

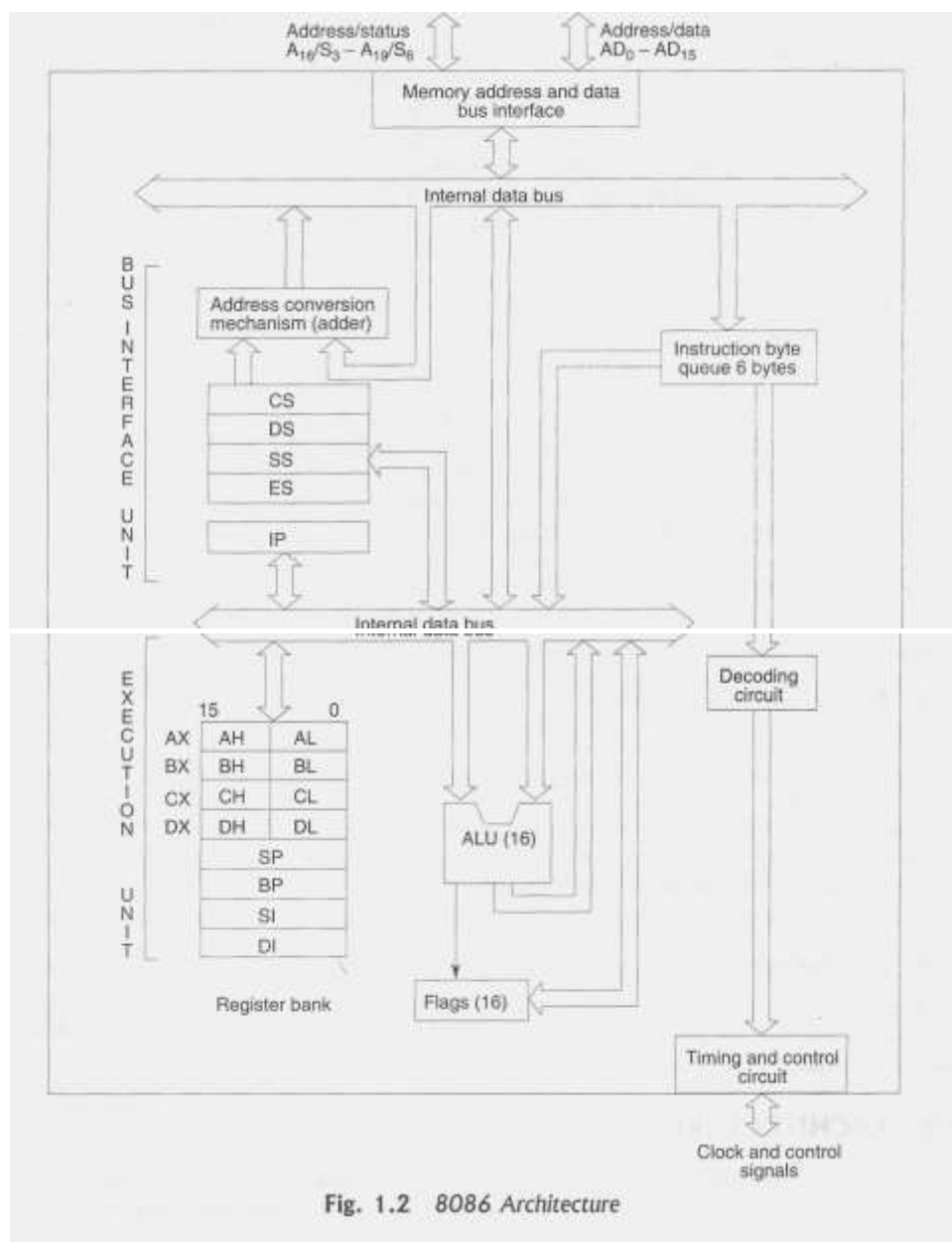
Microprocessors

UNIT III and IV

Q.1 Draw and explain in brief architecture of 8086 microprocessor?

Answer:

The internal architecture 8086 microprocessor is as shown in the figure. The 8086CPU is divided into two independent functional parts, the Bus interface unit (BIU) and execution unit (EU).The Bus Interface Unit contains Bus Interface Logic, Segment registers, Memory addressing logic and a Six byte instruction object code queue. The execution unit contains the Data and Address registers, the Arithmetic and Logic Unit, the Control Unit and flags.



The BIU sends out address, fetches the instructions from memory, read data from ports and memory, and writes the data to ports and memory. In other words the BIU handles all transfers of data and addresses on the buses for the execution unit. The execution unit (EU) of the 8086 tells the BIU where to fetch instructions or data from, decodes instructions and executes instruction. 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 ALU which can add, subtract, AND, OR, XOR, increment, decrement, complement or shift binary numbers. The EU is decoding an instruction or executing an instruction which does not require use of the buses.

The Queue: The BIU fetches up to 6 instruction bytes for the following instructions. The BIU stores these pre-fetched 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 byte(s) 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 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.

Q. 2 Explain register organization in 8086 microprocessor.

Answer:

8086 has a powerful set of registers containing general purpose and special purpose registers. All the registers of 8086 are 16-bit registers. The general purpose registers, can be used either 8-bit registers or 16-bit registers. The general purpose registers are either used for holding the data, variables and intermediate results temporarily or for other purpose like counter or for storing offset address for some particular addressing modes etc. The special purpose registers are used as segment registers, pointers, index registers or as offset storage registers for particular addressing modes. Fig 1.4 shows register organization of 8086. We will categorize the register set into four groups as follows:

General data registers:

The registers AX, BX, CX, and DX are the general 16-bit registers.

AX Register: Accumulator register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the low-order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations, rotate and string manipulation.

BX Register: This register is mainly used as a **base register**. It holds the starting base location of a memory region within a data segment. It is used as offset storage for forming physical address in case of certain addressing mode.

CX Register: It is used as default counter or **count register** in case of string and loop instructions.

DX Register: Data register can be used as a port number in I/O operations and implicit operand or destination in case of few instructions. In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.

General data registers:

AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

General data registers

CS
SS
DS
ES

Segment registers

FLAGS/PSW

SP
BP
SI
DI
IP

Pointers and index registers

Segment registers:

To complete 1Mbyte memory is divided into 16 logical segments. Each segment contains 64Kbyte of memory. There are four segment registers.

Code segment (CS) is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly. The CS register is automatically updated during far jump, far call and far return instructions. It is used for addressing a memory location in the code segment of the memory, where the executable program is stored.

Stack segment (SS) is a 16-bit register containing address of 64KB segment with program stack. By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction. It is used for addressing stack segment of memory. The stack segment is that segment of memory, which is used to store stack data.

Data segment (DS) is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment. DS register can be changed directly using POP and LDS instructions. It points to the data segment memory where the data is resided.

Extra segment (ES) is a 16-bit register containing address of 64KB segment, usually with program data. By default, the processor assumes that the DI register references the ES segment in string manipulation instructions. ES register can be changed directly using POP and LES instructions. It also refers to segment which essentially is another data segment of the memory. It also contains data.

Pointers and index registers. The pointers contain within the particular segments. The pointers IP, BP, SP usually contain offsets within the code, data and stack segments respectively

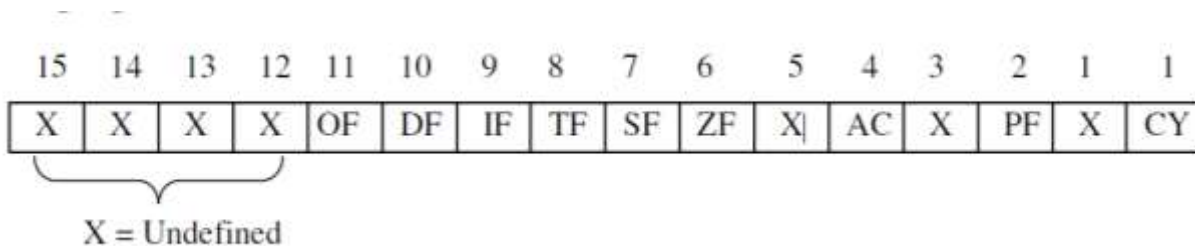
Stack Pointer (SP) is a 16-bit register pointing to program stack in stack segment.

Base Pointer (BP) is a 16-bit register pointing to data in stack segment. BP register is usually used for based, based indexed or register indirect addressing.

Source Index (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data addresses in string manipulation instructions.

Destination Index (DI) is a 16-bit register. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions.

Flag register



Flags Register determines the current state of the processor. They are modified automatically by CPU after mathematical operations, this allows to determine the type of the result, and to determine conditions to transfer

control to other parts of the program. The 8086 flag register as shown in the figure. 8086 has 9 active flags and they are divided into two categories:

1. Conditional Flags
2. Control Flags

Conditional Flags

Conditional flags are as follows:

Carry Flag (CY): This flag indicates an overflow condition for unsigned integer arithmetic. It is also used in multiple-precision arithmetic.

Auxiliary Flag (AC): 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 AC flag is set i.e. carry given by D3 bit to D4 is AC flag. This is not a general-purpose flag, it is used internally by the Processor to perform Binary to BCD conversion.

Parity Flag (PF): This flag is used to indicate the parity of result. If lower order 8-bits of the result contains even number of 1's, the Parity Flag is set and for odd number of 1's, the Parity flag is reset.

Zero Flag (ZF): It is set; if the result of arithmetic or logical operation is zero else it is reset.

Sign Flag (SF): In sign magnitude format the sign of number is indicated by MSB bit. If the result of operation is negative, sign flag is set.

Control Flags

Control flags are set or reset deliberately to control the operations of the execution unit.

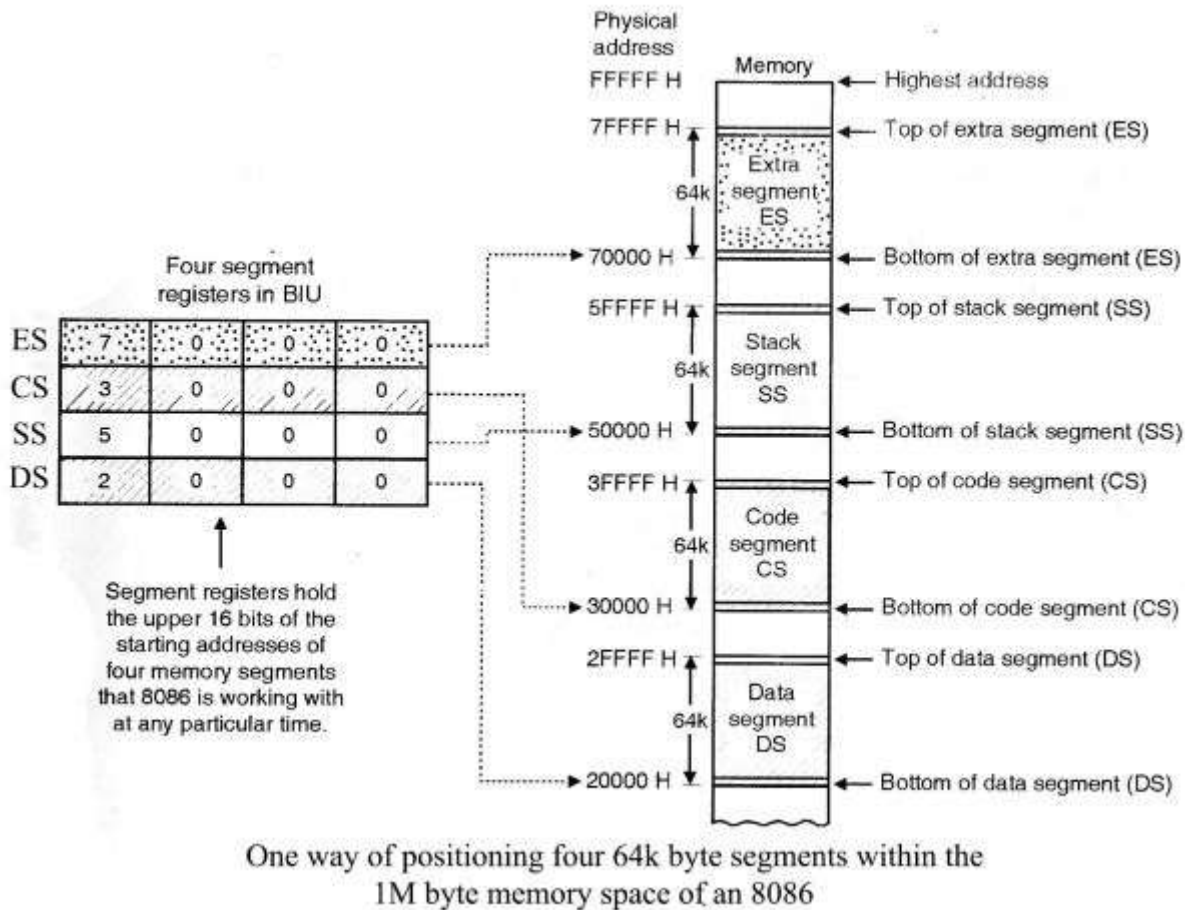
Control flags are as follows:

- **Trap Flag (TF):** It is used for single step control. It allows user to execute one instruction of a program at a time for debugging. When trap flag is set, program can be run in single step mode.
- **Interrupt Flag (IF):** It is an interrupt enable/disable flag. If it is set, the maskable interrupt of 8086 is enabled and if it is reset, the interrupt is disabled. It can be set by executing instruction `sti` and can be cleared by executing `cli` instruction.
- **Direction Flag (DF):** It is used in string operation. If it is set, string bytes are accessed from higher memory address to lower memory address. When it is reset, the string bytes are accessed from lower memory address to higher memory address.

Q. 3 How memory segmentation is implemented in 8086 microprocessor? What are its advantages.

Answer:

The BIU (Bus Interfacing Unit) contains four special purpose registers called as segment registers. These are Code Segment (CS) register, Stack Segment (SS) register, Extra Segment (ES) register and Data Segment (DS) register. All these are 16 bit registers. The number of address lines in 8086 is 20. So the 8086 BIU will send out a 20 bit address in order to access one of the 1,048,576 or 1MB memory locations. But it is interesting to note that the 8086 does not work the whole 1MB memory at any given time. However it works with only four 64 KB segments within the whole 1 MB memory. The four segment registers actually contain the upper 16 bits of the starting addresses of the four memory segments of 64 KB each with which the 8086 is working at that instant of time. A segment is a logical unit of memory that may be up to 64 kilo bytes long. Each segment is made up of memory contiguous memory locations. It is independent, separately addressable unit. Starting addresses will always be changing. They are not fixed. Figure shows one of the possible ways to position the four 64 KB segments within the 1 MB memory space of 8086.



There is no restriction on the locations of these segments in the memory. These segments can be separate from each other or they can overlap. In the users program there can be many segments but 8086 can deal with only four of them at any given time because it has only four segment registers. Whenever the segment orientation is to be changed, the base addresses have to be changed and load the upper 16 bits into the corresponding segment registers. Segment registers are very useful for large programming tasks that require isolation of program code from the data code or isolation of module data from the stack information etc. Segmentation builds relocatable and re-entrant programs easily. In many cases the task of relocating a program simply requires moving the program code and then adjusting the code segment register to point to the base of the new code area.

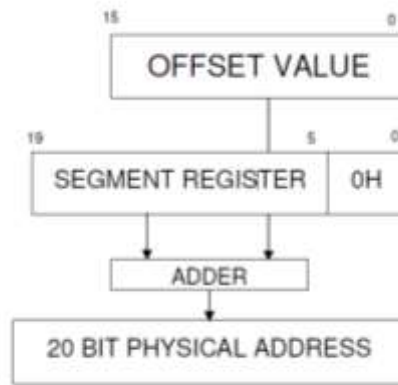
Some of the advantages of memory segmentation in the 8086 are as follows:

- With the help of memory segmentation a user is able to work with registers having only 16-bits.
- By memory segmentation the various portions of a program can be of more than 68kb.
- The data and the user code can be stored separately allowing for more flexibility.
- Also due to segmentation the logical address range is from 0000H to FFFFH the code can be loaded at any location in the memory.

Q.4 How physical address is generated in 8086 microprocessor from logical address (offset address) and base address?

Answer:

The 8086 addresses a segmented memory. The complete physical address which is 20-bits long is generated using segment and offset registers each of the size 16-bit. The content of a segment register also called as segment address, and content of an offset register also called as offset address. To get total physical address, put the lower nibble 0H to segment address and add offset address. The fig 1.3 shows formation of 20-bit physical address.

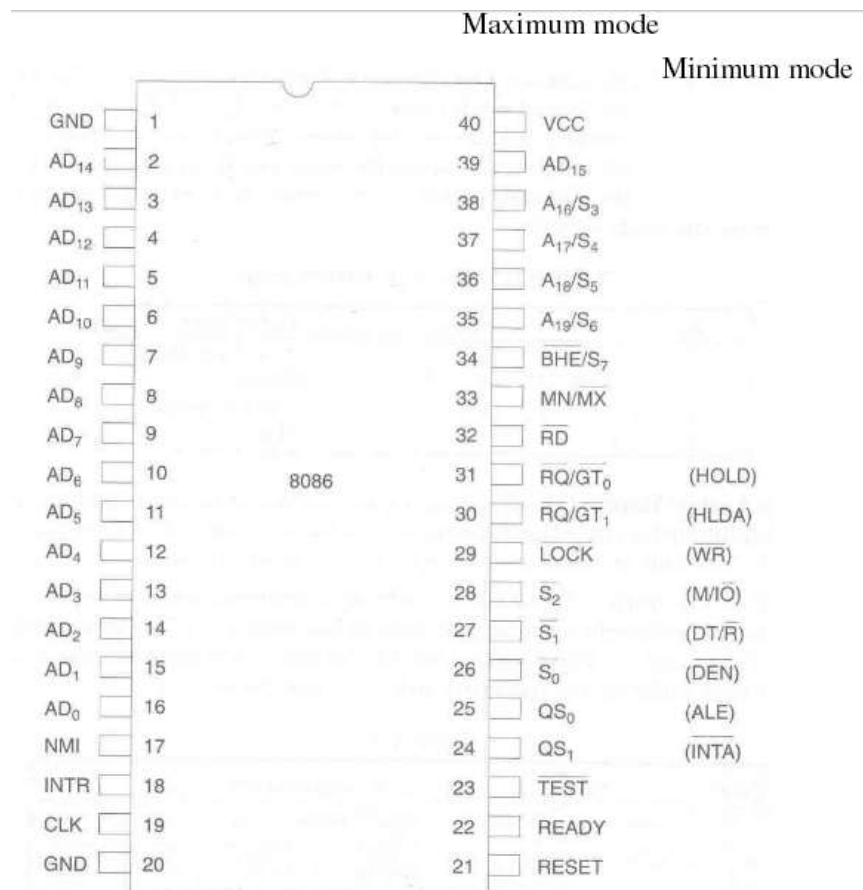


Q. 5 Explain various signals of 8086 microprocessor?

Answer:

The 8086 Microprocessor is a 16-bit CPU available in 3 clock rates, i.e. 5, 8 and 10MHz, packaged in a 40 pin Cerdip or plastic package. The 8086 Microprocessor operates in single processor or multiprocessor configurations to achieve high performance. The pin configuration is as shown in fig1. Some of the pins serve a particular function in minimum mode (single processor mode) and others function in maximum mode (multiprocessor mode) configuration.

The 8086 signals can be categorized in three groups. The first are the signals having common functions in minimum as well as maximum mode, the second are the signals which have special functions in minimum mode and third are the signals having special functions for maximum mode the following signal description are common for both the minimum and maximum modes.



AD15-AD0: These are the time multiplexed memory I/O address and data lines. Address remains on the lines during T1 state, while the data is available on the data bus during T2,T3, TW and T4. Here T1, T2, T3, T4 and TW are the clock states of a machine cycle. TW is await state. These lines are active high and float to a tri-state during interrupt acknowledge and local bus hold acknowledge cycles.

A19/S6, A18/S5, A17/S4, A16/S3: These are the time multiplexed address and status lines. During T1, these are the most significant address lines or memory operations. During I/O operations, these lines are low. During memory or I/O operations, status information is available on those lines for T2, T3, TW and T4. The status of the interrupt enable flag bit(displayed on S5) is updated at the beginning of each clock cycle. The S4 and S3combined indicate which segment register is presently being used for memory accesses as shown in Table.

S4	S3	Indication
0	0	Alternate Data
0	1	Stack
1	0	Code or none
1	1	Data

These lines float to tri-state off (tri-stated) during the local bus hold acknowledge. The status line S6 is always low (logical). The address bits are separated from the status bits using latches controlled by the ALE signal.

BHE’/S7-Bus High Enable/Status: The bus high enable signal is used to indicate the transfer of data over the higher order (D15-D8) data bus as shown in Table 1.2. It goes low for the data transfers over D15-D8 and is used to derive chip selects of odd address memory bank or peripherals. BHE is low during T1 for read, write and interrupt acknowledge cycles, when- ever a byte is to be transferred on the higher byte of the data bus. The status information is available during T2, T3 and T4. The signal is active low and is tri-stated during 'hold'. It is low during T1 for the first pulse of the interrupt acknowledges cycle.

BHE	A ₀	Indication
0	0	Whole Word
0	1	Upper byte from or to odd address
1	0	Upper byte from or to even address
1	1	None

RD’-Read: Read signal, when low, indicates the peripherals that the processor is performing a memory or I/O read operation. RD is active low and shows the state for T2,T3, TW of any read cycle. The signal remains tri-stated during the 'hold acknowledge'.

READY: This is the acknowledgement from the slow devices or memory that they have completed the data transfer. The signal made available by the devices is synchronized by the 8284A clock generator to provide ready input to the 8086. The signal is active high.

INTR-interrupt Request: This is a level triggered input. This is sampled during the last clock cycle of each instruction to determine the availability of the request. If any interrupt request is pending, the processor enters the interrupt acknowledge cycle. This can be internally masked by resetting the interrupt enable flag. This signal is active high and internally synchronized.

TEST: This input is examined by a 'WAIT' instruction. If the TEST input goes low, execution will continue, else, the processor remains in an idle state. The input is synchronized internally during each clock cycle on leading edge of clock.

NMI-Non-maskable Interrupt: This is an edge-triggered input which causes a Type2 interrupt. The NMI is not maskable internally by software. A transition from low to high initiates the interrupt response at the end of the current instruction. This input is internally synchronized.

RESET: This input causes the processor to terminate the current activity and start execution from FFFF0H. The signal is active high and must be active for at least four clock cycles. It restarts execution when the RESET returns low. RESET is also internally synchronized.

CLK-Clock Input: The clock input provides the basic timing for processor operation and bus control activity. It's an asymmetric square wave with 33% duty cycle. The range of frequency for different 8086 versions is from 5MHz to 10MHz.

VCC :+5V power supply for the operation of the internal circuit. **GND** ground for the internal circuit.

MN/MX' :The logic level at this pin decides whether the processor is to operate in either minimum (single processor) or maximum (multiprocessor) mode. The following pin functions are for the minimum mode operation of 8086.

M/IO' -Memory/IO: This is a status line logically equivalent to S2 in maximum mode. When it is low, it indicates the CPU is having an I/O operation, and when it is high, it indicates that the CPU is having a memory operation. This line becomes active in the previous T4 and remains active till final T4 of the current cycle. It is tri-stated during localbus "hold acknowledge".

INTA -Interrupt Acknowledge: This signal is used as a read strobe for interrupt acknowledge cycles. In other words, when it goes low, it means that the processor has accepted the interrupt. It is active low during T2, T3 and TW of each interrupt acknowledge cycle.

ALE-Address latch Enable: This output signal indicates the availability of the valid address on the address/data lines, and is connected to latch enable input of latches. This signal is active high and is never tristated.

DT/R' -Data Transmit/Receive: This output is used to decide the direction of data flow through the transceivers (bidirectional buffers). When the processor sends out data, this signal is high and when the processor is receiving data, this signal is low. Logically, this is equivalent to S1 in maximum mode. Its timing is the same as M/I/O. This is tristated during 'hold acknowledge'.

DEN-Data Enable This signal indicates the availability of valid data over the address/data lines. It is used to enable the transceivers (bidirectional buffers) to separate the data from the multiplexed address/data signal. It is active from the middle of T2 until the middle of T4. DEN is tri-stated during 'hold acknowledge' cycle.

HOLD, HLDA-Hold/Hold Acknowledge: When the HOLD line goes high, it indicates to the processor that another master is requesting the bus access. The processor, after receiving the HOLD request, issues the hold acknowledge signal on HLDA pin, in the middle of the next clock cycle after completing the current bus (instruction) cycle. At the same time, the processor floats the local bus and control lines. When the processor detects the HOLD line low, it lowers the HLDA signal. HOLD is an asynchronous input, and it should be

externally synchronized.If the DMA request is made while the CPU is performing a memory or I/O cycle, it will release the local bus during T4 provided:

1. The request occurs on or before T 2 state of the current cycle.
2. The current cycle is not operating over the lower byte of a word (or operating on an odd address).
3. The current cycle is not the first acknowledge of an interrupt acknowledge sequence.
4. A Lock instruction is not being executed. So far we have presented the pin descriptions of 8086 in minimum mode.

The following pin functions are applicable for maximum mode operation of 8086.

S2, S1, S0 -Status Lines: These are the status lines which reflect the type of operation, being carried out by the processor. These become active during T4 of the previous cycle and remain active during T1 and T2 of the current bus cycle. The status lines return to passive state during T3 of the current bus cycle so that they may again become active for the next bus cycle during T4. Any change in these lines during T3 indicates the starting of a new cycle, and return to passive state indicates end of the bus cycle. These status lines are encoded in table.

S ₂	S ₁	S ₀	Indication
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

LOCK: This output pin indicates that other system bus masters will be prevented from gaining the system bus, while the LOCK signal is low. The LOCK signal is activated by the 'LOCK' prefix instruction and remains active until the completion of the next instruction. This floats to tri-state off during "hold acknowledge". When the CPU is executing a critical instruction which requires the system bus, the LOCK prefix instruction ensures that other processors connected in the system will not gain the control of the bus. The 8086, while executing the prefixed instruction, asserts the bus lock signal output, which may be connected to an external bus controller.

QS1, QS0-Queue Status: These lines give information about the status of the code prefetch queue. These are active during the CLK cycle after which the queue operation is performed. These are encoded as shown in Table.

QS ₁ ,	QS ₀	Indication
0	0	No operation
0	1	First byte of opcode from the queue
1	0	Empty queue
1	1	Subsequent byte from the queue

This modification in a simple fetch and execute architecture of a conventional microprocessor offers an added advantage of *pipelined processing* of the instructions. The 8086 architecture has a 6-byte instruction prefetch queue. Thus even the largest (6-bytes) instruction can be prefetched from the memory and stored in the prefetch queue. This results in a faster execution of the instructions. In 8085, an instruction (opcode and operand) is fetched, decoded and executed and only after the execution of this instruction, the next one is fetched. By prefetching the instruction, there is a considerable speeding up in instruction execution in 8086. This scheme is known as *instruction pipelining*. At the starting the CS:IP is loaded with the required address from which the

execution is to be started. Initially, the queue will be empty and the microprocessor starts a fetch operation to bring one byte (the first byte) of instruction code, if the CS:IP address is odd or two bytes at a time, if the CS:IP address is even. The first byte is a complete opcode in case of some instructions (one byte opcode instruction) and it is a part of opcode, in case of other instructions (two byte long opcode instructions), the remaining part of opcode may lie in the second byte. But invariably the first byte of an instruction is an opcode. These opcodes along with data are fetched and arranged in the queue. When the first byte from the queue goes for decoding and interpretation, one byte in the queue becomes empty and subsequently the queue is updated. The microprocessor does not perform the next fetch operation till at least two bytes of the instruction queue are emptied. The instruction execution cycle is never broken for fetch operation. After decoding the first byte, the decoding circuit decides whether the instruction is of single opcode byte or double opcode byte. If it is single opcode byte, the next bytes are treated as data bytes depending upon the decoded instruction length, otherwise, the next byte in the queue is treated as the second byte of the instruction opcode. The second byte is then decoded in continuation with the first byte to decide the instruction length and the number of subsequent bytes to be treated as instruction data. The queue is updated after every byte is read from the queue but the fetch cycle is initiated by BIU only if at least, two bytes of the queue are empty and the EU may be concurrently executing the fetched instructions.

The next byte after the instruction is completed is again the first opcode byte of the next instruction. A similar procedure is repeated till the complete execution of the program. The main point to be noted here is, that the fetch operation of the next instruction is overlapped with the execution of the current instruction. As shown in the architecture, there are two separate units, namely, execution unit and bus interface unit. While the execution unit is busy in executing an instruction, after it is completely decoded, the bus interface unit may be fetching the bytes of the next instruction from memory, depending upon the queue status.

RQ/GT0, RQ/GT1-ReQuest/Grant: These pins are used by other local bus masters, in maximum mode, to force the processor to release the local bus at the end of the processor's current bus cycle. Each of the pins is bidirectional with RQ/GT0 having higher priority than RQ/GT1, RQ/GT pins have internal pull-up resistors and may be left unconnected. The request! grant sequence is as follows:

1. A pulse one clock wide from another bus master requests the bus access to 8086.
2. During T4 (current) or T1 (next) clock cycle, a pulse one clock wide from 8086 to the requesting master, indicates that the 8086 has allowed the local bus to float and that it will enter the "hold acknowledge" state at next clock cycle. The CPU's bus interface unit is likely to be disconnected from the local bus of the system.
3. A one clock wide pulse from another master indicates to 8086 that the 'hold' request is about to end and the 8086 may regain control of the local bus at the next clock cycle.

Thus each master to master exchange of the local bus is a sequence of 3 pulses. There must be at least one dead clock cycle after each bus exchange. The request and grant pulses are active low. For the bus requests those are received while 8086 is performing memory or I/O cycle, the granting of the bus is governed by the rules as discussed in case of HOLD, and HLDA in minimum mode.

Q. 6 What are minimum and maximum mode of 8086 microprocessor?

Ans:

Minimum Mode:

- The microprocessor 8086 is operated in minimum mode by strapping its MN/MX pin to logic 1.
- In this mode, all the control signals are given out by the microprocessor chip itself.
- There is a single microprocessor in the minimum mode system.

- The remaining components in the system are latches, trans receivers, clock generator, memory and I/O devices. Latches are generally buffered output D-type flip-flops like 74LS373 or 8282.
- They are used for separating the valid address from the multiplexed address/data signals and are controlled by the ALE signal generated by 8086.

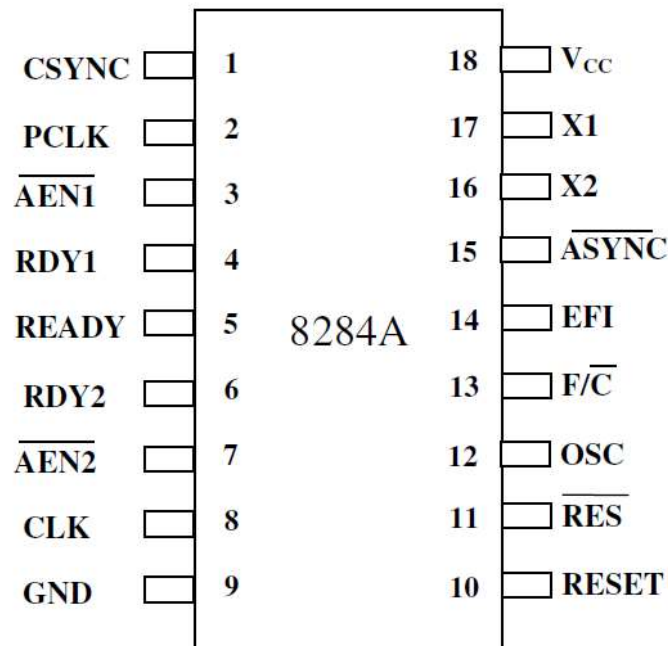
Maximum mode:

- In the maximum mode, the 8086 is operated by strapping the MN/MX pin to ground.
- In this mode, the processor derives the status signal S2, S1, S0.
- Another chip called bus controller derives the control signal using this status information.
- In the maximum mode, there may be more than one microprocessor in the system configuration. The components in the system are same as in the minimum mode system.
- The basic function of the bus controller chip IC8288, is to derive control signals like RD and WR (for memory and I/O devices), DEN, DT/R, ALE etc. using the information by the processor on the status lines.
- The bus controller chip has input lines S2, S1, S0 and CLK.
- These inputs to 8288 are driven by CPU.
- It derives the outputs ALE, DEN, DT/R, MRDC, MWTC, AMWC, IORC, IOWC and AIOWC. The AEN, IOB and CEN pins are especially useful for multiprocessor systems. AEN and IOB are generally grounded. CEN pin is usually tied to +5V. The significance of the MCE/PDEN output depends upon the status of the IOB pin.
- INTA pin used to issue two interrupt acknowledge pulses to the interrupt controller or to an interrupting device.

Q. 7 Write short note on 8284 clock generator.

Answer:

The 8284A is an ancillary component to the 8086/8088 microprocessor. Without the clock generator, many additional circuits to generate the clock (CLK in an 8086/8088 based system. The clock Generator 8284A provides the following basic functions or signals: clock generation, RESEST synchronization, READY synchronization, and a TTL level peripheral clock signal.



The 8284A is an 18-pin integrated circuit designed specifically for use with the 8086/8088 microprocessors as shown in fig1. The following is a list of each pin and its function.

AEN1* and AEN2*: The address enable pins are provided to qualify the ready signals. RDY1 and RDY2, respectively. Which are used to cause wait states, along with the RDY1 and RDY2 inputs. Wait states are generated by the READY pin of the 8086/8088 microprocessor. This is controlled by these two inputs.

RDY1 and RDY2:The bus ready inputs are provided in conjunction with the AEN1* and AEN2* pins to cause wait states in an 8086/8088 microprocessor based system.

ASYN*: The ready synchronization selection input selects either one or two stages of synchronization for the RDY1 and RDY2 inputs.

READY: Ready is an output pin that connects to the 8086/8088 microprocessor READY input. This signal is synchronized with the RDY1 and RDY2 inputs.

X1 and X2: The Crystal Oscillator pins connect to an external crystal used as the timing source for the clock generator and all its functions.

F/C*: The Frequency/Crystal select input results the clocking source for the 8284A. If this pin is held high, an external clock is provided to the EFI input pin, and if it is held low, the internal crystal oscillator provides the timing signal.

EFI: The External Frequency input is used when the F/C is pulled high. EFI supplies the timing whenever the F/C* pin is high.

CLK: The clock output pin provides CLK input signal to the 8086/8088 microprocessors and other components in the system. The CLK pin has an output signal that is one-third of the crystal or EFI input frequency and has a 33 percent duty cycle, which is required by the 8086/8088 microprocessors.

PCLK: The Peripheral Clock signal is one-sixth the crystal or EFI input frequency and has a 50 percent duty cycle. The PCLK output provides a clock signal to the peripheral equipment in the system.

OSC: The Oscillator output is a TTL level signal that is at the same frequency as the crystal or EFI input. (The OSC output provides and EFI input to other 8284A clock generators in some multiple processor systems).

RES*:The reset input is an active-low input to the 8284A. The RES* pin is often connected an RC network that provides power-on resetting.

RESET: The Reset output is connected to the 8086/8088 microprocessors RESET input pin.

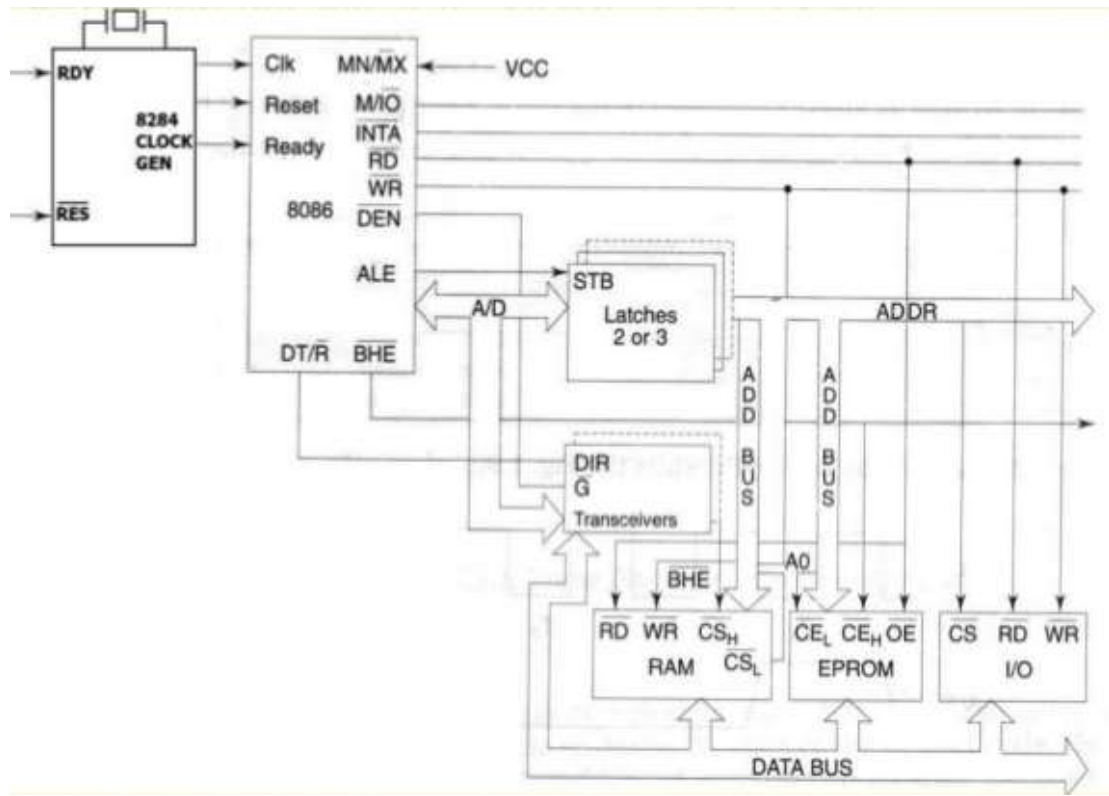
CSYNC: The clock synchronization pin is used whenever the EFI input provides synchronization in systems with multiple processors. When the internal crystal oscillator is used, this pin must be grounded.

GND: The ground pin is connects to ground.

Vcc: This power supply pin connects to + 5.0V with a tolerance of ± 10 percent

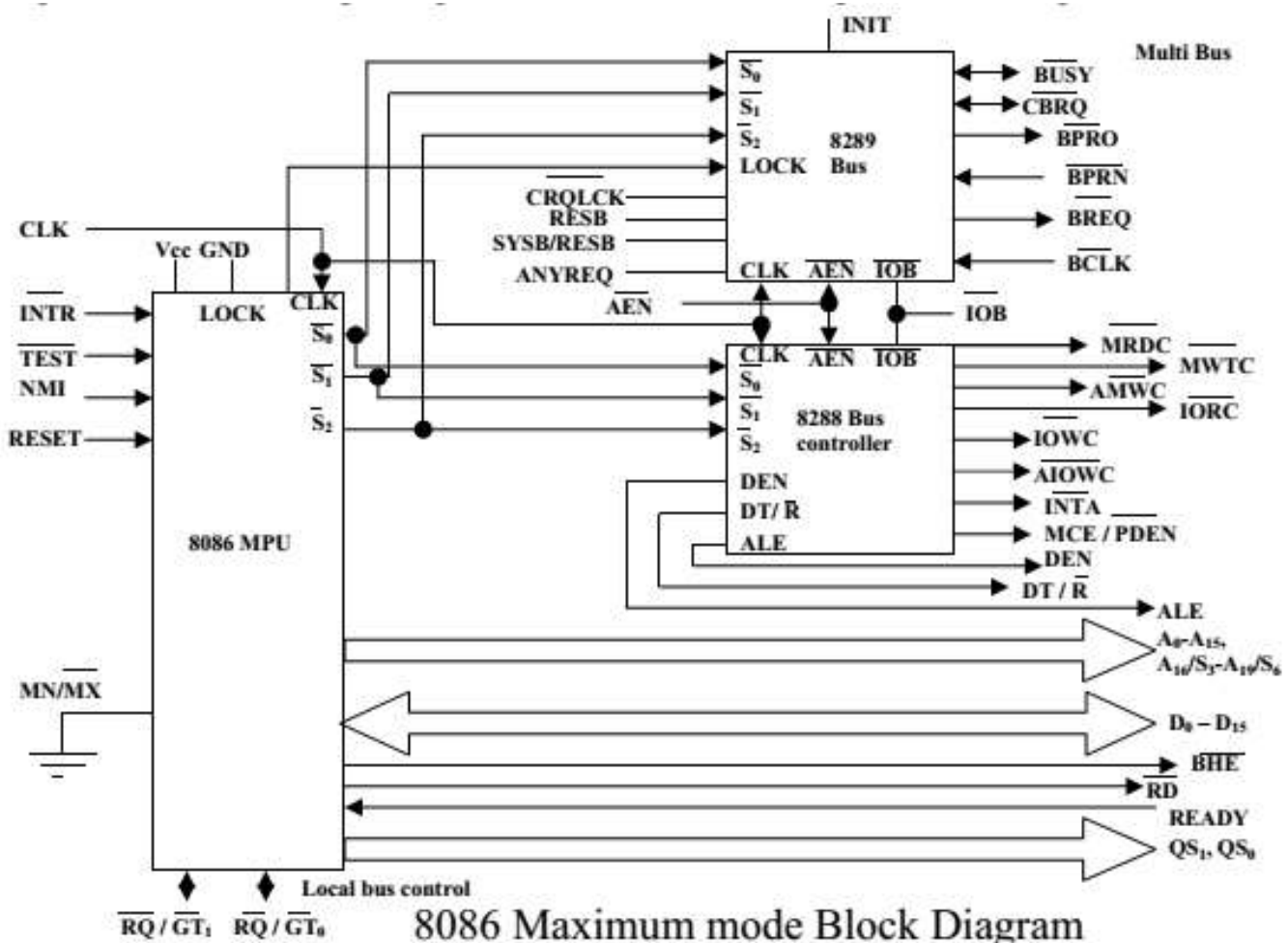
Q.8 Draw system configuration for 8086 microprocessor in minimum mode.

Answer:



Q.9 Draw system configuration for 8086 microprocessor in maximum mode.

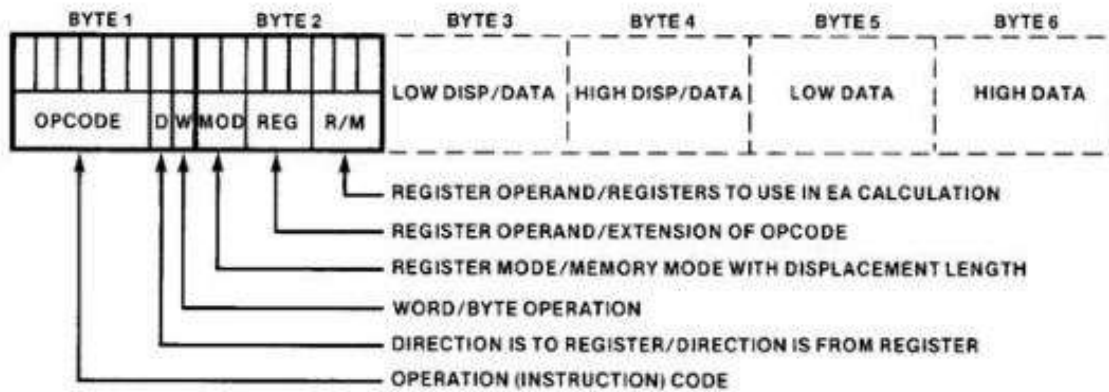
Answer:



Q.10 Draw instruction template format of 8086?

Answer:

The 8086 instruction sizes vary from one to six bytes. Depending on the type of coding, an instruction may have more than one hex code. Figure below shows the instruction format of 8086:



Q. 11 Explain with one example any five addressing modes of 8086?

Answer:

1. **Immediate addressing mode**-In this mode, the operand is specified in the instruction itself. Instructions are longer but the operands are easily identified. Example: `MOV CL, 12H` This instruction moves 12 immediately into CL register. $CL \leftarrow 12H$
2. **Register addressing mode**-In this mode, operands are specified using registers. This addressing mode is normally preferred because the instructions are compact and fastest executing of all instruction forms. Registers may be used as source operands, destination operands or both.
Example: `MOV AX, BX` This instruction copies the contents of BX register into AX register. $AX \leftarrow BX$
3. **Direct memory addressing mode**-In this mode, address of the operand is directly specified in the instruction. Here only the offset address is specified, the segment being indicated by the instruction.
Example: `MOV CL, [4321H]` This instruction moves data from location 4321H in the data segment into CL.
4. **Register based indirect addressing mode**-In this mode, the effective address of the memory may be taken directly from one of the base register or index register specified by instruction. If register is SI, DI and BX then DS is by default segment register. If BP is used, then SS is by default segment register.
Example: `MOV CX, [BX]`
This instruction moves a word from the address pointed by BX and BX + 1 in data segment into CL and CH respectively.
 $CL \leftarrow DS: [BX]$ and $CH \leftarrow DS: [BX + 1]$
Physical address can be calculated as $DS * 10H + BX$.
5. **Register relative addressing mode**-In this mode, the operand address is calculated using one of the base registers and an 8 bit or a 16 bit displacement.
Example:
`MOV CL, [BX + 04H]`
This instruction moves a byte from the address pointed by BX + 4 in data segment to CL.
 $CL \leftarrow DS: [BX + 04H]$
Physical address can be calculated as $DS * 10H + BX + 4H$.

Q.12 Write short note on assembler directives of 8086 microprocessor.

Answer:

Directives are the instructions to the assembler, and are not converted into the machine language. Following are some of directives used in 8086 programming:

1. ORG directive: This directive instructs the assembler to start the program in memory from the offset mentioned in the argument of ORG.
Example: ORG 100h will start the next instruction at an offset of 100h in the memory
2. SEGMENT directives: 8086 uses the directive SEGMENT(segment) to identify the beginning of a segment. The name of the segment should precede the keyword SEGMENT (segment). A segment thus declared will contain the code or the data depending on whether the declared segment is code segment or data segment. The words code or data or any other user defined names can be used to declare a particular segment. Its definition however will be clear in the code segment when you assign a segment name to the segment register. The name of the segment should be a unique without any blank spaces and may be up to 31 Characters long. Keywords cannot be used as name of the segment.
3. ENDS directive: 8086 make use of ENDS (ends) directive to mean the end of a segment. The keyword ENDS (ends) must be followed with the name of the segment to end.
4. END directive: The end END(end) directive tell the assembler to stop reading and assembling the program after the end directive. The statements after the end directive will be ignored by the assembler. It should be used as the last statement in a program.
5. PROC - The PROC directive is used to identify the start of a procedure. The term near or far is used to specify the type of the procedure.
6. ENDP directive: The ENDP (endp) directive is used to tell the assembler about the end of the procedure. When using the procedures in a program we need to call them and then at one point the procedure will end and return to the calling program. The example below clarifies it.

Sum_2_number procfar ;This identifies that the start of the procedure and instructs the assembler that it is far procedure.

--

--

Sum_2_number ENDP

7. ASSUME directive: ASSUME (assume) directive tells the assembler which segments are active. The format of the ASSUME (assume) directive is:

ASSUME CS:cs_name, DS: ds_name, SS: ss_name, ES: es_name\

Segment initialization

The next statement to Assume directive should be instructions. The Different segments can be initialized as follows: Suppose that a data segment has been created with ds_segment as name of segment then the next statement to assume directive will be:

MOV AX, ds_segment ; load offset of ds_segment in register AX

MOV DS, AX; copy the ds_segmet offset in DS, so DS points to 0000s

8. OFFSET directive: The offset directive will be used where the offset from starting of the segment is required in the program. The example of OFFSET is:

MOV SI, OFFSET string1; copies the offset of string1 in SI register

9. Data declaration directives: 8086 supports Different data types. The data types unlike in high level language which use keywords like INT, char, float etc., 8086 treats everything as a bytes and according have directives for declaring type for Single byte, two byte (a word), double word, quad words, ten bytes etc, These are-

DB - Defined Byte. DB declares a variable of type byte and reserves one location in memory for the variable of type byte.

Example: num1 DB 15h, Char1 db 'A'

numbersdb 100 dup(0); *Reserve an array of 50 words of memory and initialize All bytes with 00. Array is named as numbers.*

DD- Defined double Word, DW declares a variable of type byte and reserves one location in memory for the variable of type byte. Examples: num1 DW 1234h

ARR1 DW 1A3Bh, 3A4Bh, 5A6CH ; *this declares an array of 3 words and initialized with Specified values.*

ARR2 DW 50 DUP('0'); *Reserve an array of 50 words of memory and initialize All words with 0000. Array is named as ARR2.*

DQ-Define Quadword: This directive is used to define a variable of type quadword or to reserve storage location of type quadword in memory. For example

Quad_word DQ 1234123412341234H

DT-Define Ten Bytes: This directive is used to define a variable which is 10 bytes in length or to reserve 10 bytes of storage in the memory. For example

Ten_bytes DT 11223344556677889900

DUP directive: DUP directive is used to duplicate the basic data definition 'n' number of times; or saying it the other way is that DUP is used to declare array of Size n.

ARRY DB 10 dup (0), declares array of 10 byte.

EQU - This EQU directive is used to give a name to some value or to a symbol. Each time the assembler finds the name in the program, it will replace the name with the value or symbol you given to that name.

Example MULTIPLIER EQU 05H ; then subsequently where multiplication by 05h I required we can use this as follows

MUL AL, MULTIPLIER;

The advantage of using EQU in this manner is, if MULTIPLIER is used many no of times in a program and you want to Change the value, All you had to do is Change the EQU statement at beginning, it will changes the rest of All.

10. EVEN directive This directive is used to align the word boundary to next even address for faster access.

11. PUBLIC directive: This directive is used to instruct the assembler that a Specified name or label will be accessed from other modules.

Example: PUBLIC MULIPLIER, INTEREST_RATE

Q13. What is ISR? Write the sequence of steps to be followed when an interrupt occurs in 8086 microprocessor.

Solution

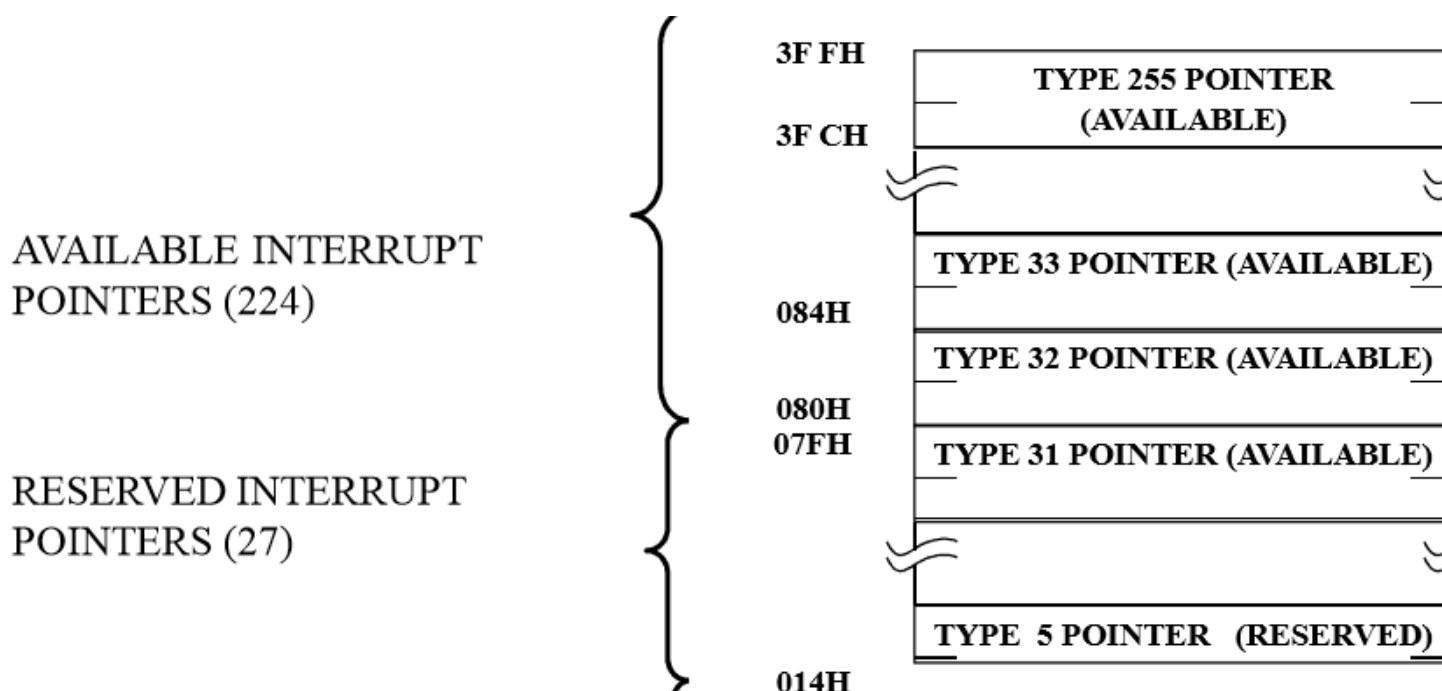
ISR: When a device interrupts, it actually wants the MP to give a service which is equivalent to asking the MP to call a subroutine. This subroutine is called ISR (Interrupt Service Routine).

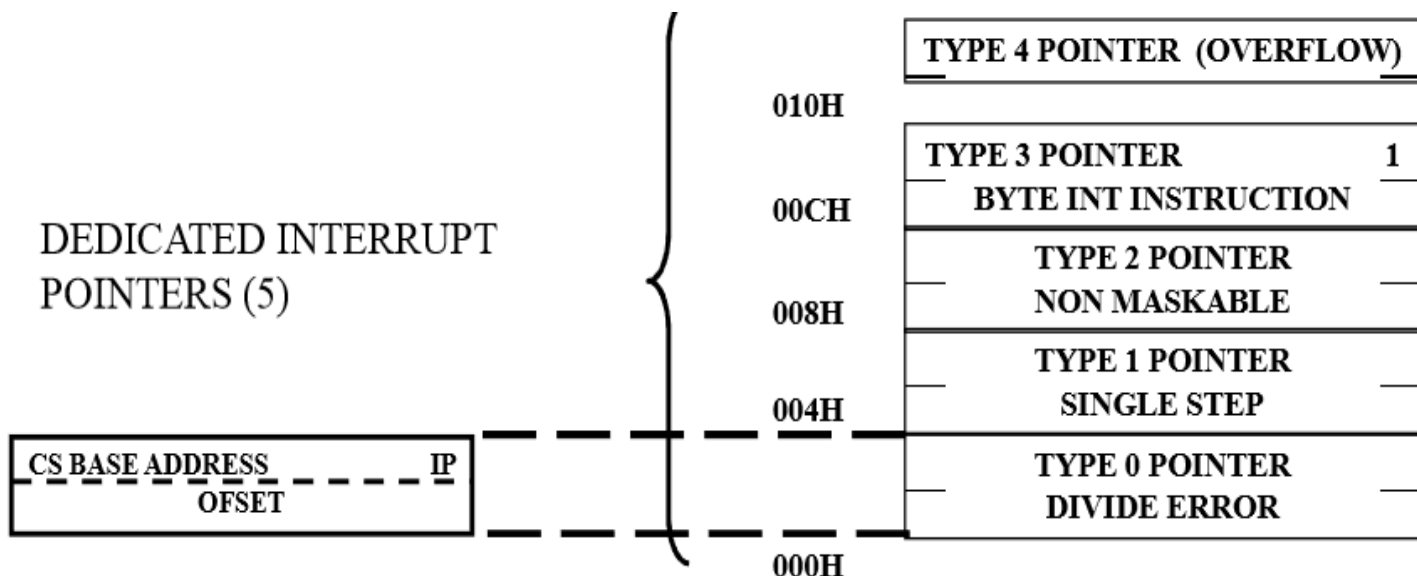
The sequence of steps to be followed when an interrupt occurs:

- 1. It decrements the stack pointer by 2 and pushes the flag register on the stack.**
- 2. It disables the 8086 INTR interrupt input by clearing the interrupt flag(IF) in the flag register.**
- 3. It resets the trap flag (TF) in the flag register.**
- 4. It decrements the stack pointer by 2 and pushes the current code segment register contents on the stack.**
- 5. It decrements the stack pointer by 2 and pushes the current instruction pointer contents on the stack.**
- 6. It does an indirect far jump to the start of the procedure the user has written to response to the interrupt.**
- 7. After it executes IRET instruction pops IP, CS and flags and resumes with microprocessor's main program.**

Q14. Show All types of interrupts in 8086 microprocessor with pointer table.

Solution





Q15. Compare RS 232, RS 422 and RS 485 Serial Communication Standards. (For knowledge only)

Answer:

RS-232 is the most common serial interface and used to ship as a standard component on most Windows-compatible desktop computers. It is used for many purposes, such as connecting a mouse, printer, or modem, as well as industrial instrumentation. It only allows for one transmitter and one receiver on each line. It also uses a Full-Duplex transmission method. Some RS-232 boards sold by National Instruments support baud rates up to 1 Mbit/s, but most devices are limited to 115.2 kbits/s.

RS-422 (EIA RS-422-A Standard) is the serial connection historically used on Apple Macintosh computers. RS-422 uses a differential electrical signal, as opposed to unbalanced signals referenced to ground with the RS-232. It provides a mechanism for transmitting data up to 10 Mbits/s. RS-422 sends each signal using two wires to increase the maximum baud rate and cable length. RS-422 is also specified for multi-drop applications where only one transmitter is connected to, and transmits on, a bus of up to 10 receivers. Differential transmission uses two lines each for transmit and receive signals which results in greater noise immunity and longer distances as compared to the RS-232. These advantages make RS-422 a better fit for industrial applications.

RS-485 is a superset of RS-422 and expands on the capabilities. RS-485 was made to address the multi-drop limitation of RS-422, allowing up to 32 devices to communicate through the same data line. Any of the slave devices on a RS-485 bus can communicate with any other 32 slave devices without going through a master device. Since RS-422 is a subset of RS-485, all RS-422 devices may be controlled by RS-485.

Specifications	RS-232	RS-422	RS-485
Mode of Operation	Single-Ended	Differential	Differential
Number of Drivers / Receivers on One Line	1 Driver 1 Receiver	1 Driver 10 Receivers	32 Drivers* 32 Receivers
Maximum Cable Length	50 ft (2500 pF)	4000 ft	4000 ft
Maximum Data Rate (at max cable length)	160 kbits/s (can be up to 1Mbit/s)	10 Mbit/s	10 Mbit/s

Q.16 Write an 8086 assembly language program for addition of two 8 bit numbers.

Answer:DATA SEGMENT

NUM1 DB 9H

NUM2 DB 7H

RESULT DB ?

ENDS

CODE SEGMENT

ASSUME DS:DATA CS:CODE

START:

MOV AX,DATA

MOV DS,AX

MOV AL,NUM1

ADD AL,NUM2

MOV RESULT,AL

ENDS

END START

Q17. Sum of BCD numbers

Answer:

DATA SEGMENT

NUM1 DB 05

NUM2 DB 06

RESULT DB ?

ENDS

CODESEGMENT

ASSUME DS:DATA CS:CODE

START:

MOV AX, DATA

MOV DS, AX

MOV AL, NUM1

ADD AL, NUM2

MOV RESULT, AL

MOV AH, 0

AAA

ENDS

END START

Q18. Sum of series of 10 numbers and store result in memory location total.

Solution: data segment

List db 12,34,56,78,98,01,13,78,18,36

Total dw ?

Data Ends

Code segment

Start: MOV AX, @data

MOV DS, AX

MOV AX, 0000H

MOV CX, 0AH ; counter

MOV BL, 00H ; to count carry

MOV SI, offset List

Back: ADD AL, [SI]

JC Label

Back1: INC SI

LOOP Back

MOV Total, AX

MOV Total+2, BL

Label: INC BL

JMP Back1

Code ends

End start

Q19. Write a program to multiply 2 numbers (16-bit data) for 8086.

Solution

data segment

Multiplier dw 1234H

Multiplicand dw 3456H

Product dw ?

Data ends

code segment

start: MOV AX, @data

MOV DS, AX

MOV AX, Multiplicand

MUL Multiplier

MOV Product, AX

MOV Product+2, DX

Code ends

End start

Q20. Write a program to find Largest No. in a block of data. Length of block is 0A. Store the maximum in location result.

data segment

List db 80, 81, 78, 65, 23, 45, 89, 90, 10, 99

Result db ?

Data ends

code segment

ASSUME CS: CODE, DS: DATA

Start:MOV AX, @data

MOV DS, AX

MOV SI, offset List

MOV AL, 00H

MOV CX, 0AH

Back: CMP AL, [SI]

JNC Ahead

MOV AL, [SI]

Ahead: INC SI

LOOP Back

MOV Result, AL

Code ends

End start

Q21. Write an 8086 assembly language program for sorting the numbers in ascending order.

Solution: DATA SEGMENT

LIST DB 12H,45H,28H,25H,A5H,F3H,14H,F4H,5AH,85H

COUNT EQU 10

DATA ENDS

CODE SEGMENT

ASSUME CS: CODE, DS: DATA

START: MOV AX, 2000H

MOV DS, AX ; INITIALIZE THE SEGMENT

```

MOV CX, COUNT

DEC CX

BACK2: MOV DX,COUNT

DEC DX

MOV SI,OFFSET LIST ;LOAD OFFSET OFVARIABLE LIST TO SI

BACK1: MOV AL, [SI]          ; LOAD THE FIRST NO. IN AL

INC SI                      ; POINTER TO NEXT NO.

CMP AL, [SI]                ; COMPARE TWO NOS.

JC NEXT    ; IF 1ST NO. LARGER THAN NEXT NO.JUMP TO REPEAT LOOP

MOV BH,[SI]

MOV [SI],AL

DEC SI

MOV [SI],BH

INC SI

NEXT: DEC DX

JNZ BACK1

LOOP BACK2

CODE ENDS

END START

```

Q.22 Write an 8086 assembly language program for sorting the numbers in descending order.

Solution: DATA SEGMENT

```

LIST DB 12H,45,28H,25H,A5H,F3H,14H,F4H,5AH,85H

COUNT EQU 10

DATA ENDS

```

CODE SEGMENT

ASSUME CS: CODE, DS: DATA

START: MOV AX, 2000H

```

MOV DS, AX                ; INITIALIZE THE SEGMENT

MOV CX, COUNT

DEC CX

BACK2: MOV DX,COUNT

DEC DX

MOV SI,OFFSET LIST ;LOAD OFFSET OFVARIABLE LIST TO SI

BACK1: MOV AL, [SI]        ; LOAD THE FIRST NO. IN AL

INC SI                    ; POINTER TO NEXT NO.

CMP AL, [SI]              ; COMPARE TWO NOS.

JNC NEXT                  ; IF 1ST NO.SMALLER THAN NEXT NO.JUMP TO REPEAT LOOP

MOV BH,[SI]

MOV [SI],AL

DEC SI

MOV [SI],BH

INC SI

NEXT: DEC DX

JNZ BACK1

LOOP BACK2

CODE ENDS

END START

```

Q23. Write difference between 8085 and 8086 microprocessor.

Difference Between 8085 & 8086 Microprocessor

8085 Microprocessor	8086 Microprocessor
<ul style="list-style-type: none">• Is an 8 Bit Microprocessor	<ul style="list-style-type: none">• Is a 16 Bit Microprocessor
<ul style="list-style-type: none">• Has 8 bit data bus	<ul style="list-style-type: none">• Has 16 bit data bus
<ul style="list-style-type: none">• Has 16 bit address line	<ul style="list-style-type: none">• Has 20 bit address line
<ul style="list-style-type: none">• Only 64kB of memory can be used (2^{16})	<ul style="list-style-type: none">• 1MB of memory can be used (2^{20})
<ul style="list-style-type: none">• Has 5 Flags (Carry, Parity, Sign, Zero, Auxillary Carry)	<ul style="list-style-type: none">• Has 9 Flags (Carry, Parity, Sign, Zero, Auxillary Carry, Direction, Trap, Interrupt, Overflow)
<ul style="list-style-type: none">• It is Accumulator based processor	<ul style="list-style-type: none">• It is General Purpose Register Based Processor
<ul style="list-style-type: none">• It has no MIN mode or MAX mode	<ul style="list-style-type: none">• It can operate in any one of MIN or MAX Mode
<ul style="list-style-type: none">• Does not support Pipelining	<ul style="list-style-type: none">• Supports Pipelining
<ul style="list-style-type: none">• Does not support Memory Segmentation	<ul style="list-style-type: none">• Supports Memory Segmentation
<ul style="list-style-type: none">• Has 6500 transistors	<ul style="list-style-type: none">• Has 29000 transistors

ASSEMBLY LANGUAGE PROGRAMMING EXAMPLES 8085:

Addition Programs

Example 1: Addition of two 8-bit numbers whose sum is 8-bits.

Explanation: This assembly language program adds two 8-bit numbers stored in two memory locations .The sum of the two numbers is 8-bits only.The necessary algorithm and flow charts are given below.

ALGORITHM:

Step1. : Initialize H-L pair with memory address XX00 (say: 9000).

Step2. : Clear accumulator.

Step3. : Add contents of memory location M to accumulator.

Step4. : Increment memory pointer (i.e. XX01).

Step5. : Add the contents of memory indicated by memory pointer to accumulator.

Step6. : Store the contents of accumulator in 9002.

Step7. : Halt

PROGRAM:

Address of the memory location	Hex code	Label	Mnemonics		Comments
			Op-code	Operand	

ALGORITHM:

Step1. : Initialize H-L pair with memory address X (say: 8500).

Step2. : Clear accumulator.

Step3. : Add contents of memory location M to accumulator.

Step4. : Increment memory pointer (i.e. 8501).

Step5. : Add the contents of memory indicated by memory pointer to accumulator.


Step6. : Check for Carry

Step 7 : Store the sum in 8502.

Step8 : Store the Carry in 8503 location


Step 9 : Halt

PROGRAM:

Address of the memory location	Hex code	Label	Mnemonics		Comments
			Op-code	Operand	
8000	21,00,85		LXI	H, 8500 H	Initialise memory pointer to point the first data location 9000.
8003	3E		MVI	A,00	Clear accumulator
8004	00				
8005	86		ADD	A, M	The first number is added to accumulator [A]  [A]+M
8006	0E		MVI	C,00	Initial value of Carry is 0
8007	00				
8008	23		INX	H	Increment the memory pointer to next location of the Data.
8009	86		ADD	A, M	The 2 nd number is added to contents of accumulator
800A	32		JNC	FWD	Is Carry exists ? No,go to the label FWD
800B	0E				
800C	80				
800D	0C		INR	C	Make carry =1
800E	32	FWD	STA	8502 H	The sum is stored in memory location 8502.
800F	02				
8010	85				
8011	79		MOV	A,C	

8012	32		STA	8503 H	Store the carry at 8503 location
------	----	--	-----	--------	-------------------------------------

Address of the memory location	Hex code	Label	Mnemonics		Comments
			Op-code	Operand	
8000	21, 00,85		LXI	H, 8500 H	Initialise memory pointer to point the first data location 9000.

8003	0E		MVI	C, 00	Clear accumulator
8004	00				
8005	7E		MOV	A, M	The first number is added to accumulator [A]  [A]+M
8006	23		INX	H	Increment the memory pointer to next location of the Data.
8007	86		ADD	A, M	The 2 nd number is added to contents of accumulator
8008	27		DAA		
8009	D2		JNC	FWD	Is Carry exists? No, go to the label FWD
	0D				
	80				
800C	0C		INR	C	Make carry =1
800D	32	FWD	STA	8502 H	The contents of accumulator are stored in memory location 8502.
800E	02				
800F	85				
8010	79		MOV	A, C	Carry is moved to accumulator
8011	32		STA	8503 H	A Carry is stored in the location 8503
8012	03				
8013	85				
8014	76		HLT		Stop the execution

Ex: Input: Ex : 8500 – 67 D
 8501 – 85 D

RESULT: 8502 – 52 D
 8503 – 01 (Carry)

Example 4: Addition of two 16-bit numbers whose sum is 16 bits or more

Explanation: First 16-bit number is stored in two consecutive locations (Ex 8500 &8501) because in each location we can store only one 8-bit number. Store the second 16-bit number in the next two consecutive locations (For Ex: 8502 &8503). Add the LSB of the first number to the LSB of the second number and the MSB of the first number to the MSB of the second number using the DAD instruction. Store the sum in the next two locations and the carry (if any) in the third location

ALGORITHM:

Step1: First 16 bit number is in locations 8500 & 8501 respectively

Step2: Second 16-bit number is in locations 8502 & 8503

Step3: Add the two 16-bit numbers using DAD Instruction.

Step4: Sum is stored in locations 8504 & 8505.

Step5: Carry (if any) is stored in the location 8506.

Step6: Halt

PROGRAM:

ADDRESS	HEX – CODE	LABEL	MNEMONIC		COMMENTS
			OPCODE	OPERAND	
8000	2A,00,85		LHLD	8500 H	First 16-bit number in H-L pair
8001	00				
8002	85				
8003	EB		XCHG		Exchange first number to D-E Pair
8004	2A		LHLD	8502 H	
8005	02				
8006	85				
8007	0E		MVI	00	MSB of the sum is initially 00
8008	00				
8009	19		DAD	D	Add two 16 –bit numbers
800A	D2		JNC	FWD	Is Carry? If yes go to the next line .Else go to the 800E LOCATION
800B	0E				
800C	80				
800D	0C		INR	C	Increment carry
800E	22	FWD	SHLD	8504 H	Store the LSB of the Sum in 8504 & MSB in 8505 locations
800F	04				
8010	85				

8011	79		MOV	A,C	MSBs of the sum is in
------	----	--	-----	-----	-----------------------

					Accumulator
8012	32		STA	8506 H	Store the MSB (Carry) of the result in 8506 location
8013	06				
8014	85				
8015	76		HLT		Stop execution

Ex: INPUT: 8500- 12 H LSB of the Ist Number **RESULT :** 8504 - 25H LSB of the Sum
 8501- 13 H MSB of the Ist Number 8505 – 25H MSB of the Sum
 8502 -13 H LSB of the IInd Number 8506 -- 00 Carry .
 8503 -12H MSB of the IInd number

Subtraction Programs:

Example 5: Subtraction of two 8-bit numbers without borrows.

Explanation: It's a simple program similar to addition of two 8- bit numbers, except that we use the instruction SUB instead of ADD. The first 8-bit number is stored in XX00 memory location and the second 8-bit number is stored in the XX01 location .Use the SUB instruction and store the result in the XX02 location.

ALGORITHM:

Step1. : Initialise H-L pair with the address of minuend.

Step2. : Move the minuend into accumulator

Step3. : Increment H-L pair

Step4. : Subtract the subtrahend in memory location M from the minuend.

Step5. : Store the result in XX02.

Step6. : Stop the execution

ADDRESS	HEX CODE	LABEL	MNEMONIC		COMMENTS
			OPCODE	OPERAND	
			E	D	

8000	21		LXI	H, 8500	Initialise H-L pair and get the First number in to 8500 location
8001	00				
8002	85				
8003	7E		MOV	A,M	$[A] \odot [M]$
8004	23		INX	H	$[M+1] \odot [M]$
8005	96		SUB	M	$A \odot [A] - [M]$
8006	23		INX	H	Next memory location
8007	77		MOV	M,A	Store the result in the location 8502
8008	76		HLT		Stop the execution

PROGRAM:

INPUT: Ex : 8500- 59H Result: 8502 – 29H
 8501- 30H

Example 6: Subtraction of two 8-bit Decimal numbers.

Explanation: In this program we can't use the DAA instruction after SUB or SBB instruction because it is decimal adjust after addition only. So, for decimal subtraction the number which is to be subtracted is converted to 10's complement and then DAA is applied.

ALGORITHM:

Step1. : Initialise H-L pair with the address of second number (XX01).

Step2. : Find its ten's complement

Step3. : Decrement the H-L pair for the first number (XX00)


Step4. : Add the first number to the 10's complement of second number.

Step5. : Store the result in XX02.

Step6. : Stop the execution

PROGRAM:

ADDRESS	HEX CODE	LAB EL	MNEMONIC		COMMENTS
			OPCODE	OPERAND	
8000	21		LXI	H,8500	Initialise H-L pair and get theSecond number in to 8501 location
8001	00				
8002	85				

8003	3E		MVI	A,99	[A]  99
8004	99				
8005	96		SUB	M	9's complement of second number
8006	3C		INR	A	10's complement of second number
8007	2B		DCX	H	Address of the first number

8008	86		ADD	M	Add first number to 10's complement of second number
8009	27		DAA		
800A	32		STA	8502	Store the result in the location 8502
800B	02				
800C	85				
800D	76		HLT		Stop the execution

Ex: Input: 8500 -76 D

Result: 8502 - 41 D

8501- 35 D

Example 6: Subtraction of two 16 –bit numbers.

Explanation: It is very similar to the addition of two 16-bit numers.Here we use SUB &SBB instructions to get the result .The first 16-bit number is stored in two consecutive locations and the second 16-bit number is stored in the next two consecutive locations.The lsbs are subtracted using SUB instruction and the MSBs aare subtracted using SBB instruction.The result is stored in different locations.

ALGORITHM:

Step1. : Store the first number in the locations 8500 & 8501.

Step2. : Store the second number in the locations 8502 &8503.

Step4. : Subtract the second number from the first number with borrow.

Step5. : Store the result in locations 8504 & 8505.

Step6. : Store the borrow in location 8506

Step 7: Stop the execution

PROGRAM:

Ex: INPUT : 8500- FF H LSB of the Ist Number

RESULT: 8504 - 11H LSB

ADDRESS	HEX CODE	LABEL	MNEMONIC		COMMENTS
			OPCODE	OPERAND	
8000	2A, 00, 85		LHLD	8500 H	First 16-bit number in H-L pair
8003	EB		XCHG		Exchange first number to D-E Pair
8004	2A		LHLD	8502 H	Get the second 16-bit number in H-L pair
8005	02				
8006	85				
8007	7B		MOV	A, E	Get the lower byte of the First number in to Accumulator
8008	95		SUB	L	Subtract the lower byte of the second number
8009	6F		MOV	L, A	Store the result in L- register
800A			MOV	A, D	Get higher byte of the first number
800A	9C		SBB	H	Subtract higher byte of second number with borrow
800B	67		MOV	H, A	
800C	22		SHLD	8504	Store the result in memory locations with LSB in 8504 & MSB in 8505
800D	04				
800E	85				
800F	76		HLT		Stop execution

8501 - FF H MSB of the Ist Number

8505 – 11 H MSB

8502 -EE H LSB of the IInd Number

8503 –EE H MSB of the IInd number

Multiplication Programs

Example 7: Multiplication of two 8-bit numbers. Product is 16-bits.

Explanation: The multiplication of two binary numbers is done by successive addition. When multiplicand is multiplied by 1 the product is equal to the multiplicand, but when it is multiplied by zero, the product is zero. So, each bit of the multiplier is taken one by one and checked whether it is 1 or 0 .If the bit of the multiplier is 1 the multiplicand is added to the product and the product is shifted to left by one bit. If the bit of the multiplier is 0 , the product is simply shifted left by one bit. This process is done for all the 8-bits of the multiplier.

ALGORITHM:

Step 1 : Initialise H-L pair with the address of multiplicand.(say 8500)

Step 2 : Exchange the H-L pair by D-E pair. so that multiplicand is in D-E pair.

Step 3 : Load the multiplier in Accumulator.

Step 4 : Shift the multiplier left by one bit.

Step 5 : If there is carry add multiplicand to product.

Step 6 : Decrement the count.

Step 7 : If count \neq 0; Go to step 4

Step 8 : Store the product i.e. result in memory location.

Step 9 : Stop the execution

PROGRAM:

ADDRESS	HEX - COD E	LABE L	MNEMONIC		COMMENTS
			OPCOD E	OPERAND	
8000	2A, 00,8 5		LHLD	H, 8500	Load the multiplicand in to H-L pair
8003	EB		XCHG		Exchange the multiplicand in to D-E pair
8004	3A		LDA	8502	Multiplier in Accumulator
8005	02				
8006	85				
8007	21		LXI	H.0000	Initial value in H-L pair is 00
8008	00				
8009	00				

800A	0E		MVI	C,08	Count =08
800B	08				
800C	29	LOOP	DAD	H	Shift the partial product left by one bit.

800D	17		RAL		Rotate multiplier left by one bit
800E	D2		JNC	FWD	Is Multiplier bit =1? No go to label FWD
800F	12				
8010	80				
8011	19		DAD	D	Product =Product +Multiplicand
8012	0D	FWD	DCR	C	COUNT=COUNT-1
8013	C2		JNZ	LOOP	
8014	0C				
8015	80				
8016	22		SHLD	8503	Store the result in the locations 8503 & 8504
8017	03				
8018	85				
8019	76		HLT		Stop the execution

INPUT :

Address	Data
8500	8AH – LSB of Multiplicand
8501	00 H – MSB of Multiplicand
8502	52 H - Multiplier

Result:

8503	34 H – LSB of Product
8504	2C H – MSB of Product

Example 7: Division of a 16- bit number by a 8-bit number.

Explanation: The division of a 16/8-bit number by a 8-bit number follows the successive subtraction method. The divisor is subtracted from the MSBs of the dividend .If a borrow occurs, the bit of the quotient is set to 1 else 0.For correct subtraction process the dividend is shifted left by one bit before each subtraction. The dividend and quotient are in a pair of register H-L.The vacancy arised due to shifting is occupied by the quotient .In the present example the dividend is a 16-bit number and the divisor is a 8-bit number. The dividend is in locations 8500 &8501.Similarly the divisor is in the location 8502.The quotient is stored at 8503 and the remainder is stored at 8504 locations.

ALGORTHM:

STEP1. : Initialise H-L pair with address of dividend.

STEP2. : Get the divisor from 8502 to register A & then to Reg.B

STEP3. : Make count C=08

STEP4. : Shift dividend and divisor left by one bit

STEP 5: Subtract divisor from dividend.

STEP6. : If carry = 1 : goto step 8 else step7.

STEP7. : Increment quotient register.

STEP8. : Decrement count in C

STEP9. : If count not equal to zero go to step 4

STEP10: Store the quotient in 8503

STEP11: Store the remainder in 8504

STEP12: Stop execution.






PROGRAM:

ADD R- ESS	HEX CODE	LABEL	MNEMONIC		COMMENTS
			OPCODE	OPERAND	
8000	21		LHLD	H, 8500	Initialize the H-L pair for dividend
8001	00				

8002	85				
8003	3A		LDA	8502 H	Load the divisor from location 8502 to accumulator
8004	02				
8005	85				
8006	47		MOV	B,A	Move Divisor to Reg.B from A
8007	0E		MVI	C,08	Count =08
8008	08				
8009	29	BACK	DAD	H	Shift dividend and quotient left by one bit

800A	7C		MOV	A,H	MSB of dividend in to accumulator
800B	90		SUB	B	Subtract divisor from MSB bits of divisor
800C	DA		JC	FWD	Is MSB part of dividend > divisor ? No, goto label FWD
800D	11				
800E	80				
800F	67		MOV	H,A	MSB of the dividend in Reg.H
8010	2C		INR	L	Increment quotient
8011	0D	FWD	DCR	C	Decrement count
8012	C2		JNZ	BACK	If count is not zero jump to 8009 location
8013	09				
8014	80				
8015	22		SHLD	8503H	Store quotient in 8503 and remainder in 8504 locations
8016	03				
8017	85				
8018	76		HLT		Stop execution

Ex: Input & Result

Address	Data
8500	64  LSB of Dividend
8501	00  MSB of Dividend
8502	07  Divisor
8503	0E  Quotient
8504	02  Remainder

Largest & Smallest numbers in an Array

Example 8: To find the largest number in a data array

Explanation: To find the largest number in a data array of N numbers (say) first the count is placed in memory location (8500H) and the data are stored in consecutive locations.(8501....onwards).The first number is copied to Accumulator and it is compared with the second number in the memory location. The larger of the two is stored in Accumulator. Now the third number in the memory location is again compared with the accumulator. And the largest number is kept in the accumulator. Using the count, this process is completed , until all the numbers are compared .Finally the accumulator stores the smallest number and this number is stored in the memory location.85XX.

ALGORITHM:

Step1: Store the count in the Memory location pointed by H-L register.

Step2: Move the 1 st number of the data array in to accumulator

Step3: Compare this with the second number in Memory location.

Step4: The larger in the two is placed in Accumulator

Step5: The number in Accumulator is compared with the next number in memory .

Step 6: The larger number is stored in Accumulator.

Step 7; The process is repeated until the count is zero.

Step 8: Final result is stored in memory location.

Step 9: Stop the execution

PROGRAM

ADDRESS	HEX – CODE	LABEL	MNEMONIC		COMMENTS
			OPCODE	OPERAND	
8000	21,00,85		LXI	H, 8500	INITIALISE H-L PAIR
8003	7E		MOV	C,M	Count in the C register
8004	23		INX	H	First number in H-L pair
8005	4E		MOV	A,M	Move first number in to Accumulator

8006	0D		DCR	C	Decrement the count
8007	91	LOOP1	INX	H	Get the next number
8008	BE		CMP	M	Compare the next number with previous number

8009	D2		JNC	LOOP2	Is next number >previous maximum?No,go to the loop2
800A	0D				
800B	80				
800C	7E		MOV	A,M	If,yes move the large number in to Accumulator
800D	0D	LOOP2	DCR	C	Decrement the count
800E	C2		JNZ	LOOP1	If count not equal to zero,repeat
800F	07				
8011	80				
8012	78				
8013	32		STA	85XX	Store the largest number in the location 85XX
8014	XX				
8015	85				
8016	76		HLT		Stop the execution

Ex : Input : 8500- N(Say N=7)

Result : 8508 - 7F

8501-05

8502-0A

8503-08

8504-14

8505 -7F

8506-25

8507-2D

Example 9 : To find the smallest number in a data array.

Explanation: To find the smallest number in a data array of N numbers (say) first the count is placed in memory location (8500H) and the data are stored in consecutive locations.(8501....onwards).The first number is copied to Accumulator and it is compared with the second number in the memory location.The smaller of the two is stored in Accumulator.Now the third number in the memory location is again compared with the accumulator.and the smallest number is kept in the accumulator.Using the count,this process is completed until all the numbers are compared .Finally the accumulator stores the smallest number and this number is stored in the memory location.85XX.

ALGORITHM :

Step1: Store the count in the Memory location pointed by H-L register.

Step2: Move the 1st number of the data array in to accumulator

Step3: Compare this with the second number in Memory location.

Step4: The smaller in the two is placed in Accumulator

Step5: The number in Accumulator is compared with the next number in memory .

Step 6: The smaller number is stored in Accumulator.

Step 7; The process is repeated until the count is zero.

Step 8: Final result is stored in memory location.

Step 9: Stop the execution

PROGRAM

ADD R- ESS	HEX – CODE	LABEL	MNEMONIC		COMMENTS
			OPCODE	OPERAND	
8000	21		LXI	H, 8500	Initialise the H-L pair.
8001	00				
8002	85				
8003	7E		MOV	C,M	Count in the C register
8004	23		INX	H	First number in H-L pair
8005	4E		MOV	A,M	Move first number in to Accumulator
8006	0D		DCR	C	Decrement the count
8007	91	LOOP1	INX	H	Get the next number

8008	BE		CMP	M	Compare the next number with previous number
8009	D2		JC	LOOP2	Is next number <previous smallest ?If yes go to the loop2

800A	0D				
800B	80				
800C	7E		MOV	A,M	No,move the smaller number in to Accumulator
800D	0D	LOOP2	DCR	C	Decrement the count
800E	C2		JNZ	LOOP1	If count not equal to zero,repeat
800F	07				
8011	80				
8012	78				
8013	32		STA	85XX	Store the smallest number in the location 85XX
8014	XX				
8015	85				
8016	76		HLT		Stop the execution

Ex: Input : 8500 - N((Say N=7)

Result : 8508 – 04

8501-09

8502-0A

8503-08

8504-14

8505 -7F

8506-04

8507-2D