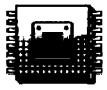
1 The Processors: 8086/8088—Architectures, Pin Diagrams and Timing Diagrams



INTRODUCTION

Intel introduced its first 4-bit microprocessor 4004 in 1971 and its 8-bit microprocessor 8008 in 1972. These microprocessors could not survive as general purpose microprocessors due to their design and performance limitations. The launch 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 added features to its architecture, which resulted in a functionally complete microprocessor. The main limitations of the 8-bit microprocessors were their low speed, low memory addressing capability, limited number of general purpose registers and a less powerful instruction set. All these limitations of the 8-bit microprocessors pushed the designers to build 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 to be launched in 1978. The introduction of the 16-bit processor was a result of the increasing demand for more powerful and high speed computational resources. The 8086 microprocessor has a much more powerful instruction set along with the architectural developments which imparts 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 organisation 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 organisation is also discussed in significant details for both the operating modes of 8086 and 8088, along with necessary timing diagrams.

1.1 REGISTER ORGANISATION OF 8086

8086 has a powerful set of registers known as *general purpose* and *special purpose registers*. All of them are 16-bit registers. The general purpose registers, can be used as either 8-bit registers or 16-bit registers. They

may be either used for holding data, variables and intermediate results temporarily or for other purposes like a 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. We will categorize the register set into four groups, as follows:

1.1.1 General Data Registers

Figure 1.1 shows the register organisation of 8086. The registers AX,BX,CX and DX are the general purpose 16-bit registers. AX is used as 16-bit *accumulator*, with the lower 8-bits of AX designated as AL and higher 8-bits as AH. AL can be used as an 8-bit accumulator for 8-bit operations. This is the most important general purpose register having multiple functions, which will be discussed later.

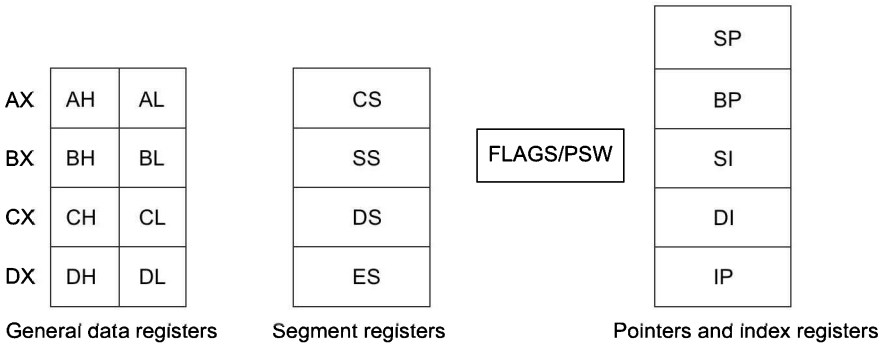


Fig. 1.1 *Register organisation of 8086*

Usually the letters L and H specify the lower and higher bytes of a particular register. For example, CH means the higher 8-bits of the CX register and CL means the lower 8-bits of the CX register. The letter X is used to specify the complete 16-bit register. The register CX is also used as a default counter in case of string and loop instructions. The register BX is used as an offset storage for forming physical addresses in case of certain addressing modes. DX register is a general purpose register which may be used as an implicit operand or destination in case of a few instructions. The detailed uses of these registers will be more clear when we discuss the addressing modes and the instruction set of 8086.

1.1.2 Segment Registers

Unlike 8085, the 8086 addresses a segmented memory. The complete 1 megabyte memory, which the 8086 addresses, is divided into 16 logical segments. Each segment thus contains 64 Kbytes of memory. There are four segment registers, viz. Code Segment Register (CS), Data Segment Register (DS), Extra Segment Register (ES) and Stack Segment Register (SS). The code segment register is used for addressing a memory location in the code segment of the memory, where the executable program is stored. Similarly, the data segment register points to the data segment of the memory, where the data is resided. The extra segment also refers to a segment which essentially is another data segment of the memory. Thus, the extra segment also contains data. The stack segment register is used for addressing stack segment of memory i.e. memory which is used to store stack data. The CPU uses the stack for temporarily storing important data, e.g. the contents of the CPU registers which will be required at a later stage. The stack grows down, i.e. the data is pushed onto the stack in the memory locations with decreasing addresses. When this information will be required by the CPU, they will be popped off from the stack. While addressing any location in the memory bank, the physical address is calculated from two parts, the first is *segment address* and the second is *offset*. The segment registers contain 16-bit segment base addresses, related to different segments. Any of the pointers and index registers or BX may contain the offset of the location to be addressed. The advantage of this scheme is that

instead of maintaining a 20-bit register for a physical address, the processor just maintains two 16-bit registers which are within the word length capacity of the machine. Thus the CS, DS, SS and ES segment registers, respectively, contain the segment addresses for the code, data, stack and extra segments of memory. It may be noted that all these segments are the logical segments. They may or may not be physically separated. In other words, a single segment may require more than one memory chip or more than one segment may be accommodated in a single memory chip.

1.1.3 Pointers and Index Registers

The pointers contain offset within the particular segments. The pointers IP, BP and SP usually contain offsets within the code (JP), and stack (BP & SP) segments. The *index registers* are used as general purpose registers as well as for offset storage in case of indexed, based indexed and relative based indexed addressing modes. The register SI is generally used to store the offset of source data in data segment while the register DI is used to store the offset of destination in data or extra segment. The index registers are particularly useful for string manipulations.

1.1.4 Flag Register

The 8086 *flag register* contents indicate the results of computations in the ALU. It also contains some flag bits to control the CPU operations. Details of the flag register are discussed later in this chapter.

1.2 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 Fig.1.2, describes the overall organization of different units inside the chip.

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's 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
```

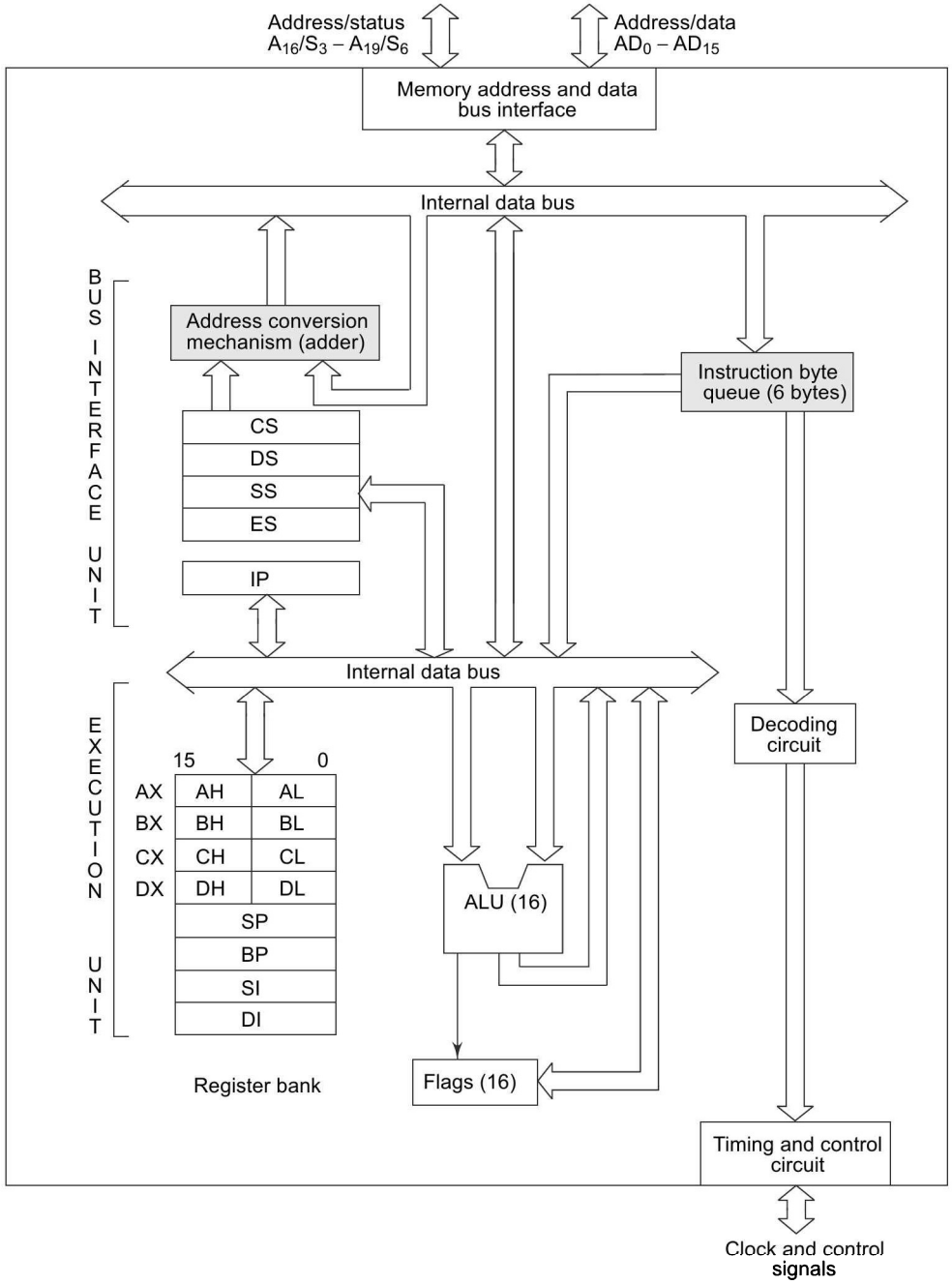


Fig. 1.2 *8086 Architecture*

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, BP 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 utilised 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 known 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.

1.2.1 Memory Segmentation

The memory in an 8086/8088 based system is organised 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. The physical address formation has been explained previously in Section 1.2.

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 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 10×10 (rows $\times$ 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 Fig. 1.3(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 an overlapped segment area. Figure 1.3(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 the program is executed, i.e. provision for relocation is done.

In the Overlapped Area Locations Physical Address = $CS_1 + IP_1 = CS_2 + IP_2$, where '+' indicates the procedure of physical address formation.

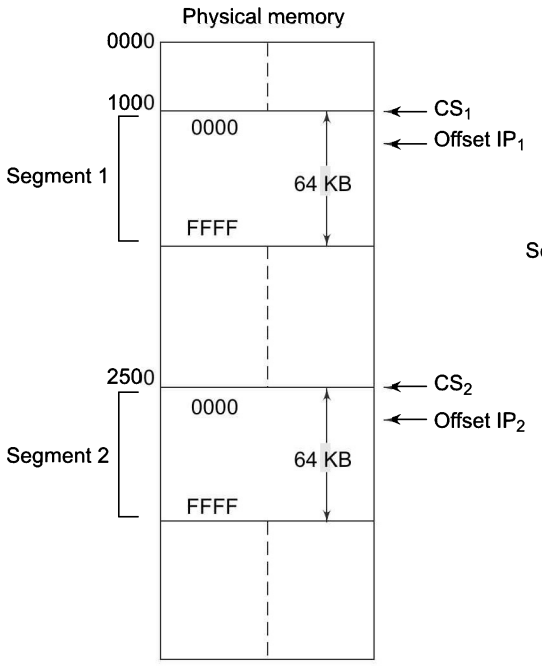


Fig. 1.3(a) *Non-overlapping Segments*

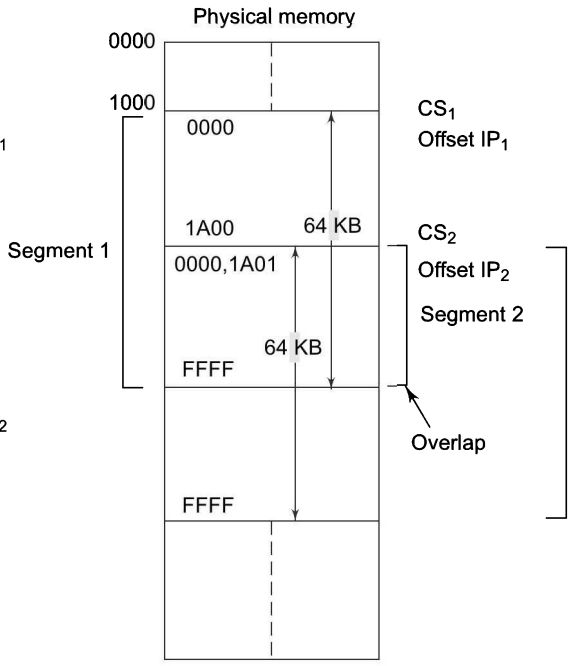


Fig. 1.3(b) *Overlapping Segments*

1.2.2 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*. This flag is identical to the 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* (I) and *trap flag* (T).

The complete bit configuration of 8086 flag register is shown in Fig. 1.4.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	O	D	I	T	S	Z	X	Ac	X	P	X	Cy

O — Overflow flag
D — Direction flag
I — Interrupt flag
T — Trap flag
S — Sign flag
Z — Zero flag
Ac — Auxiliary carry flag
P — Parity flag
Cy — Carry flag
X — Not used

Fig. 1.4 *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 1s.

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.

I-Interrupt 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*. We will describe string manipulations later in chapter 2 in more detail.

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.

1.3 SIGNAL 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 Fig. 1.5. 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.

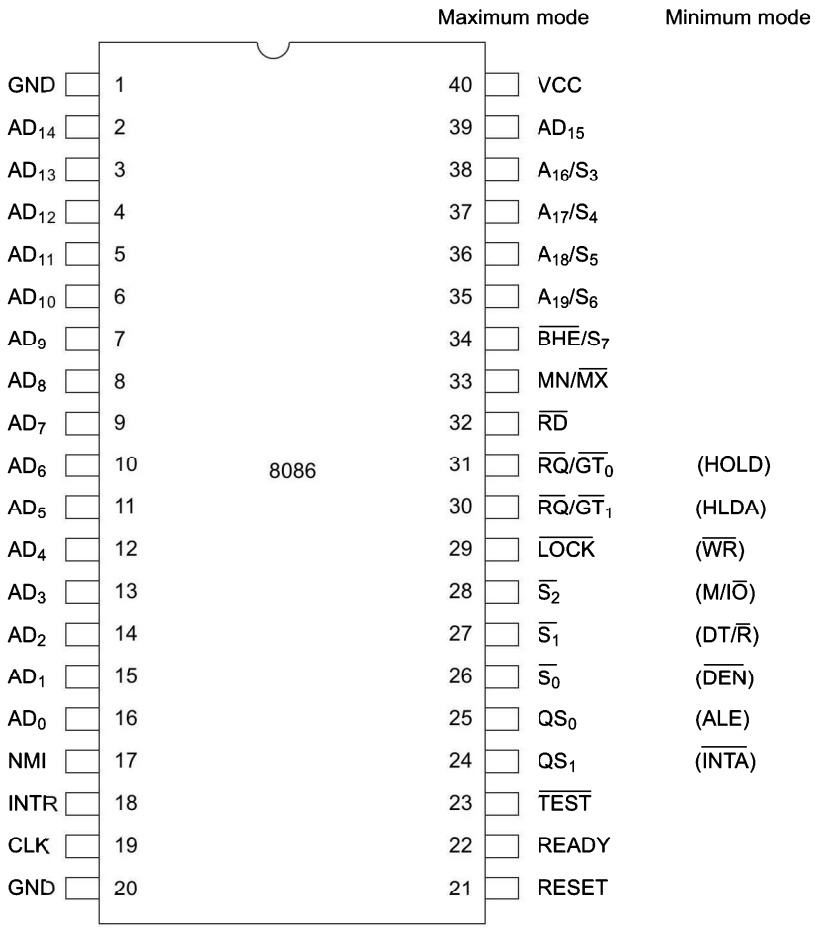


Fig. 1.5 *Pin Configuration of 8086*

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

AD_{15}-AD_0 These are the time multiplexed memory I/O address and data lines. Address remains on the lines during T_1 state, while the data is available on the data bus during T_2,T_3,T_w and T_4. Here T_1,T_2,T_3,T_4 and

T_w are the clock states of a machine cycle. T_w is a wait state. These lines are active high and float to a tristate during interrupt acknowledge and local bus hold acknowledge cycles.

A_{19}/S_6,A_{18}/S_5,A_{17}/S_4, A_{16}/S_3 These are the time multiplexed address and status lines. During T_1, these are the most significant address lines for memory operations. During I/O operations, these lines are low. During memory or I/O operations, status information is available on those lines for T_2,T_3,T_w and T_4. The status of the interrupt enable flag bit(displayed on S_5) is updated at the beginning of each clock cycle. The S_4 and S_3 together indicate which segment register is presently being used for memory accesses, as shown in Table 1.1. These lines float to tri-state off (tristated) during the local bus hold acknowledge. The status line S_6 is always low (logical). The address bits are separated from the status bits using latches controlled by the ALE signal.

Table 1.1 *Status*

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

$\overline{BHE}/S_7$-Bus High Enable/Status The bus high enable signal is used to indicate the transfer of data over the higher order (D_{15}–D_8) data bus as shown in Table 1.2. It goes low for the data transfers over D_{15}–D_8 and is used to derive chip selects of odd address memory bank or peripherals. $\overline{BHE}$ is low during T_1 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 T_2, T_3 and T_4. The signal is active low and is tristated during 'hold'. It is low during T_1 for the first pulse of the interrupt acknowledge cycle. S_7 is not currently used.

Table 1.2 *Bus High Enable and A_0*

$\overline{BHE}$	A_0	*Indication*
0	0	Whole word (2 bytes)
0	1	Upper byte from or to odd address.
1	0	Lower byte from or to even address
1	1	None

$\overline{RD}$-Read Read signal, when low, indicates the peripherals that the processor is performing a memory or I/O read operation. $\overline{RD}$ is active low and shows the state for T_2, T_3, T_w 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.

$\overline{\text{TEST}}$ This input is examined by a 'WAIT' instruction. If the $\overline{\text{TEST}}$ 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 synchronised.

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/$\overline{\text{MX}}$ 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/$\overline{\text{IO}}$-Memory/IO This is a status line logically equivalent to $\overline{S_2}$ in the 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 T_4 and remains active till final T_4 of the current cycle. It is tristated during local bus "hold acknowledge".

$\overline{\text{INTA}}$-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 T_2, T_3 and T_w 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/$\overline{\text{R}}$-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 $\overline{S}_1$ in maximum mode. Its timing is the same as M/$\overline{\text{IO}}$. This is tristated during 'hold acknowledge'.

$\overline{\text{DEN}}$-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 T_2 until the middle of T_4. $\overline{\text{DEN}}$ 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 T_4 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.

$\overline{S}_2$, $\overline{S}_1$, $\overline{S}_0$-Status Lines These are the status lines which indicate the type of operation, being carried out by the processor. These become active during T_4 of the previous cycle and remain active during T_1 and T_2 of the current bus cycle. The status lines return to passive state during T_3 of the current bus cycle so that they may again become active for the next bus cycle during T_4. Any change in these lines during T_3 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 1.3.

Table 1.3

$\overline{S}_2$	$\overline{S}_1$	$\overline{S}_0$	*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

$\overline{\text{LOCK}}$ This output pin indicates that other system bus masters will be prevented from gaining the system bus, while the $\overline{\text{LOCK}}$ signal is low. The $\overline{\text{LOCK}}$ 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.

QS_1, QS_0-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 1.4.

Table 1.4

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*.

In the beginning, 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, the execution unit and the 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 1.6 explains the queue operation.

$\overline{RQ}/\overline{GT}_0$, $\overline{RQ}/\overline{GT}_1$ -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 $\overline{RQ}/\overline{GT}_0$ having higher priority than $\overline{RQ}/\overline{GT}_1$. $\overline{RQ}/\overline{GT}$ 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 T_4 (current) or T_1 (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 in the 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.

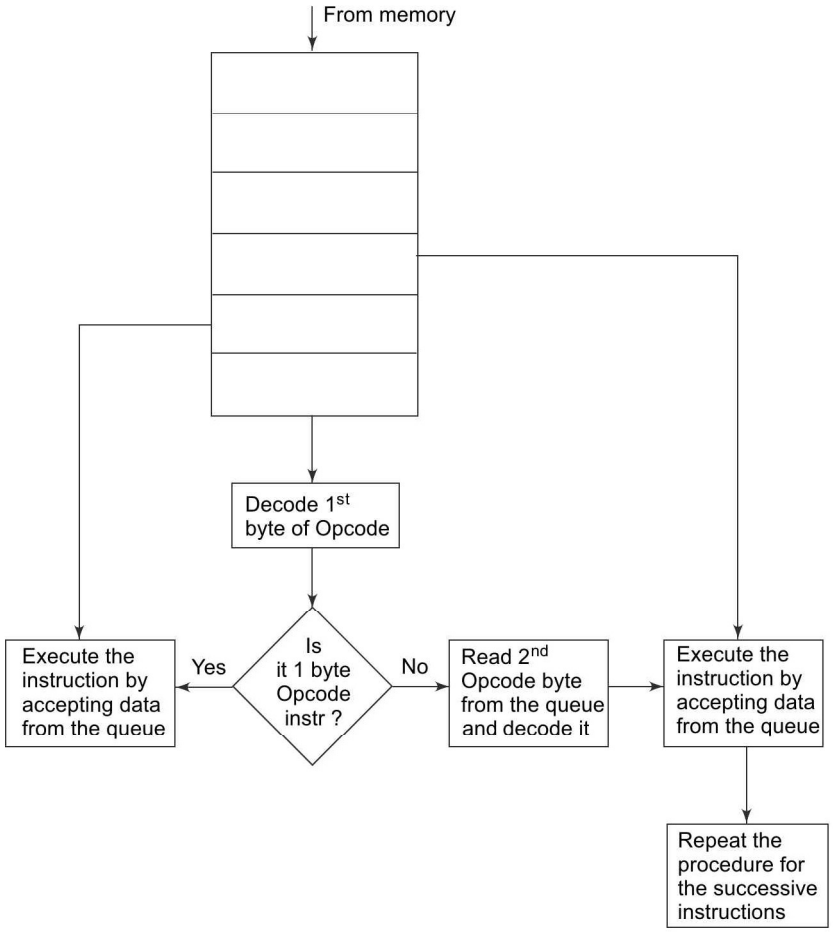


Fig. 1.6 *The Queue Operation*

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 the 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.

1.4 PHYSICAL MEMORY ORGANISATION

In an 8086 based system, the 1Mbytes memory is physically organised as an odd bank and an even bank, each of 512 Kbytes, addressed in parallel by the processor. Byte data with an even address is transferred on D_7-D_0, while the byte data with an odd address is transferred on $D_{15}-D_8$ buss lines. The processor provides two enable signals, $\overline{BHE}$ and A_0 for selection of either even or odd or both the banks. The instruction

stream is fetched from memory as words and is addressed internally by the processor as necessary. In other words, if the processor fetches a word (consecutive two bytes) from memory, there are different possibilities, like:

1. Both the bytes may be data operands
2. Both the bytes may contain opcode bits
3. One of the bytes may be opcode while the other may be data

All the above possibilities are taken care of by the internal decoder circuit of the microprocessor. The opcodes and operands are identified by the internal decoder circuit which further derives the signals those act as input to the timing and control unit. The timing and control unit then derives all the signals required for execution of the instruction.

While referring to word data, the BIU requires one or two memory cycles, depending upon whether the starting byte is located at an even or odd address. It is always better to locate the word data at an even address. To read or write a complete word from/to memory, if it is located at an even address, only one read or write cycle is required. If the word is located at an odd address, the first read or write cycle is required for accessing the lower byte while the second one is required for accessing the upper byte. Thus, two bus cycles are required, if a word is located at an odd address. It should be kept in mind that while initialising the structures like stack they should be initialised at an even address for efficient operation.

8086 is a 16-bit microprocessor and hence can access two bytes of data in one memory or I/O read or write operation. But the commercially available memory chips are only byte size, i.e. they can store only one byte in a memory location. Obviously, to store 16-bit data, two successive memory locations are used and the lower byte of 16-bit data can be stored in the first memory location while the second byte is stored in the next location. In a sixteen bit read or write operation both of these bytes will be read or written in a single machine cycle.

A map of an 8086 memory system starts at 00000H and ends at FFFFFH. 8086 being a 16-bit processor is expected to access 16-bit data to / from 8-bit commerically available memory chips in parallel, as shown below in Fig. 1.7.

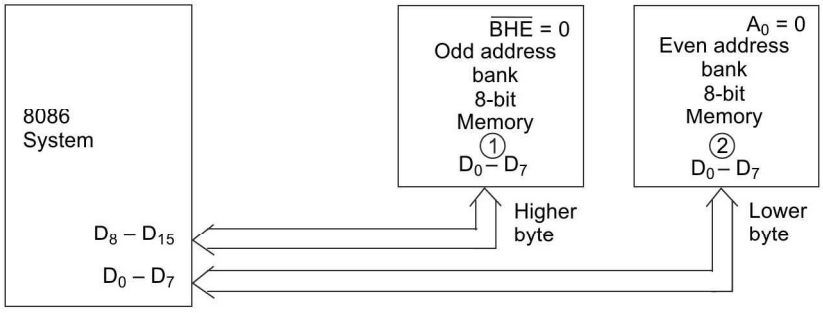


Fig. 1.7 *Physical Memory Organisation*

Thus, bits D_0–D_7 of a 16-bit data will be transferred over D_0–D_7 (lower byte) of 16-bit data bus to / from 8-bit memory (2) and bit D_8–D_{15} of the 16-bit data will be transferred over D_8–D_{15} (higher byte) of the 16-bit data bus of the microprocessor, to / from 8-bit memory (1). Thus to achieve 16-bit data transfer using 8-bit memories, in parallel, the map of the complete system byte memory addresses will obviously be divided into the two memory banks as shown in Fig. 1.7.

The lower byte of a 16-bit data is stored at the first address of the map 00000H and it is to be transferred over D_0–D_7 of the microprocessor bus so 00000H must be in 8-bit memory (2). Higher byte of the 16-bit data is stored in the next address 00001H; it is to be transferred over D_8–D_{15} of the microprocessor bus so the address 00001H must be in 8-bit memory (1) of Fig. 1.7. On similar lines, for the next 16-bit data stored in the memory, immediately after the previous one, the lower byte will be stored at the next address 00002H and it must be in 8-bit memory (2) while the higher byte will be stored at the next address 00003H that must be in 8-bit memory (1). Thus, if it is imagined that the complete memory map of 8086 is filled with 16-bit data, all the lower bytes (D_0–D_7) will be stored in the 8-bit memory bank (2) and all the higher bytes (D_8–D_{15}) will be stored in the 8-bit memory bank (1). Consequently, it can be observed that all the lower bytes have to be stored at even addresses and all the higher bytes have to be stored at odd addresses. Thus, the 8-bit memory bank (1) will be called an odd address bank and the 8-bit memory bank (2) will be called an even address bank. The complete memory map of 8086 system is thus divided into even and odd address memory banks.

If 8086 transfers a 16-bit data to / from memory, both of these banks must be selected for the 16-bit operation. However, to maintain an upward compatibility with 8085, 8086 must be able to implement 8-bit operations. In which case, two possibilities arise; the first being 8-bit operation with even memory bank, i.e. with an even address and the second one is 8-bit operation with odd address memory bank, i.e. with an odd address. The two signals A_0 and $\overline{BHE}$ solve the problem of selection of appropriate memory banks as presented in Table 1.2.

Certain locations in memory are reserved for specific CPU operations. The locations from FFFF0H to FFFFFH are reserved for operations including jump to initialisation programme and I/O-processor initialisation. The locations 00000H to 003FFH are reserved for *interrupt vector table*. The interrupt structure provides space for a total of 256 interrupt vectors. The vectors, i.e. CS and IP for each interrupt routine requires 4 bytes for storing it in the interrupt vector table. Hence, 256 types of interrupt require 256 C 4 = 03FFH (1Kbyte) locations for the complete interrupt vector table.

1.5 GENERAL BUS OPERATION

The 8086 has a combined address and data bus commonly referred to as a time multiplexed address and data bus. The main reason behind multiplexing address and data over the same pins is the maximum utilisation of processor pins and it facilitates the use of 40 pin standard DIP package. The bus can be demultiplexed using a few latches and transreceivers, whenever required. In the following text, we will discuss a general bus operation cycle.

Basically, all the processor bus cycles consist of at least four clock cycles. These are referred to as T_1, T_2,T_3 and T_4. The address is transmitted by the processor during T_1. It is present on the bus only for one cycle. During T_2, i.e. the next cycle, the bus is tristated for changing the direction of bus for the following data read cycle. The data transfer takes place during T_3 and T_4. In case, an addressed device is slow and shows 'NOT READY' status the wait states T_w are inserted between T_3 and T_4. These clock states during wait period are called *idle states* (T_i), *wait states* (T_w) or *inactive states*. The processor uses these cycles for internal housekeeping. The Address Latch Enable (ALE) signal is emitted during T_1 by the processor (minimum mode) or the bus controller (maximum mode) depending upon the status of the MN/$\overline{MX}$ input. The negative edge of this ALE pulse is used to separate the address and the data or status information. In maximum mode, the status lines $\overline{S}_0$, $\overline{S}_1$ and $\overline{S}_2$ are used to indicate the type of operation as discussed in the signal description section of this chapter. Status bits $\overline{S}_3$ to $\overline{S}_7$ are multiplexed with higher order address bits and the $\overline{BHE}$ signal. Address is valid during T_1 while the status bits S_3 to S_7 are valid during T_2 through T_4. Figure 1.8 shows a general bus operation cycle of 8086.

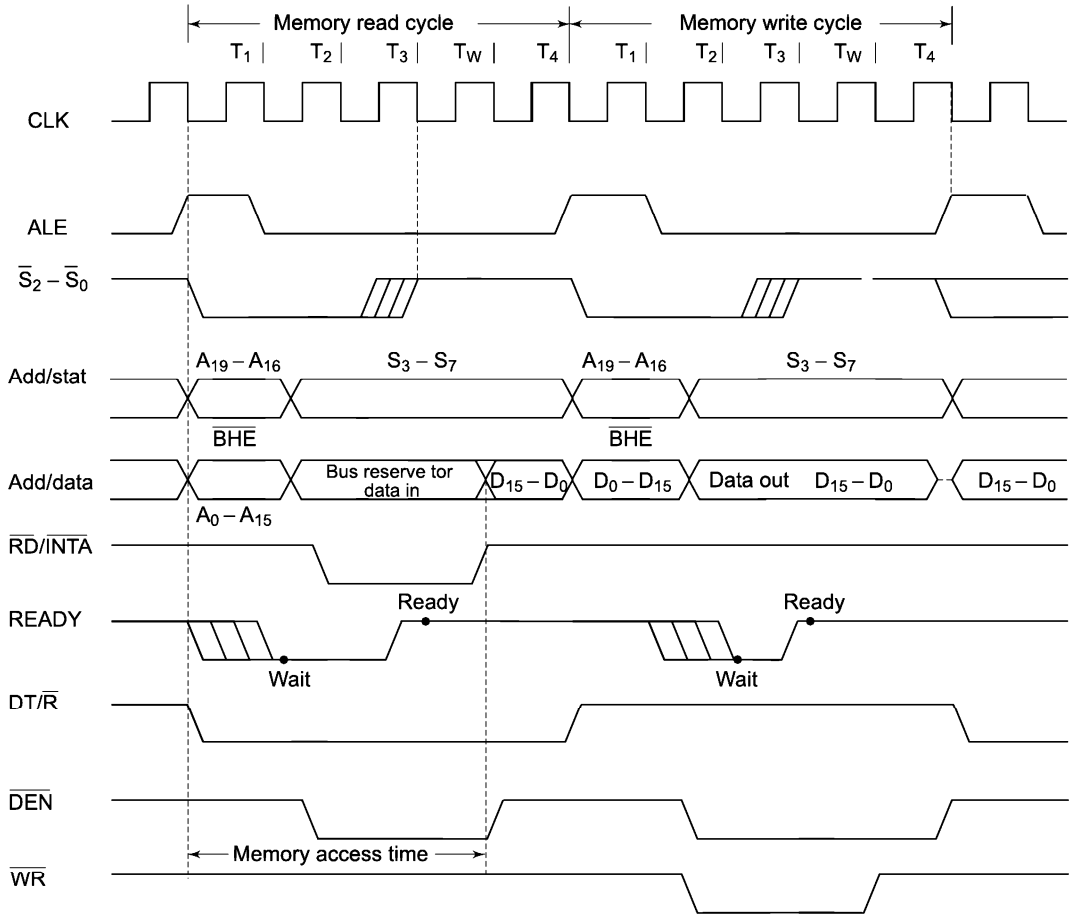


Fig. 1.8 *General Bus Operation Cycle of 8086*

1.6 I/O ADDRESSING CAPABILITY

The 8086/8088 processor can address up to 64K I/O byte registers or 32K word registers. The limitation is that the address of an I/O device must not be greater than 16 bits in size, this means that a maximum number of 2^{16}, i.e. 64Kbyte I/O devices may be accessed by the CPU. The I/O address appears on the address lines A_0 to A_{15} for one clock cycle (T_1). It may then be latched using the ALE signal. The upper address lines (A_{16}–A_{19}) are at logic 0 level during the I/O operations.

The 16-bit register DX is used as 16-bit I/O address pointer, with full capability to address up to 64K devices. In this case, the I/O ports are addressed in the same manner as memory locations in the based addressing mode using BX. In memory mapped I/O interfacing, the I/O device addresses are treated as memory locations in page 0, i.e. segment address 0000H. Even addressed bytes are transferred on D_7–D_0 and odd addressed bytes are transferred on D_8–D_{15} lines. While designing any 8-bit I/O system around 8086, care must be taken that all the byte registers in the system should be even addressed. Figure 1.9 shows 8086 IO addressing scheme.

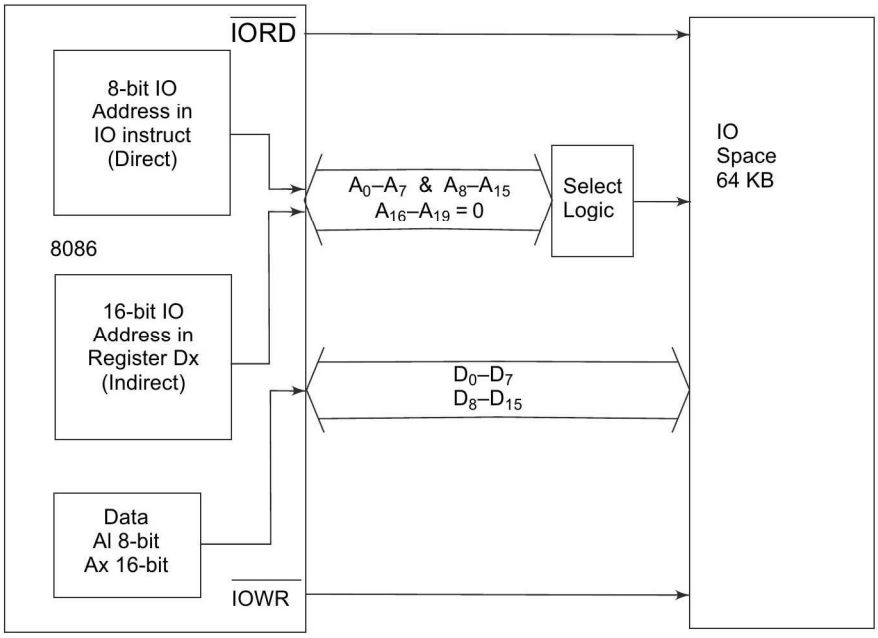


Fig. 1.9 *8086 IO Addressing*

1.7 SPECIAL PROCESSOR ACTIVITIES

1.7.1 Processor Reset and Initialisation

When logic 1 is applied to the RESET pin of the microprocessor, it is reset. It remains in this state till logic 0 is again applied to the RESET pin. The 8086 terminates the on-going operation on the positive edge of the reset signal. When the negative edge is detected, the reset sequence starts and is continued for nearly 10 clock cycles. During this period, all the internal register contents are set to 0000H except CS is set to value FFFFH . Thus, the execution starts again from the physical address FFFF0H. Due to this, the EPROM in an 8086 system is interfaced so as to have the physical memory locations FFFF0H to FFFFFH in it, i.e. at the end of the map.

For the reset signal to be accepted by 8086, it must be high for at least 4 clock cycles. From the instant the power is on, the reset pulse should not be applied to 8086 before 50 μs to allow proper initialisation of 8086. In the reset state, all 3-state outputs are tristated. Status signals are active in idle state for the first clock cycle after the reset becomes active, and then floats to tristate. The ALE and HLDA lines are driven low during the reset operation.

Non-maskable interrupt enable request, which appears before the second clock after the end of the reset operation, will not be served. For the NMI request to be served, it must appear after the second clock cycle during reset initialisation or later. If a HOLD request appears immediately after RESET, it will be immediately served after initialisation, before execution of any instruction.

1.7.2 HALT

When the processor executes a HLT instruction, it enters the 'halt' state. However, before doing so, it indicates that it is entering 'halt' state in two ways, depending upon whether it is in the minimum or maximum mode. When the processor is in minimum mode and wants to enter halt state, it issues an ALE pulse but does not issue any control signal. When the processor is in maximum mode and wants to enter the halt state, it puts the HALT status (011) on $\overline{S}_2$, $\overline{S}_1$ and $\overline{S}_0$ pins and then the bus controller issues an ALE pulse but no qualifying signal, i.e.

no appropriate address or control signals are issued to the bus. Only an interrupt request or reset will force the 8086 to come out of the 'halt' state. Even the HOLD request cannot force the 8086 out of 'halt' state.

1.7.3 TEST and Synchronisation with External Signals

Besides the interrupt, hold and general I/O capabilities, the 8086 has an extra facility of the $\overline{\text{TEST}}$ signal. When the CPU executes a WAIT instruction, the processor preserves the contents of the registers, before execution of the WAIT instruction, and the CPU waits for the $\overline{\text{TEST}}$ input pin to go low. If the $\overline{\text{TEST}}$ pin goes low, it continues further execution, otherwise, it keeps on waiting for the $\overline{\text{TEST}}$ pin to go low. For the $\overline{\text{TEST}}$ signal to be accepted, it must be low for at least 5 clock cycles. The activity of waiting does not consume any bus cycle. The processor remains in the idle state while waiting. While waiting, any 'HOLD' request from an external device may be served. If an interrupt occurs when the processor is waiting, it fetches the wait instruction once more, executes it, and then serves the interrupt. After returning from the interrupt, it fetches the wait instruction once more and continues in the 'wait' state.

Thus, the execution of the portion of a program which appears in the program after WAIT instruction can be synchronized with an external signal connected with the $\overline{\text{TEST}}$ input.

1.7.4 Deriving System Bus

The 8086 has a multiplexed 16-bit address / data bus (A D_0–A D_{15}) and a multiplexed 4-bit address / status bus A_{16} / S_3–A_{19} / S_6. The address can be latched using signal ALE, as shown in Fig. 1.10. Commercially available latch chips contain eight latches. Thus for demultiplexing twenty address lines one requires three latch chips like 74373. While demultiplexing the address bus, two of the three latch chips will be fully used and four latches of the third chip will be used. Figure 1.10 shows arrangement for latching the twenty bit address. Di indicate D inputs of latches and Q indicate the respective latch outputs.

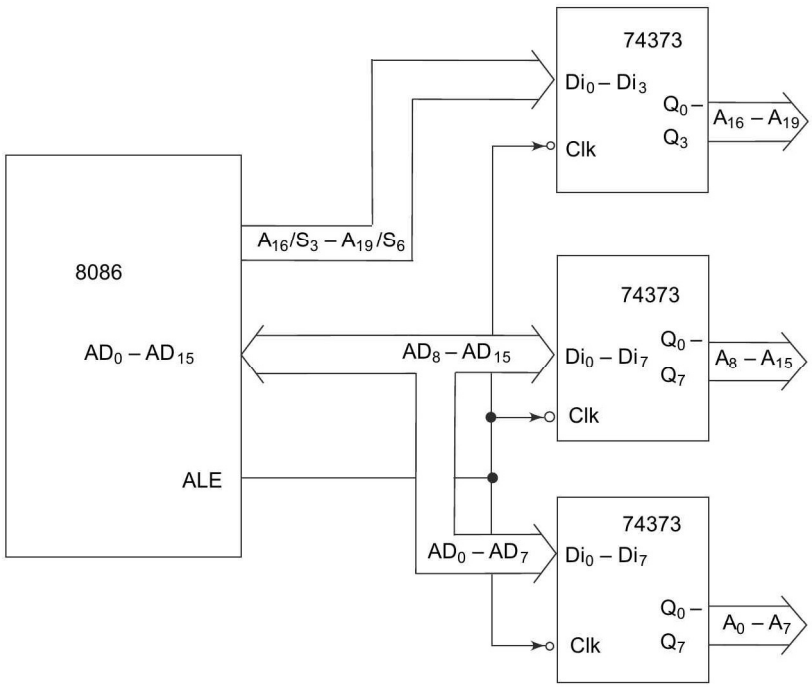


Fig. 1.10 *Latching 20-Bit Address of 8086*

8086 has multiplexed 16-bit data bus in the form of AD_0–AD_{15}. The data can be separated from the address and buffered using two bidirectional buffers 74245. It may be noted that the data can either be transferred from microprocessor to memory or from memory to microprocessor in case of write or read operations respectively hence bidirectional buffers are required for deriving the data bus. The signals $\overline{DNE}$ and DT / $\overline{R}$ indicate the presence of data on the bus and the direction of the data, i.e. to / from the microprocessor. They are used to drive the chip select (enable) and direction pins of the buffers as indicated below in Fig. 1.11.

If $\overline{DNE}$ is low it indicates that the data is available on the multiplexed bus and both the buffers (74245) are enabled to transfer data. When DIR pin goes high the data available at X pins of 74245 are transferred to Y pins, i.e. data is transmitted from microprocessor to either memory or IO device (write operation). If DIR pin goes low the data available at Y pins of 74245 is transferred to X pins, i.e. data is received by microprocessor from memory or IO device (read operation).

For deriving control bus from the available control signals $\overline{RD}$, $\overline{WR}$ and M / $\overline{IO}$ in case of minimum mode of operation any combinational logic circuit may be used as shown in Fig. 1.12 (a) and Fig. 1.12 (b).

In case of maximum mode of operation a chip bus controller derives all the control signals using status signals $\overline{S}_0$, $\overline{S}_1$ and $\overline{S}_2$.

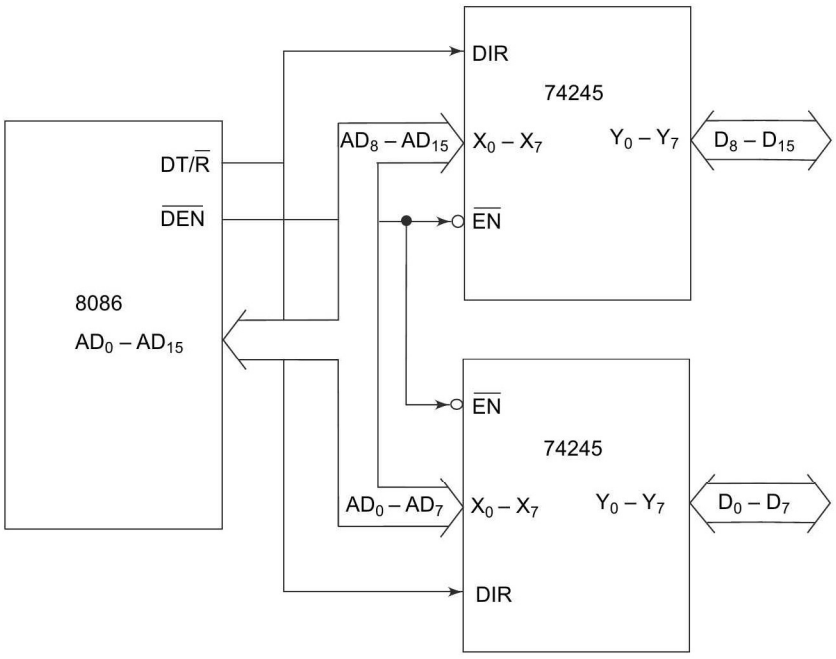


Fig. 1.11 *Buffering Data Bus of 8086*

1.8 MINIMUM MODE 8086 SYSTEM AND TIMINGS

In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its MN/$\overline{MX}$ 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, transreceivers, clock generator, memory and I/O devices. Some type of chip selection logic may be required for selecting memory or I/O devices, depending upon the address map of the system.

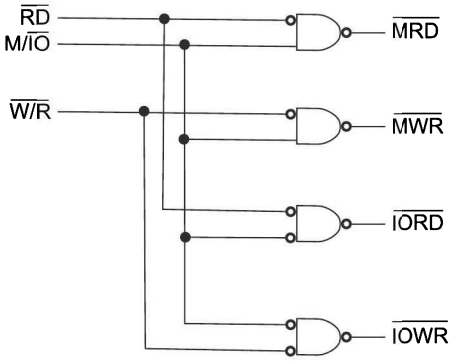


Fig. 1.12 (a) *Deriving 8086 Control Signals*

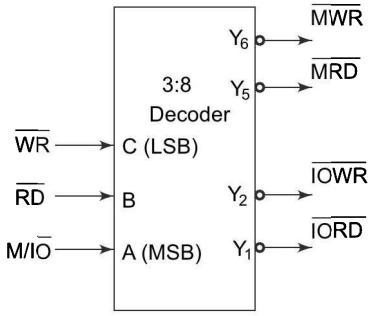


Fig. 1.12 (b) *Deriving 8086 Control Signals*

The 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. Transreceivers are the bidirectional buffers and sometimes they are called data amplifiers. They are required to separate the valid data from the time multiplexed address/data signal. They are controlled by two signals, namely, $\overline{DEN}$ and DT/$\overline{R}$. The $\overline{DEN}$ signal indicates that the valid data is available on the data bus, while DT/$\overline{R}$ indicates the direction of data, i.e. from / to the processor. The system contains memory for the monitor and users program storage. Usually, EPROMS are used for monitor storage, while RAMs for users' program storage. A system may contain I/O devices for communication with the processor as well as some special purpose I/O devices. The clock generator (IC8284) generates the clock from the crystal oscillator and then shapes it to make it more precise so that it can be used as an accurate timing reference for the system. The clock generator also synchronizes some external signals with the system clock. The general system organisation is shown in Fig. 1.13. Since it has 20 address lines and 16 data lines, the 8086 CPU requires three octal address latches and two octal data buffers for the complete address and data separation.

The working of the minimum mode configuration system can be better described in terms of the timing diagrams rather than qualitatively describing the operations. The opcode fetch and read cycles are similar. Hence, the timing diagram can be categorized in two parts, the first is the timing diagram for *read cycle* and the second is the timing diagram for *write cycle.*

The read cycle begins in T_1 with the assertion of the Address Latch Enable (ALE) signal and M/$\overline{IO}$ signal. During the negative going edge of this signal, the valid address is latched on the local bus. The $\overline{BHE}$ and A_0 signals address low, high or both bytes. From T_1 to T_4, the M/$\overline{IO}$ signal indicates a memory or I/O operation. At T_2, the address is removed from the local bus and is sent to the output. The bus is then tristated. The Read ($\overline{RD}$) control signal is also activated in T_2. This signal causes the addressed device to enable its data bus drivers. After $\overline{RD}$ goes low, the valid data is available on the data bus. The addressed device will drive the READY line high. When the processor returns the read signal to high level, the addressed device will again tristate its bus drivers. CS logic indicates chip select logic and 'e' and 'O' sufixes indicate even and odd address memory banks.

A write cycle also begins with the assertion of ALE and the emission of the address. The M/$\overline{IO}$ signal is again asserted to indicate a memory or I/O operation. In T_2, after sending the address in T_1, the processor sends the data to be written to the addressed location. The data remains on the bus until the middle of T_4 state. The $\overline{WR}$ becomes active at the beginning of T_2 (unlike $\overline{RD}$ is somewhat delayed in T_2 to provide time for floating).

The $\overline{BHE}$ and A_0 signals are used to select the proper byte or bytes of memory or I/O word to be read or written as already discussed in the signal description section of this chapter.

The M/$\overline{IO}$, $\overline{RD}$ and $\overline{WR}$ signals indicate the types of data transfer as specified in Table 1.5.

Table 1.5

$M/\overline{IO}$	$\overline{RD}$	$\overline{DEN}$	*Transfer Type*
0	0	1	I/O read
0	1	0	I/O write
1	0	1	Memory read
1	1	0	Memory write

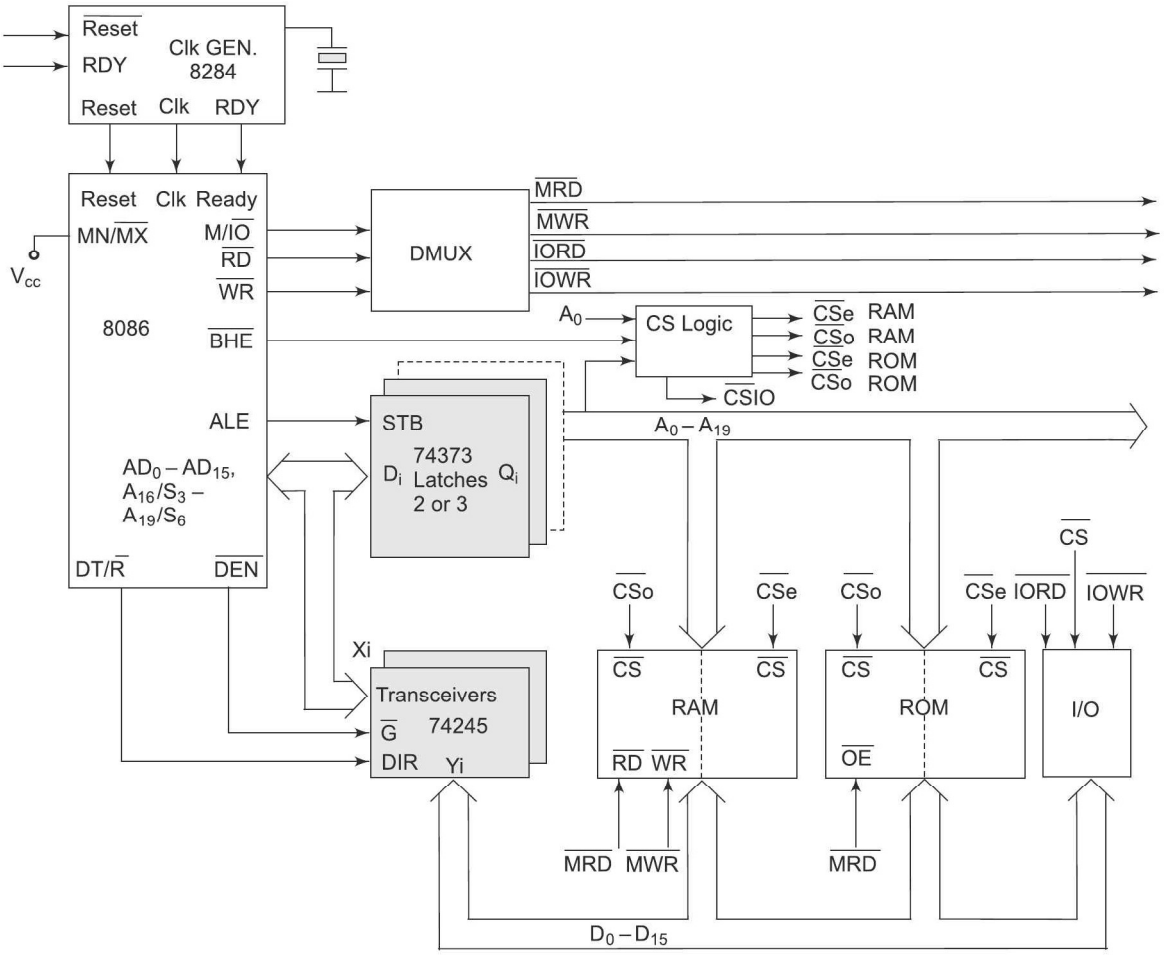


Fig. 1.13 *Minimum Mode 8086 System*

Figure 1.14(a) shows the read cycle while Fig. 1.14(b) shows the write cycle.

1.8.1 HOLD Response Sequence

The HOLD pin is checked at the end of each bus cycle. If it is received active by the processor before T_4 of the previous cycle or during T_1 state of the current cycle, the CPU activates HLDA in the next clock cycle and for the succeeding bus cycles, the bus will be given to another requesting master. The control of the bus is not regained by the processor until the requesting master does not drop the HOLD pin low. When the request

is dropped by the requesting master, the HLDA is dropped by the processor at the trailing edge of the next clock, as shown in Fig. 1.14 (c). The other conditions have already been discussed in the signal description section for the HOLD and HLDA signals.

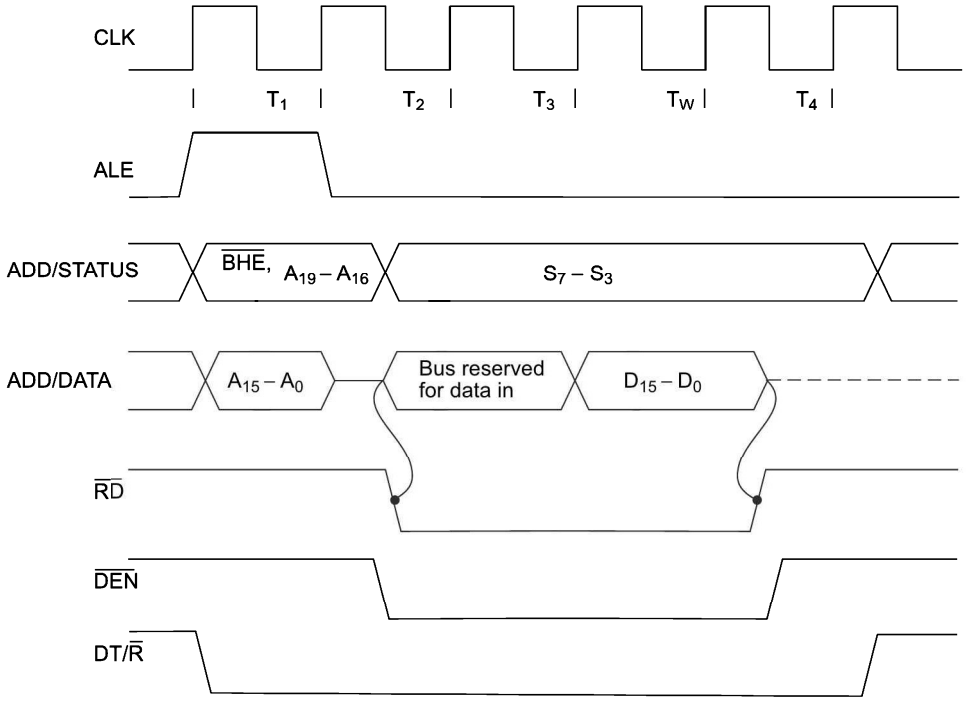


Fig. 1.14(a) *Read Cycle Timing Diagram for Minimum Mode*

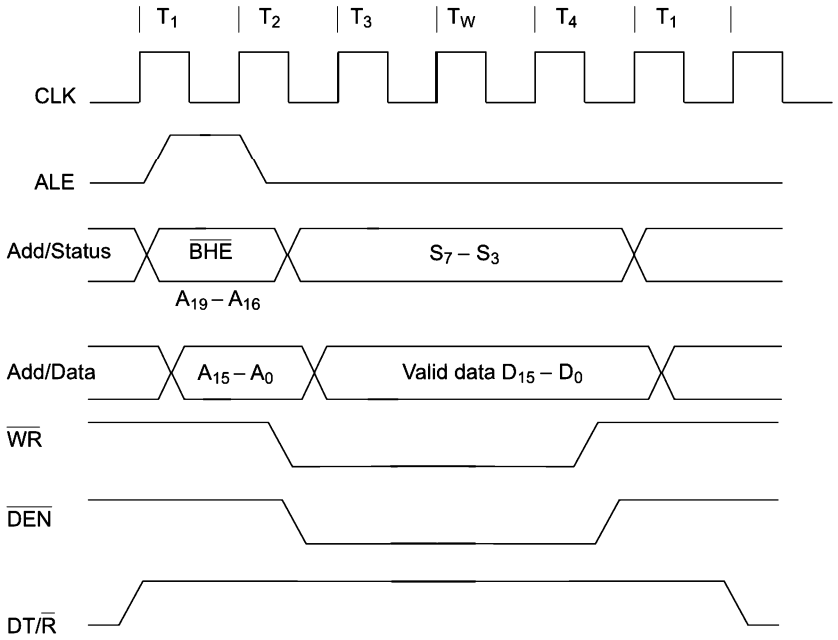


Fig. 1.14(b) *Write Cycle Timing Diagram for Minimum Mode Operation*

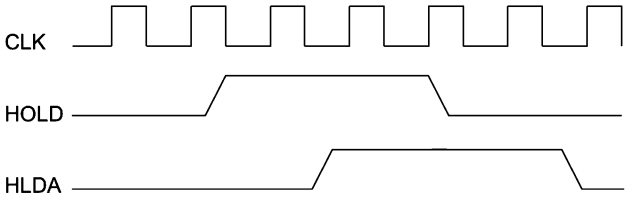


Fig. 1.14(c) *Bus Request and Bus Grant Timings in Minimum Mode System*

1.9 MAXIMUM MODE 8086 SYSTEM AND TIMINGS

In the maximum mode, the 8086 is operated by strapping the MN/$\overline{MX}$ pin to ground. In this mode, the processor derives the status signals $\overline{S}_2$, $\overline{S}_1$ and $\overline{S}_0$. Another chip called bus controller derives the control signals using this status information. In the maximum mode, there may be more than one microprocessor in the system configuration. The other components in the system are the same as in the minimum mode system. In this section, we will study the bus controller chip and its functions in brief. The functions of all the pins having special functions in maximum mode have already been discussed in the pin diagram section of this chapter.

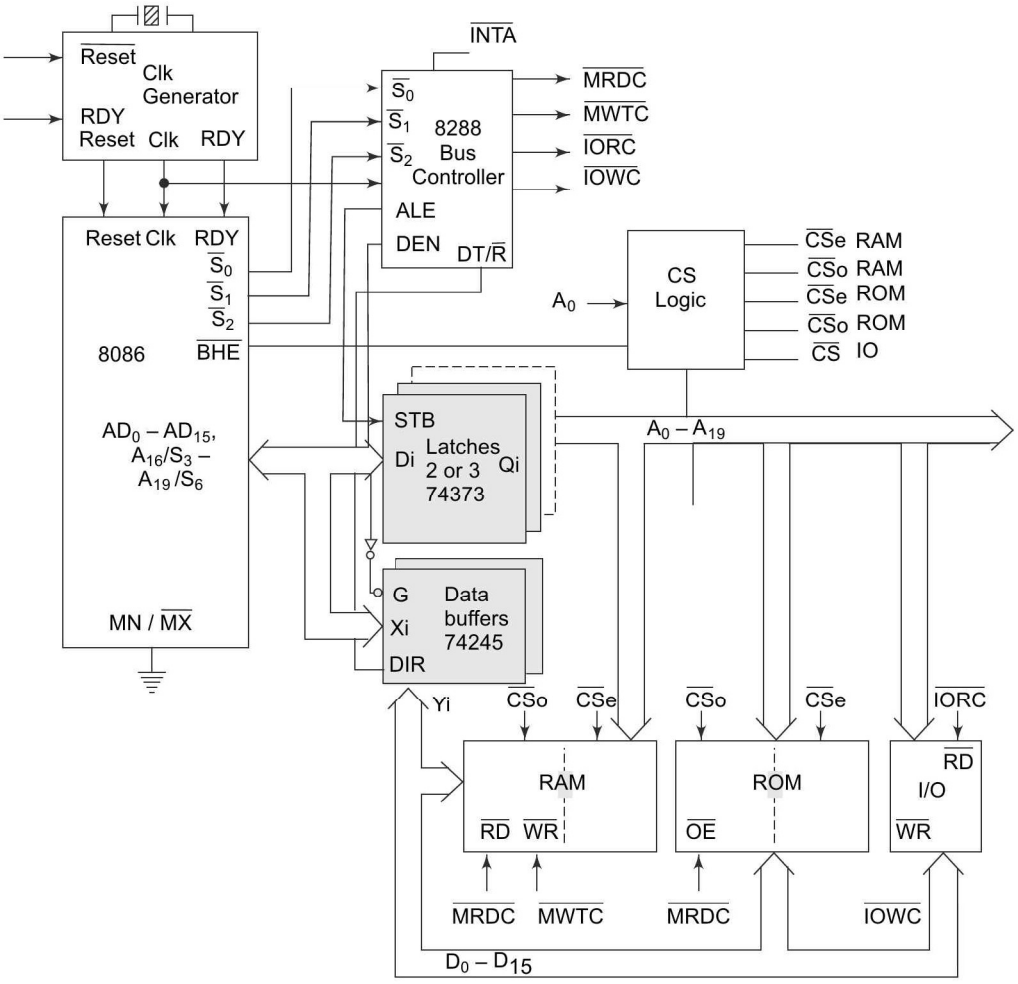


Fig. 1.15 *Maximum Mode 8086 System*

The basic functions of the bus controller chip IC8288, is to derive control signals like $\overline{RD}$ and $\overline{WR}$ (for memory and I/O devices), $\overline{DEN}$, DT/$\overline{R}$, ALE, etc. using the information made available by the processor on the status lines. The bus controller chip has input lines $\overline{S}_2$, $\overline{S}_1$ and $\overline{S}_0$ and CLK, which are driven by the CPU. It derives the outputs ALE, DEN, DT/$\overline{R}$, $\overline{MRDC}$, $\overline{MWTC}$, $\overline{AMWC}$, $\overline{IORC}$, $\overline{IOWC}$ and $\overline{AIOWC}$. The $\overline{AEN}$, IOB and CEN pins are specially useful for multiprocessor systems. $\overline{AEN}$ and IOB are generally grounded. CEN pin is usually tied to +5V. The significance of the MCE/$\overline{PDEN}$ output depends upon the status of the IOB pin. If IOB is grounded, it acts as master cascade enable to control cascaded 8259A, else it acts as peripheral data enable used in the multiple bus configurations. $\overline{INTA}$ pin is used to issue two interrupt acknowledge pulses to the interrupt controller or to an interrupting device.

$\overline{IORC}$, $\overline{IOWC}$ are I/O read command and I/O write command signals respectively. These signals enable an IO interface to read or write the data from or to the addressed port. The $\overline{MRDC}$, $\overline{MWTC}$ are memory read command and memory write command signals respectively and may be used as memory read and write signals. All these command signals instruct the memory to accept or send data to or from the bus. For both of these write command signals, the advanced signals namely $\overline{AIOWC}$ and $\overline{AMWTC}$ are available. They also serve the same purpose, but are activated one clock cycle earlier than the $\overline{IOWC}$ and $\overline{MWTC}$ signals, respectively. The maximum mode system is shown in Fig. 1.15.

The maximum mode system timing diagrams are also divided in two portions as read (input) and write (output) timing diagrams. The address/data and address/status timings are similar to the minimum mode. ALE is asserted in T_1, just like minimum mode. The only difference lies in the status signals used and the available control and advanced command signals. Figure 1.16 (a) shows the maximum mode timings for the read operation while the Fig. 1.16 (b) shows the same for the write operation. The CS Logic block represents chip select logic and the 'e' and 'O' suffixes indicate even and odd address memory bank.

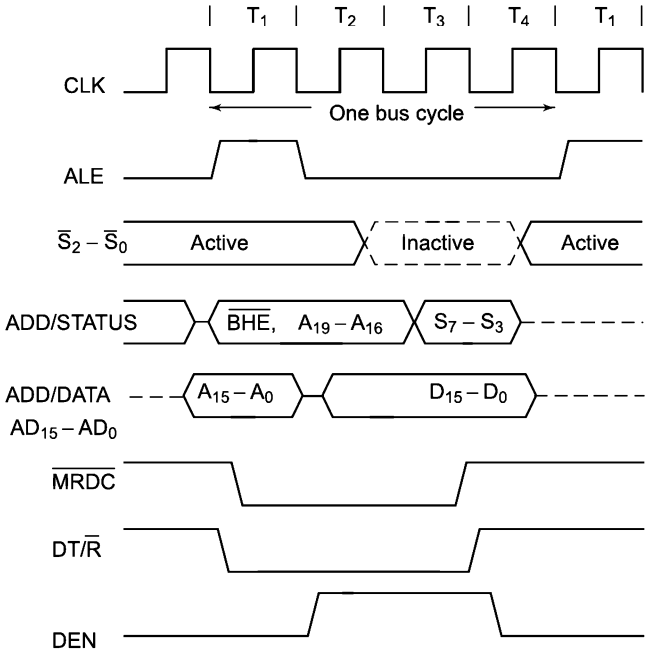


Fig. 1.16 (a) *Memory Read Timing in Maximum Mode*

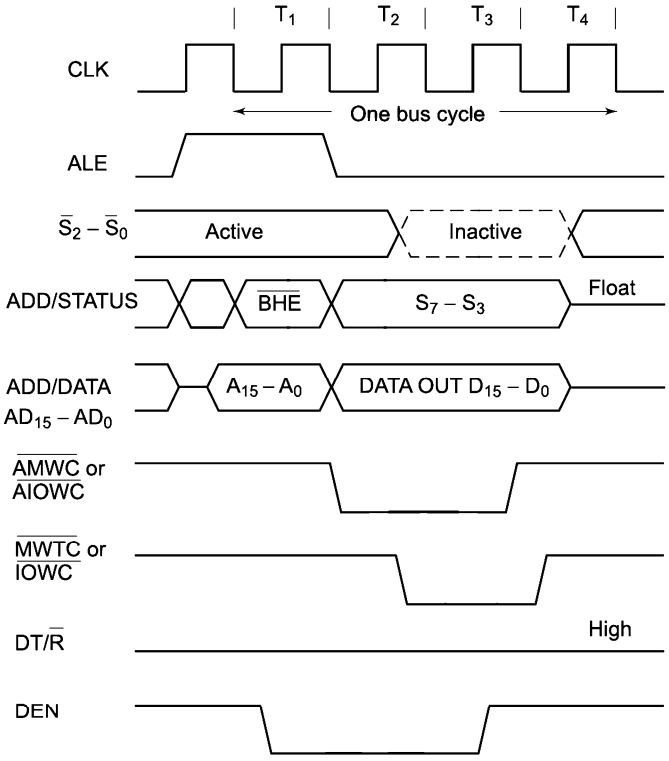


Fig. 1.16(b) *Memory Write Timing in Maximum Mode*

1.9.1 Timings for $\overline{RQ}/\overline{GT}$ Signals

The request/grant response sequence contains a series of three pulses as shown in the timing diagram Fig.1.16 (c). The request/grant pins are checked at each rising pulse of clock input. When a request is detected and if the conditions discussed in pin diagram section of this chapter for valid HOLD request are satisfied, the processor issues a grant pulse over the $\overline{RQ}/\overline{GT}_0$ pin immediately during the T_4 (current) or T_1 (next) state. When the requesting master receives this pulse, it accepts the control of the bus. The requesting master uses the bus till it requires. When it is ready to relinquish the bus, it sends a release pulse to the processor (host) using the $\overline{RQ}/GT$ pin. This sequence is shown in Fig. 1.16 (c).

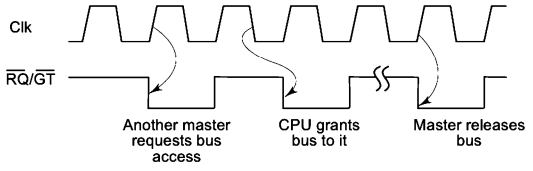


Fig. 1.16 (c) *$\overline{RQ}/\overline{GT}$ Timings in Maximum Mode*

1.10 THE PROCESSOR 8088

The launching of the processor 8086 is seen as a remarkable step in the development of high speed computing machines. Before the introduction of 8086, most of the circuits required for the different applications in computing and industrial control fields were already designed around the 8-bit processor 8085. The 8086 imparted

tremendous flexibility in the programming as compared to 8085. So naturally, after the introduction of 8086, there was a search for a microprocessor chip which has the programming flexibility like 8086 and the external interface like 8085, so that all the existing circuits built around 8085 can work as before, with this new chip. The chip 8088 was a result of this demand. The microprocesser 8088 has all the programming facilities that 8086 has, along with some hardware features of 8086, like 1Mbyte memory addressing capability, operating modes (MN/$\overline{\text{MX}}$), interrupt structure etc. However, 8088, unlike 8086, has 8-bit data bus. This feature of 8088 makes the circuits, designed around 8085, compatible with 8088, with little or no modification.

All the peripheral interfacing schemes with 8088 are the same as those for the 8-bit pro-cessors. The memory and I/O addressing schemes are now exactly similar to 8085 schemes except for the increased memory (1Mbyte) and I/O (64Kbyte) capabilities. The architecture shows the developments in 8088 over 8086. The abilities and limitations of 8088 are same as 8086. In this section, we will discuss those properties of 8088 which are different from that of 8086 in some respects.

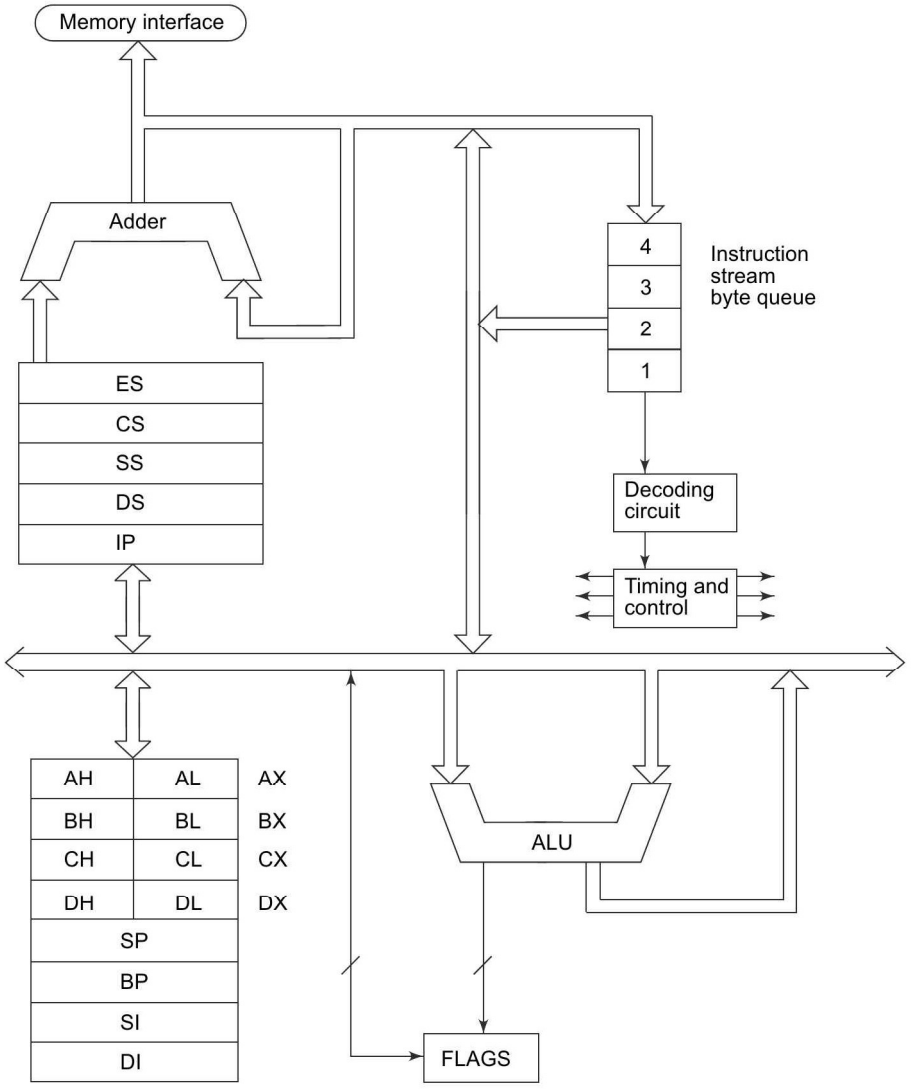


Fig. 1.17 *Architecture of 8088*