i.e, bit three, during subtraction.
- **OF- Over flow Flag**: This flag is set, if an overflow occurs, i.e, if the result of a signed operation is large enough to accommodate in a destination register. The result is of more than 7-bits in size in case of 8-bit signed operation and more than 15-bits in size in case of 16-bit sign operations, and then the overflow will be set.

- **TF- Tarp Flag:** If this flag is set, the processor enters the single step execution mode. The processor executes the current instruction and the control is transferred to the Trap interrupt service routine.
- **IF- Interrupt Flag**: If this flag is set, the mask able interrupts are recognized by the CPU, otherwise they are ignored.
- **D- Direction Flag:** This is used by string manipulation instructions. If this flag bit is '0', the string is processed beginning from the lowest address to the highest address, i.e., auto incrementing mode. Otherwise, the string is processed from the highest address towards the lowest address, i.e., auto decrementing mode.

Addressing modes of 8086:

- Addressing mode indicates a way of locating data or operands.
- The addressing modes describe the types of operands and the way they are accessed for executing an instruction.
- According to the flow of instruction execution, the instructions may be categorized as Sequential control flow instructions and Control transfer instructions
- Sequential control flow instructions are the instructions, which after execution, transfer control to the next instruction appearing immediately after it (in the sequence) in the program. For example, the arithmetic, logic, data transfer and processor control instructions are sequential control flow instructions.
- The control transfer instructions, on the other hand, transfer control to some predefined address or the address somehow specified in the instruction, after their execution. For example, INT, CALL, RET and JUMP instructions fall under this category.
- The addressing modes for sequential control transfer instructions are:

1. **Immediate:** In this type of addressing, immediate data is a part of instruction and appears in the form of successive byte or bytes.
Ex: 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.
Ex: MOV AX, [5000H]
Here, data resides in a memory location in the data segment, whose effective address may be completed using 5000H as the offset address and content of DS as segment address. The effective address here, is 10H * DS + 5000H.

3. **Register**: In register addressing mode, the data is stored in a register and is referred using the particular register. All the registers, except IP, may be used in this mode.
Ex: 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 register. This mode of addressing is known as register indirect mode. In this addressing mode, the offset address of data is in either BX or SI or DI register. 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.

Ex: 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: I**n this addressing mode, offset of the operand is stored in one of the index registers. DS and ES are the default segments for index registers, SI and DI respectively. This is a special case of register indirect addressing mode.

Ex: 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 10*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.

Ex: 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 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.

Ex: 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 or 16-bit displacement with the sum of the contents of any one of the base register (BX or BP) and any one of the index register, in a default segment.

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

Here, 50H is an immediate displacement, BX is base register and SI is an index register the effective address of data is computed as

10H * DS + [BX] + [SI] + 50H

.9. **Intrasegment Direct Mode**: In this mode, the address to which the control is to be transferred lies in the same segment in which the control transfer instruction lies and appears directly in the instruction as an immediate displacement value. In this addressing mode, the displacement is computed relative to the content of the instruction pointer IP.

The effective address to which the control will be transferred is given by the sum of 8 or 16-bit displacement and current content of IP. In the case of jump instruction, if the signed displacement (d) is of 8-bits (i.e –128<d<+128) we term it as short jump and if it is of 16-bits (i.e-32, 768<d<+32,768) it is termed as long jump.

10. **Intrasegment Indirect Mode:** In this mode, the displacement to which the control is to be transferred, is in the same segment in which the control transfer instruction lies, but it is passed to the instruction indirectly. Here, the branch address is found as the content of a register or a memory location. This addressing mode may be used in unconditional branch instructions.

11. **Intersegment Direct:** In this mode, the address to which the control is to be transferred is in a different segment. This addressing mode provides a means of branching from one code segment to another code segment. Here, the CS and IP of the destination address are specified directly in the instruction.

12. **Intersegment Indirect:** In this mode, the address to which the control is to be transferred lies in a different segment and it is passed to the instruction indirectly, i.e contents of a memory block containing four bytes, i.e IP (LSB), IP(MSB), CS(LSB) and CS (MSB) sequentially. The starting address of the memory block may be referred using any of the addressing modes, except immediate mode.

Pin Diagram of 8086:

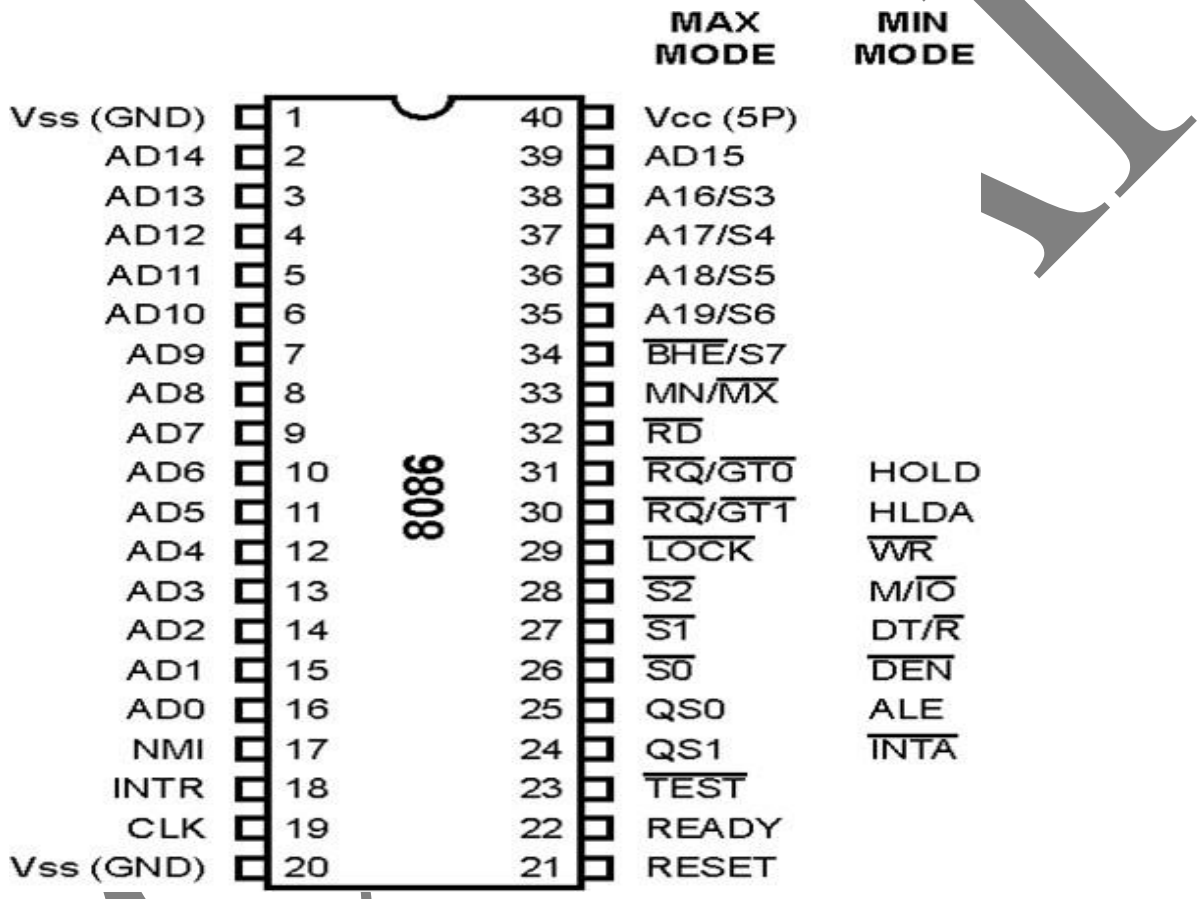
- The 8086 is a 16-bit microprocessor. This microprocessor operates in single processor or multiprocessor configurations to achieve high performance.
- The pin configuration of 8086 is shown in the figure. Some of the pins serve a particular function in minimum mode (single processor mode) and others function in maximum mode (multiprocessor mode).
- The 8086 signals are categorized into 3 types:
  1. Common signals for both minimum mode and maximum mode.
  2. Special signals which are meant only for minimum mode
  3. Special signals which are meant only for maximum mode
- Common Signals for both Minimum mode and Maximum mode:

1. **AD7 -AD0 :** The address/ data bus lines are the multiplexed address data bus and contain the right most eight bit of memory address or data. The address and data bits are separated by using ALE signal.

2. **AD15- AD8 :** The address/data bus lines compose the upper multiplexed address/data bus. This lines contain address bit A15 A8 or data bus D15 D8 . The address and data bits are separated by using ALE signal.

3. **A19 / S6- A18 / S3:** The address/status bus bits are multiplexed to provide address signals A19 /A16 and also status bits S6 /S3 . The address bits are separated from the status bits using the ALE signals. The status bit S6 is always a logic 0, bit S5 indicates the condition of the interrupt flag bit. The S4 and S3 being used for memory access indicate which segment register is presently.

S4	S3	Type of segment register used
0	0	Extra segment
0	1	Stack segment
1	0	Code or no segment
1	1	Data Segment



4. **BHE / S7:**The bus high enable (BHE) signal is used to indicate the transfer of data over the higher order (D15 - D8) data bus. It goes low for the data transfer over D15 - D8 and is used to derive chip select of odd address memory bank or peripherals.

BHE	A0	Indication
0	0	Whole word
0	1	Upper byte from or to odd address
1	0	Lower byte from or to even address
1	1	None

5. **RD(Read):** whenever the read signal is at logic 0, the data bus receives the data from the memory or I/O devices connected to the system

6. **READY:** This is the acknowledgement from the slow devices or memory that they have completed the data transfer operation. This signal is active high.

7. **INTR(Interrupt Request):** Interrupt request is used to request a hardware interrupt of INTR is held high when interrupt enable flag is set, the 8086 enters an interrupt acknowledgement cycle after the current instruction has completed its execution.

8. **TEST :** This input is tested by "WAIT" instruction. If the TEST input goes low; execution will continue. Else the processor remains in an idle state.

9. **NMI(Non-maskable Interrupt):** The non-maskable interrupt input is similar to INTR except that the NMI interrupt does not check for interrupt enable flag is at logic 1, i.e, NMI is not maskable internally by software. If NMI is activated, the interrupt input uses interrupt vector 2.

10. **RESET:** The reset input causes the microprocessor to reset itself. When 8086 reset, it restarts the execution from memory location FFFF0H. The reset signal is active high and must be active for at least four clock cycles.

11. **CLK:** Clock input: The clock input signal provides the basic timing input signal for processor and bus control operation. It is asymmetric square wave with 33% duty cycle.

12. **VCC (+5V):** Power supply for the operation of the internal circuit

13. **GND:** Ground for the internal circuit

14. **MN / MX :** The minimum/maximum mode signal to select the mode of operation either in minimum or maximum mode configuration. Logic 1 indicates minimum mode.

Minimum mode Signals:

The following signals are for minimum mode operation of 8086.

1. **M / IO(Memory/IO):**M / IO signal selects either memory operation or I/O operation. This line indicates that the microprocessor address bus contains either a memory address or an I/O port address. Signal high at this pin indicates a memory operation. This line is logically equivalent to S2 in maximum mode.

2. **INTA(Interrupt acknowledge):** The interrupt acknowledge signal is a response to the INTR input signal. The INTA signal is normally used to gate the interrupt vector number onto the data bus in response to an interrupt request.

3. **ALE(Address Latch Enable):** This output signal indicates the availability of valid address on the address/data bus, and is connected to latch enable input of latches.

4. **DT / R (Data transmit/Receive):** This output signal is used to decide the direction of date flow through the bi-directional buffer. DT / R 1 Indicates transmitting and DT / R 0 indicates receiving the data.

5. **DEN(Data Enable):** Data bus enable signal indicates the availability of valid data over the address/data lines.

6. **(Write):** whenever the write signal is at logic 0, the data bus transmits the data to the memory or I/O devices connected to the system.

7. **HOLD:** The hold input request a direct memory access (DMA). If the hold signal is at logic 1, the micro process stops its normal execution and places its address, data and control bus at the high impedance state.

8. **HLDA:** Hold acknowledgement indicates that 8086 has entered into the hold state.

Maximum mode signal:

- The following signals are for maximum mode operation of 8086.

1. **S2 , S1, S0(Status lines):** These are the status lines that reflect the type of operation being carried out by the processor. These status lines are encoded as follow

S2	S1	S0	Function
0	0	0	Interrupt Acknowledge
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 (In active)

2. **LOCK :** The lock output is used to lock peripherals off the system, i.e, the other system bus masters will be prevented from gaining the system bus.

3. **QS1 and QS0 (Queue status):** The queue status bits shows the status of the internal instruction queue. The encoding of these signals is as follows

QS1	QS0	Function
0	0	No operation,queue is idle
0	1	First byte of opcode
1	0	Queue is empty
1	1	Subsequent byte of opcode

4. **RQ / GT1 and RQ / GT 0 (request/Grant):** The request/grant pins are used by other local bus masters to force the processor to release the local bus at the end of the processors current bus cycle. These lines are bi- directional and are used to both request and grant a DMA operation. RQ / GT 0 is having higher priority than RQ / GT1

EXPERIMENT-1

AIM:- Write a Program to add two 8 bit numbers

PROGRAM:-

```
8000    MOV     AH,00
8002    MOV     SI, A050
8005    MOV     AL,[SI]
8007    INC     SI
8008    MOV     BL, [SI]
800A    ADD     AL,BL
800C    INT     03

        ORG     A050
        DB      _ _
        DB      _ _
```

RESULTS:-

INPUT DATA		OUTPUT	
A050		AX	
A051		BX	
		SI	

EXPERIMENT-2

AIM:- Write a Program to Subtract Two 8 bit numbers

PROGRAM:-

```
8000    MOV     AH,00
8002    MOV     SI, A050
8005    MOV     AL,[SI]
8007    INC     SI
8008    MOV     BL, [SI]
800A    SUB     AL,BL
800C    INT     03

        ORG     A050
        DB      _ _
        DB      _ _
```

RESULTS:-

INPUT	DATA	OUTPUT	
A050		AX	
A051		BX	
		SI	

EXPERIMENT-3

AIM:- Write a Program to Multiply two 8 bit numbers

PROGRAM:-

```
8000    MOV     AH,00
8002    MOV     SI, A050
8005    MOV     AL,[SI]
8007    INC     SI
8008    MOV     BL, [SI]
800A    MUL     BL
800C    INT     03

        ORG     A050
        DB      _ _
        DB      _ _
```

RESULTS:-

INPUT DATA		OUTPUT	
A050		AX	
A051		BX	
		SI	

EXPERIMENT-4

AIM:- Write a Program to Divide two 8 bit numbers

PROGRAM:-

```
8000    MOV     AH,00
8002    MOV     SI, A050
8005    MOV     AL,[SI]
8007    INC     SI
8008    MOV     BL, [SI]
800A    DIV     BL
800C    INT     03

        ORG     A050
        DB      _ _
        DB      _ _
```

RESULTS:-

INPUT DATA		OUTPUT	
A050		AX	
A051		BX	
		SI	

EXPERIMENT-5

AIM:- Write a Program to add two 16 bit numbers

PROGRAM:-

```
8000    MOV     SI,A050
8003    MOV     AX,[SI]
8005    INC     SI
8006    INC     SI
8007    MOV     BX, [SI]
8009    ADD     AX,BX
800B    INT     03

        ORG     A050
        DW      _ _ _ _
        DW      _ _ _ _
```

RESULTS:-

INPUT DATA		OUTPUT	
A050		AX	
A052		BX	
		SI	

EXPERIMENT-6

AIM:- Write a Program to Subtract Two 16 bit numbers

PROGRAM:-

```
8000    MOV     SI,A050
8003    MOV     AX,[SI]
8005    INC     SI
8006    INC     SI
8007    MOV     BX, [SI]
8009    SUB     AX,BX
800B    INT     03

        ORG     A050
        DW      _ _ _ _
        DW      _ _ _ _
```

RESULTS:-

INPUT DATA		OUTPUT	
A050		AX	
A052		BX	
		SI	

EXPERIMENT-

AIM:- Write a Program to Multiply two 16 bit numbers

PROGRAM:-

```
8000    MOV     SI,A050
8003    MOV     DX,0000
8006    MOV     AX,[SI]
8008    INC     SI
8009    INC     SI
800A    MOV     BX, [SI]
800C    MUL     BX
800E    INT     03

        ORG     A050
        DW      _ _ _ _
        DW      _ _ _ _
```

RESULTS:-

INPUT DATA		OUTPUT	
A050		AX	
A052		BX	
		DX	
		SI	

EXPERIMENT-

AIM:- Write a Program to Divide two 16 bit numbers

PROGRAM:-

```
8000    MOV     DX,0000
8003    MOV     SI, A050
8006    MOV     AX,[SI]
8008    INC     SI
8009    INC     SI
800A    MOV     BX, [SI]
800C    DIV     BX
800E    INT     03

        ORG     A050
        DW      _ _ _ _
        DW      _ _ _ _
```

RESULTS:-

INPUT DATA		OUTPUT	
A050		AX	
A052		BX	
		DX	
		SI	

EXPERIMENT-

AIM:- Write a Program to add n 16-Bit numbers.

PROGRAM:-

```
        8000    MOV     AX, 0000
        8003    MOV     SI, A050
        8006    MOV     CX, 0005
        8009    MOV     DX, 0000
Back:   800C    ADD     AX, [SI]
        800E    JNC     8011 (Next)
        8010    INC     DX
Next:   8011    INC     SI
        8012    INC     SI
        8013    LOOP    800C (Back)
        8015    INT     03

                ORG     A050
                DW      _ _ _ _
                DW      _ _ _ _
                DW      _ _ _ _
                DW      _ _ _ _
                DW      _ _ _ _
```

RESULTS:-

INPUT		OUTPUT	
A050		AX	
A052		DX	
A054		CX	
A056		SI	
A058			

EXPERIMENT-24

AIM:- Write a Program to find the largest of two 8-bit numbers

PROGRAM:-

```
        8000    MOV     SI,A050
        8003    MOV     AL, [SI]
        8005    INC     SI
        8006    MOV     BL, [SI]
        8008    CMP     AL,BL
        800A    JNC     800E (Large)
        800C    MOV     AL,BL
Large:  800E    INT     03

                ORG     A050
                DB      _ _
                DB      _ _
```

RESULTS:-

INPUT DATA		OUTPUT	
A050		AX	
A051		BX	
		SI	

EXPERIMENT-25

AIM:- Write a Program to find the smallest of two 8-bit numbers

PROGRAM:-

```
        8000    MOV     SI, A050
        8003    MOV     AL, [SI]
        8005    INC     SI
        8006    MOV     BL, [SI]
        8008    CMP     AL,BL
        800A    JC      800E (Small)
        800C    MOV     AL,BL
Small:  800E    INT     03

                ORG     A050
                DB      _ _
                DB      _ _
```

RESULTS:-

INPUT DATA		OUTPUT	
A050		AX	
A051		BX	
		SI	

EXPERIMENT-26

AIM:- Write a Program to find the largest of two 16-BIT numbers.

PROGRAM:-

```
        8000    MOV     SI, A050
        8003    MOV     AX, [SI]
        8005    INC     SI
        8006    INC     SI
        8007    MOV     BX,[SI]
        8009    CMP     AX,BX
        800B    JNC     800F (Large)
        800D    MOV     AX,BX
Large:  800F    INT     03

                ORG     A050
                DW      _ _ _ _
                DW      _ _ _ _
```

RESULTS:-

INPUT		OUTPUT	
A050		AX	
A052		BX	
		SI	

EXPERIMENT-27

AIM:- Write a Program to find the smallest of two 16-BIT numbers.

PROGRAM:-

```
        8000    MOV     SI, A050
        8003    MOV     AX, [SI]
        8005    INC     SI
        8006    INC     SI
        8007    MOV     BX,[SI]
        8009    CMP     AX,BX
        800B    JC      800F (Small)
        800D    MOV     AX,BX
Small:  800F    INT     03

                ORG     A050
                DW      _ _ _ _
                DW      _ _ _ _
```

RESULTS:-

INPUT		OUTPUT	
A050		AX	
A052		BX	
		SI	

EXPERIMENT-28

AIM:- Write a Program to find the largest of n 8-BIT numbers.

PROGRAM:-

```
        8000    MOV     CL,04
        8002    MOV     SI, A050
        8005    MOV     AL, [SI]
Back:   8007    INC     SI
        8008    MOV     BL,[SI]
        800A    CMP     AL,BL
        800C    JNC     8010 (Large)
        800E    MOV     AL,BL
Large:  8010    LOOP    8007 (Back)
        8012    INT     03

                ORG     A050
                DB      _ _
                DB      _ _
                DB      _ _
                DB      _ _
                DB      _ _
```

RESULTS:-

INPUT		OUTPUT	
A050		AX	
A051		BX	
A052		CX	
A053		SI	
A054			

EXPERIMENT-29

AIM:- Write a Program to find the smallest of n 8-BIT numbers.

PROGRAM:-

```
        8000    MOV     CL,04
        8002    MOV     SI, A050
        8005    MOV     AL, [SI]
Back:   8007    INC     SI
        8008    MOV     BL,[SI]
        800A    CMP     AL,BL
        800C    JC      8010 (Small)
        800E    MOV     AL,BL
Small:  8010    LOOP    8007 (Back)
        8012    INT     03

                ORG     A050
                DB      _ _
                DB      _ _
                DB      _ _
                DB      _ _
                DB      _ _
```

RESULTS:-

INPUT		OUTPUT	
A050		AX	
A051		BX	
A052		CX	
A053		SI	
A054			

EXPERIMENT-30

AIM:- Write a Program to find the largest of n 16-BIT numbers.

PROGRAM:-

```
        8000    MOV     CL,04
        8002    MOV     SI, A050
        8005    MOV     AX, [SI]
Back:   8007    INC     SI
        8008    INC     SI
        8009    MOV     BX,[SI]
        800B    CMP     AX,BX
        800D    JNC     8011 (Large)
        800F    MOV     AX,BX
Large:  8011    LOOP    8007 (Back)
        8013    INT     03

                ORG     A050
                DW      _ _ _ _
                DW      _ _ _ _
                DW      _ _ _ _
                DW      _ _ _ _
                DW      _ _ _ _
```

RESULTS:-

INPUT		OUTPUT	
A050		AX	
A052		BX	
A054		CX	
A056		SI	
A058			

EXPERIMENT-31

AIM:- Write a Program to find the smallest of n 16-BIT numbers.

PROGRAM:-

```
        8000    MOV     CL,04
        8002    MOV     SI, A050
        8005    MOV     AX, [SI]
Back:   8007    INC     SI
        8008    INC     SI
        8009    MOV     BX,[SI]
        800B    CMP     AX,BX
        800D    JC      8011 (Small)
        800F    MOV     AX,BX
Small:  8011    LOOP    8007 (Back)
        8013    INT     03

                ORG     A050
                DW      _ _ _ _
                DW      _ _ _ _
                DW      _ _ _ _
                DW      _ _ _ _
                DW      _ _ _ _
```

RESULTS:-

INPUT		OUTPUT	
A050		AX	
A052		BX	
A054		CX	
A056		SI	
A058			

EXPERIMENT-32

AIM:- Write a Program to perform Block Transfer

PROGRAM:-

```
        8000    MOV     CL,05
        8002    MOV     SI, A050
        8005    MOV     DI, B050
Back:   8008    MOV     AL, [SI]
        800A    MOV     [DI],AL
        800C    INC     SI
        800D    INC     DI
        800E    LOOP    8008 (Back)
        8010    INT     03

                ORG     A050
                DB      _ _
                DB      _ _
                DB      _ _
                DB      _ _
                DB      _ _
```

RESULTS:-

INPUT DATA		OUTPUT	
A050		B050	
A051		B051	
A052		B052	
A053		B053	
A054		B054	

EXPERIMENT-19

AIM:- Write a Program to sort n numbers

PROGRAM:-

```
        8000    MOV     CH,04
Loop1:  8002    MOV     CL,04
        8004    MOV     SI,A050
Loop2:  8007    MOV     AL, [SI]
        8009    INC     SI
        800A    CMP     AL,[SI]
        800C    JNC     8014 (Next)
        800E    XCHG    AL,[SI]
        8010    DEC     SI
        8011    MOV     [SI],AL
        8013    INC     SI
Next:   8014    DEC     CL
        8016    JNZ     8007 (Loop 2)
        8018    DEC     CH
        801A    JNZ     8002 (Loop 1)
        801C    INT     03

                ORG     A050
                DB      _ _
                DB      _ _
                DB      _ _
                DB      _ _
                DB      _ _
```

RESULTS:-

INPUT DATA		OUTPUT	
A050		A050	
A051		A051	
A052		A052	
A053		A053	
A054		A054	

EXPERIMENT-35

AIM:- Write a Program to search a 8 bit number

PROGRAM:-

```
        8000    MOV     CL,05
        8002    MOV     SI,A050
        8005    MOV     AL,45
Back:   8007    MOV     BL, [SI]
        8009    CMP     AL,BL
        800B    JZ      8010 (Found)
        800D    INC     SI
        800E    LOOP    8007 (Back)
Found:  8010    INT     03

                ORG     A050
                DB      _ _
                DB      _ _
                DB      _ _
                DB      _ _
                DB      _ _
```

RESULTS:-

INPUT DATA		OUTPUT	
A050		AX	
A051		BX	
A052		CX	
A053		SI	
A054			

EXPERIMENT-21

AIM:- Write a Program to find number of 1's in an 8 bit number

PROGRAM:-

```
        8000    MOV     SI,A050
        8003    MOV     AL,[SI]
        8005    MOV     CL,08
        8007    MOV     BL,00
Back:   8009    ROL     AL,1
        800B    JNC     800E (NEXT)
        800D    INC     BL
Next:   800E    LOOP    8009 (BACK)
        8010    INT     03

                ORG     A050
                DB      _ _
```

RESULTS:-

INPUT DATA		OUTPUT	
A050		BX	
		SI	

EXPERIMENT-22

AIM:- Write a Program to find 2’s complement of a number

PROGRAM:-

```
8000    MOV     SI,A050
8003    MOV     AL,[SI]
8005    MOV     AH,00
8007    NOT     AL
8009    INC     AL
800B    INT     03

        ORG     A050
        DB      _ _
```

RESULTS:-

INPUT DATA		OUTPUT	
A050		AX	
		SI	

EXPERIMENT-23

AIM:- Write a Program to find negative of a number using NEG

PROGRAM:-

```
8000    MOV     SI,A050
8003    MOV     AL,[SI]
8005    MOV     AH,00
8007    NEG     AL
8009    INT     03

        ORG     A050
        DB      – –
```

RESULTS:-

INPUT DATA		OUTPUT	
A050		AX	
		SI	