Introduction

Intel introduced its first 4-bit microprocessor 4004 in 1971 and 8-bit microprocessor 8008 in 1972. These microprocessors could not survive as general-purpose microprocessors due to their design and performance limitations. Launching of the first general purpose 8-bit microprocessor 8080 in 1974 by Intel is considered to be the first major stepping stone towards the development of advanced microprocessors. The microprocessor 8085 followed 8080, with a few more features added to its architecture, which resulted in a functionally complete microprocessor. The main limitations of the 8-bit microprocessors were their low speed of execution, low memory addressing capability, limited number of general purpose registers and a less powerful instruction set. All these limitations of the 8-bit microprocessors tempted the designers to go for more powerful processors in terms of advanced architecture, more processing capability, larger memory addressing capability and a more powerful instruction set. The 8086 was a result of such developmental design efforts.

In the family of 16-bit microprocessors, Intel's 8086 was the first one launched in 1978. The introduction of the 16-bit processor was a result of the increasing demand for more and more powerful and high speed computational resources. 8086 microprocessors has a much more powerful instruction set along with the architectural developments which imparted substantial programming flexibility and improvement in speed over the 8-bit microprocessors.

The peripheral chips designed earlier for 8085 were compatible with microprocessor 8086 with slight or no modifications. Though there is a considerable difference between the memory addressing techniques of 8085 and 8086, the memory interfacing technique is similar, but includes the use of a few additional signals. The clock requirements are also different as compared to 8085, but the overall minimal system organization of 8086 is similar to that of a general 8-bit microprocessor. In this chapter, the architectures of 8086 and 8088 are discussed in adequate details along with the interfacing of the supporting chips with them to form a minimum system. The system organization is also discussed in significant details for both the operating modes of 8086 and 8088, along with necessary timing diagrams.

Internal Architecture

The architecture of 8086 provides a number of improvements over 8085 architecture. It supports a 16-bit ALU, a set of 16-bit registers and provides segmented memory addressing capability, a

rich instruction set, powerful interrupt structure, fetched instruction queue for overlapped fetching and execution etc. The internal block diagram, shown in Figure 2.1, describes the overall organization of different units inside the chip.

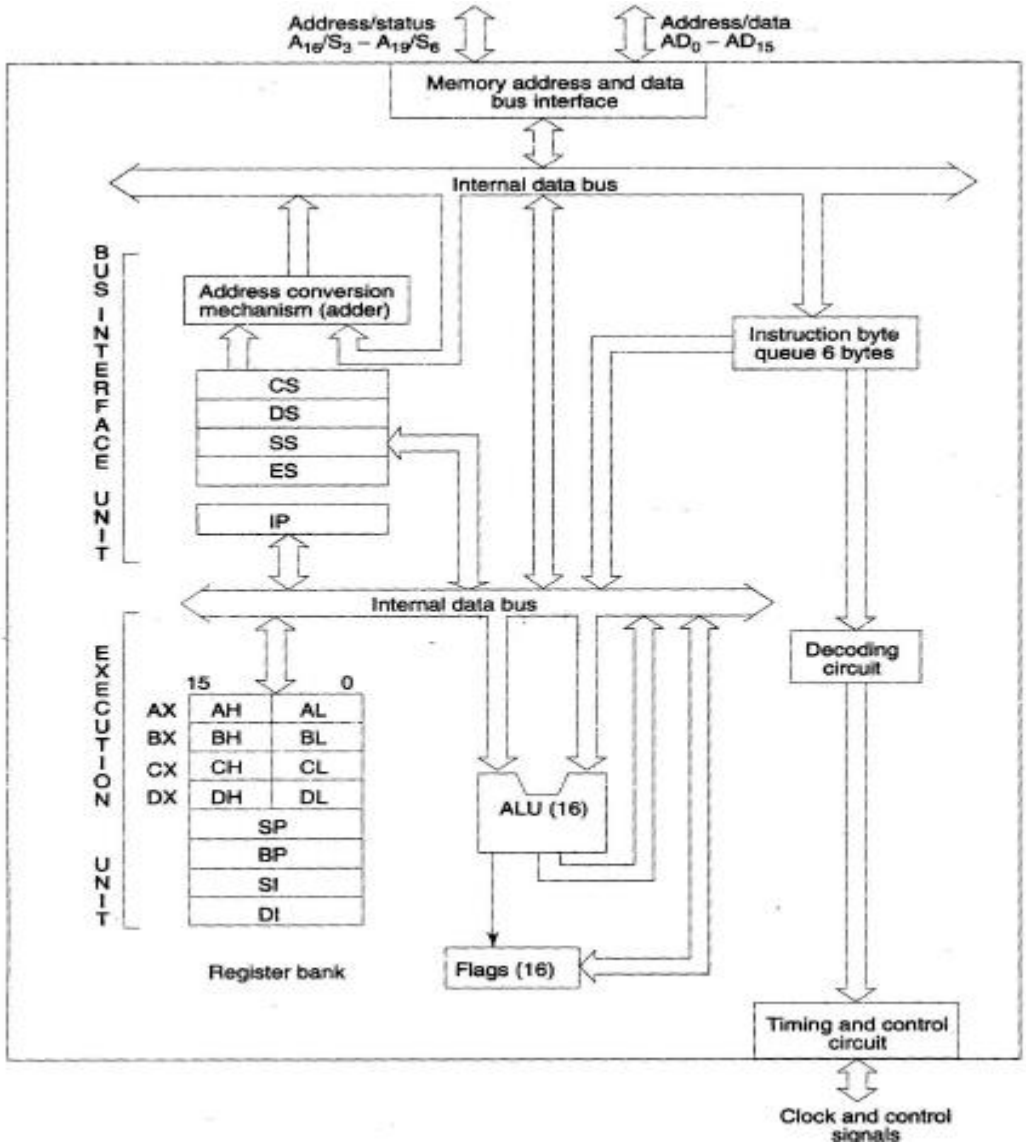


Figure 2.1: 8086 Architecture

The complete architecture of 8086 can be divided into two parts (a) Bus Interface Unit (BIU) and (b)Execution Unit (EU). The bus interface unit contains the circuit for physical address calculations and a predecoding instruction byte queue (6 bytes long). The bus interface unit makes the system bus signals available for external interfacing of the devices. In other words, this unit is responsible for establishing communications with external devices and peripherals including memory via the bus. As already stated, the 8086 addresses a segmented memory. The

complete physical address which is 20-bits long is generated using segment and offset registers, each 16-bits long.

For generating a physical address from contents of these two registers, the content of a segment register also called as segment address is shifted left bit-wise four times and to this result, content of an offset register also called as offset address is added, to produce a 20-bit physical address. For example, if the segment address is 1005H and the offset is 5555H, then the physical address is calculated as below.

```
Segment address → 1005H
Offset address  → 5555H
Segment address → 1005H →          0001  0000  0000  0101
Shifted by 4 bit positions →  0001  0000  0000  0101  0000
                                    +
Offset address                →     0101  0101  0101  0101
Physical address              → 0001  0101  0101  1010  0101
                                  1     5     5     A     5
```

Thus the segment addressed by the segment value 1005H can have offset values from 0000H to FFFFH within it, i.e. maximum 64K locations may be accommodated in the segment. Thus the segment register indicates the base address of a particular segment , while the offset indicates the distance of the required memory location in the segment from the base address. Since the offset is a 16-bit number, each segment can have a maximum of 64K locations. The bus interface unit has a separate adder to perform this procedure for obtaining a physical address while addressing memory. The segment address value is to be taken from an appropriate segment register depending upon whether code, data or stack are to be accessed, while the offset may be the content of IP, BX, SI, DI, SP or an immediate 16-bit value, depending upon the addressing mode.

In case of 8085, once the opcode is fetched and decoded, the external bus remains free for some time, while the processor internally executes the instruction. This time slot is utilized in 8086 to achieve the overlapped fetch and execution cycles. While the fetched instruction is executed internally, the external bus is used to fetch the machine code of the next instruction and arrange it in a queue called as predecoded instruction byte queue. It is a 6 bytes long, first-in first-out structure. The instructions from the queue are taken for decoding sequentially. Once a byte is decoded, the queue is rearranged by pushing it out and the queue status is checked for the possibility of the next opcode fetch cycle. While the opcode is fetched by the bus

interface unit (BIU), the execution unit (EU) executes the previously decoded instruction concurrently. The BIU along with the execution unit (EU) thus forms a pipeline. The bus interface unit thus manages the complete interface of execution unit with memory and I/O devices, of course, under the control of the timing and control unit.

The execution unit contains the register set of 8086 except segment registers and IP. It has a 16-bit ALU, able to perform arithmetic and logic operations. The 16-bit flag register reflects the results of execution by the ALU. The decoding unit decodes the opcode bytes issued from the instruction byte queue. The timing and control unit derives the necessary control signals to execute the instruction opcode received from the queue, depending upon the information made available by the decoding circuit. The execution unit may pass the results to the bus interface unit for storing them in memory.

Memory Segmentation

The memory in an 8086/8088 based system is organized as segmented memory. In this scheme, the complete physically available memory may be divided into a number of logical segments. Each segment is 64K bytes in size and is addressed by one of the segment registers. The 16-bit contents of the segment register actually point to the starting location of a particular segment. To address a specific memory location within a segment, we need an offset address. The offset address is also 16-bit long so that the maximum offset value can be FFFFH, and the maximum size of any segment is thus 64K locations.

To emphasize this segmented memory concept, we will consider an example of a housing colony containing say, 100 houses. The simplest method of numbering the houses will be just to assign the numbers from 1 to 100 to each house sequentially. Suppose, now, if one wants to find out house number 67, then he will start from house number 1 and go on till he finds the house, numbered 67. Consider another case where the 100 houses are arranged in the 10 x 10 (rows x columns) pattern. In this case, to find out house number 67, one will directly go to the 6th row and then to the 7th column. In the second scheme, the efforts required for finding the same house will be too less. This second scheme in our example is analogous to the segmented memory scheme, where the addresses are specified in terms of segment addresses analogous to rows and offset addresses analogous to columns.

The CPU 8086 is able to address 1Mbytes of physical memory. The complete 1Mbytes memory can be divided into 16 segments, each of 64Kbytes size. The addresses of the segments may be

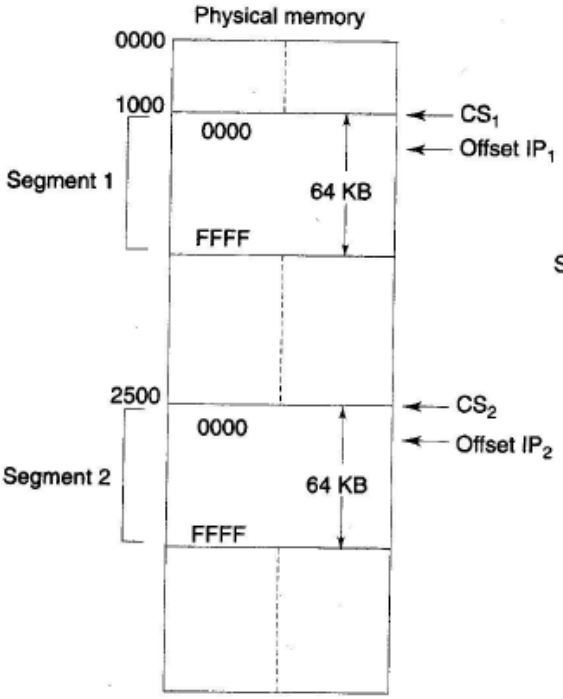
assigned as 0000H to F000H respectively. The offset address values are from 0000H to FFFFH so that the physical addresses range from 00000H to FFFFFH. In the above said case, the segments are called non-overlapping segments.

The non-overlapping segments are shown in Figure 2.2(a). In some cases, however, the segments may be overlapping. Suppose a segment starts at a particular address and its maximum size can be 64Kbytes. But, if another segment starts before this 64Kbytes locations of the first segment, the two segments are said to be overlapping segments. The area of memory from the start of the second segment to the possible end of the first segment is called as overlapped segment area. Figure 2.2(b) explains the phenomenon more clearly.

The locations lying in the overlapped area may be addressed by the same physical address generated from two different sets of segment and offset addresses. The main advantages of the segmented memory scheme are as follows:

1. Allows the memory capacity to be 1Mbytes although the actual addresses to be handled are of 16-bit size.
2. Allows the placing of code, data and stack portions of the same program in different parts (segments) of memory, for data and code protection.
3. Permits a program and/or its data to be put into different areas of memory each time program is executed, i.e. provision for relocation may be done.

In the Overlapped Area Locations Physical Address = CS + IF = CS + IF + indicates the procedure of physical address formation.



Fieure 2.2(a): Non-overlappoine Segments

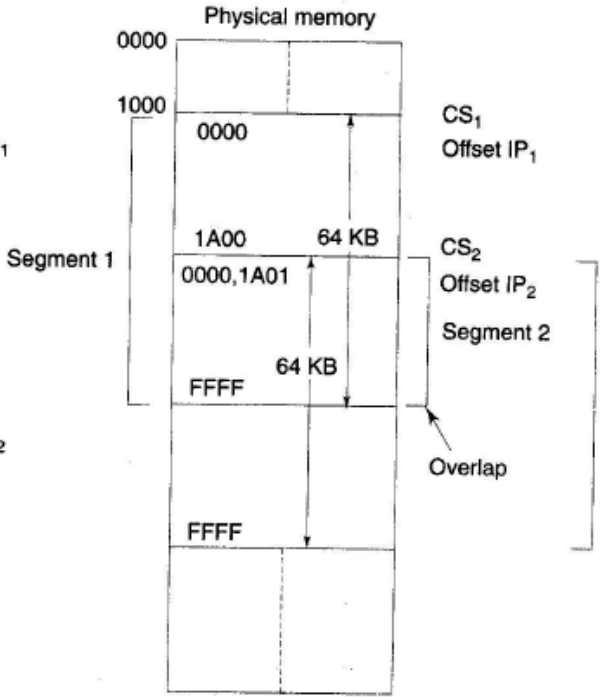


Figure 2.2(b): Overlapping Segments

Flag Register

8086 has a 16-bit flag register which is divided into two parts, viz. (a) condition code or status flags and (b) machine control flags. The condition code flag register is the lower byte of the 16-bit flag register along with the overflow flag. The condition code flag register is identical to 8085 flag register, with an additional overflow flag, which is not present in 8085. This part of the flag register of 8086 reflects the results of the operations performed by ALU. The control flag register is the higher byte of the flag register of 8086. It contains three flags, viz. direction flag (D), interrupt flag (1) and trap flag (T). The complete bit configuration of 8086 flag register is shown in Figure 2.3.

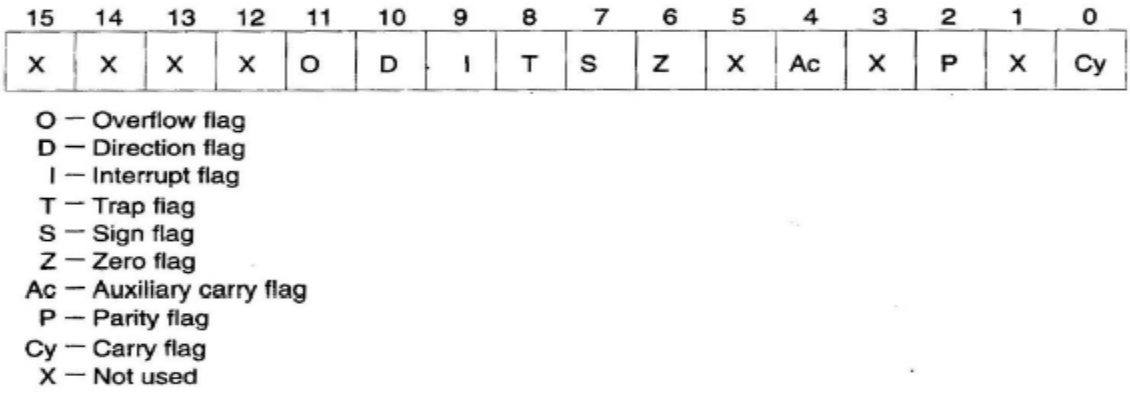


Figure 2.3: Flag Register of 8086

The description of each flag bit is as follows:

S-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.

Z-Zero Flag: This flag is set, if the result of the computation or comparison performed by the previous instruction/instructions is zero.

P-Parity Flag: This flag is set to 1, if the lower byte of the result contains even number of 1's.

C-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. For example, when two numbers are added, a carry may be generated out of the most significant bit position. The carry flag, in this case, will be set to '1'. In case, no carry is generated, it will be '0'. Some other instructions also affect or use this flag and will be discussed later in this text.

T-Trap Flag: If this flag is set, the processor enters the single step execution mode. In other words, a trap interrupt is generated after execution of each instruction. The processor executes the current instruction and the control is transferred to the Trap interrupt service routine.

1-lnterrupt Flag: If this flag is set, the maskable interrupts are recognised 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. autoincrementing mode. Otherwise, the string is processed from the highest address towards the lowest address, i.e. autodecrementing mode.

Ac-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.

O-Overflow Flag: This flag is set, if an overflow occurs, i.e. if the result of a signed operation is large enough to be accommodated in a destination register. For example, in case of the addition of two signed numbers, if the result overflows into the sign bit, i.e. the result is of more than 7-bits in size in case of 8-bit signed operations and more than 15-bits in size in case of 16-bit signed operations, then the overflow flag will be set.

Pin Descriptions of 8086

The microprocessor 8086 is a 16-bit CPU available in three clock rates, i.e. 5, 8 and 10 MHz, packaged in a 40 pin CERDIP or plastic package. The 8086 operates in single processor or multiprocessor configurations to achieve high performance. The pin configuration is shown in

Figure 2.4. 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 categorised 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 for minimum mode and the third are the signals having special functions for maximum mode.

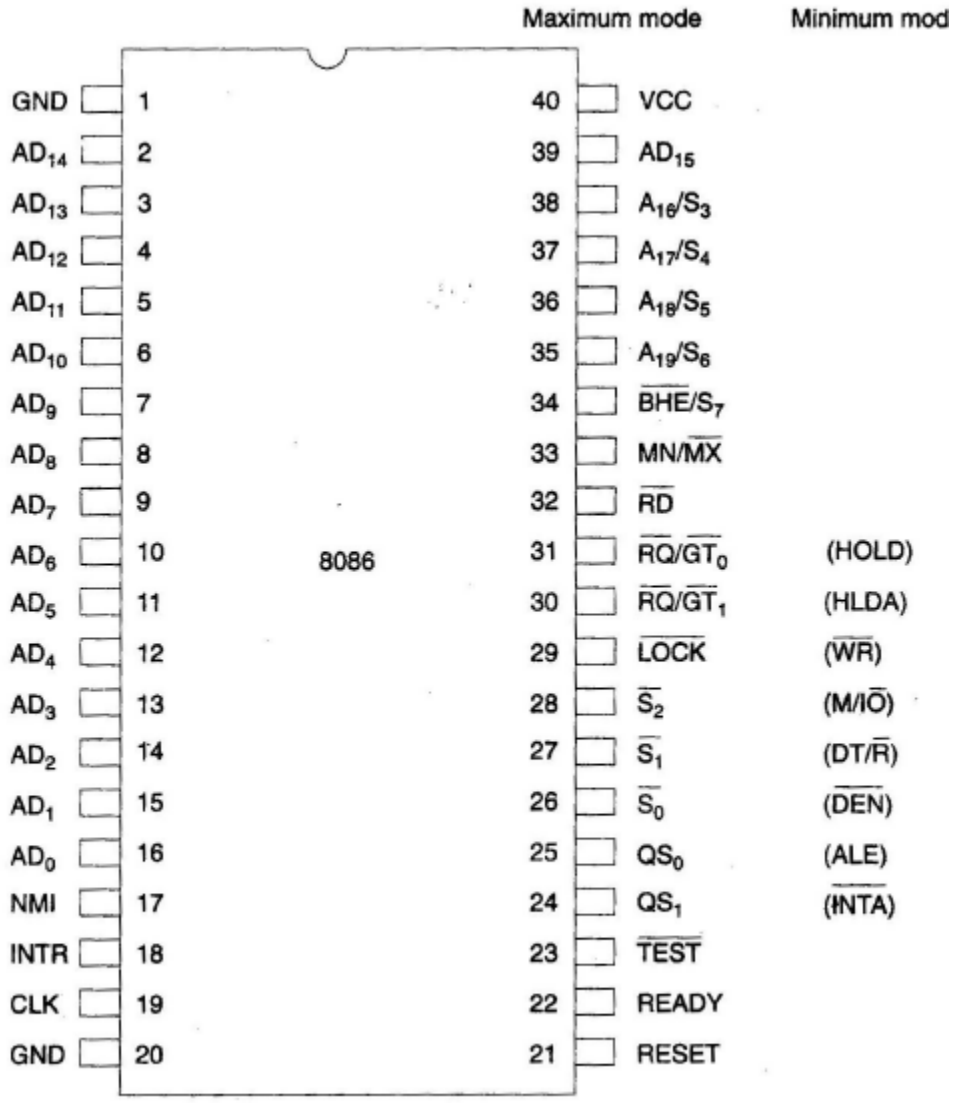


Figure 2.4: Pin Configuration of 8086

The following signal descriptions are common for both the minimum and maximum mode

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 a wait state. These lines are active high and float to a tristate 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 for memory operations. During l/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 S3 combinedly indicate which segment register is presently being used for memory accesses as shown in Table 2.1 These lines float to tri-state off (tristated) 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.

Table 2.1: Bus High Enable/status

S_4	S_3	Indications
0	0	Alternate Data
0	1	Stack
1	0	Code or none
1	1	Data

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 2.2. 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, whenever 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 tristated during 'hold'. It is low during T1 for the first pulse of the interrupt acknowledges cycle.

Table 2.2

BHE	A_0	Indication
0	0	Whole word
0	1	Upper byte from or to odd address.
1	0	Lover 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 tristated 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 FFFFOH.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 synchronised.

CLK-Clock Input: The clock input provides the basic timing for processor operation and bus control activity. Its an asymmetric square wave with 33% duty cycle. The range of frequency for different 8086 versions is from 5MHz to l0MHz.

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/I/O -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 tristated during local bus "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 duringNT2, 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 transreceivers (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 transreceivers (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 tristated 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 T2 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 2.3.

Table 2.3

S_2	S_1	S_0	Indication
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O Port
0	1	0	Wrtie 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 2.4.

Table 2.4

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.
Figure 2.5 explains the queue operation.

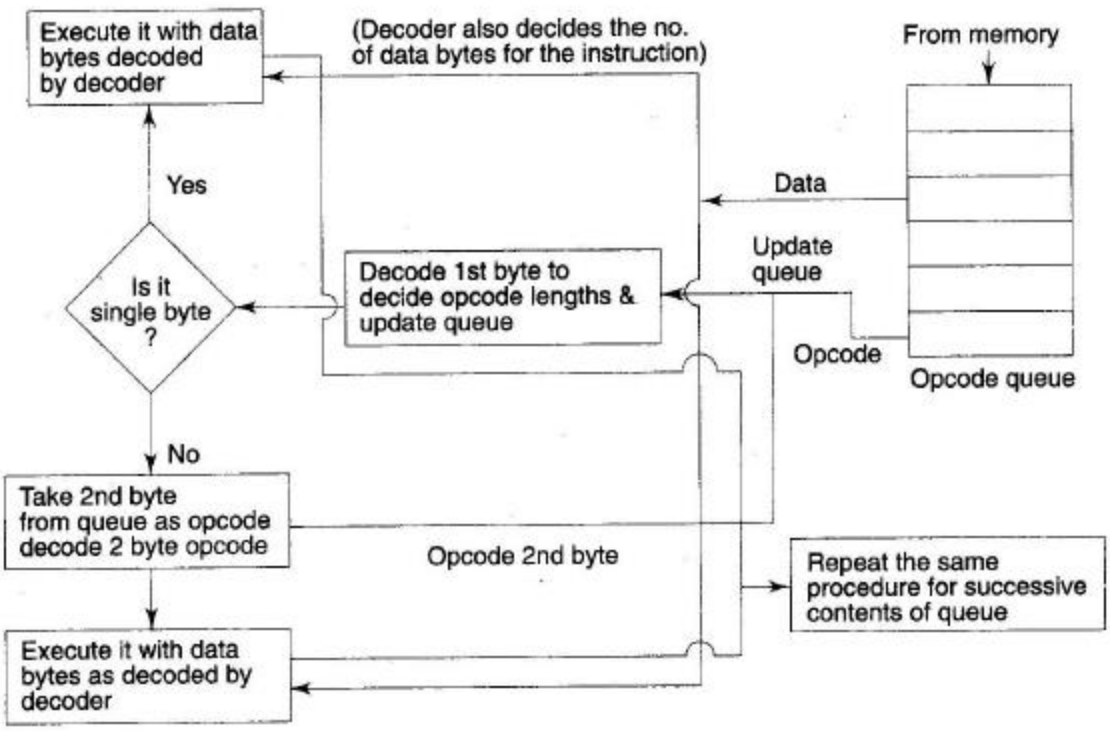


Figure 2.5: The Queue Operation

RQ/CT0, RQ/G1-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/GT() 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 74 (current) or T, (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 the 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. Until now, we have described the architecture and pin configuration of 8086. In the next section, we will study some operational features of 8086 based systems.