MICROPROCESSOR & MICROCONTROLLER

LECTURE NOTES

B.TECH
(III YEAR – II SEM)
(2017-18)

Prepared by:

Mr. K Murali Krishna, Associate Professor

Mrs. Vaidehi, Assistant Professor

Department of Electronics and Communication Engineering

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12 (B) of UGC ACT 1956

(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)

Maisammaguda, Dhulapally (Post Via. Kompally), Secunderabad – 500100, Telangana State, India

MALLA REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

III Year B.Tech. ECE-II Sem

L	T/P/D	C
5	-/-/-	4

(R15A0414) MICROPROCESSORS AND MICROCONTROLLERS

OBJECTIVES:

- To understand the basics of microprocessors and microcontrollers architectures and its functionalities
- To develop an in-depth understanding of the operation of microprocessors and microcontrollers, machine language programming & interfacing techniques.
- To design and develop Microprocessor/ microcontroller based systems for real time applications using low level language like ALP.

UNIT -I:

8086 Architecture: Architecture of 8086, Register Organization, Programming Model, Memory addresses, Memory Segmentation, Physical Memory Organization, Signal descriptions of 8086- Common Function Signals, Minimum and Maximum mode signals, Timing diagrams.

UNIT -II:

Instruction Set and Assembly Language Programming of 8086: Instruction formats, Addressing modes, Instruction Set, Assembler Directives, Procedures, Macros, Simple Programs involving Logical, Branch and Call Instructions, Sorting, Evaluating Arithmetic Expressions, String Manipulations.

UNIT -III:

I/O Interface: 8255 PPI, Various Modes of Operation and Interfacing to 8086, D/A and A/D Converter, stepper motor, Interfacing of DMA controller 8257

Interfacing with advanced devices: Memory Interfacing to 8086, Interrupt Structure of 8086, Vector Interrupt Table, Interrupt Service Routine, architecture of 8259

Communication Interface: Serial Communication Standards, Serial Data Transfer Schemes, 8251 USART Architecture and Interfacing.

UNIT -IV:

Introduction to Microcontrollers: Overview of 8051 Microcontroller, Architecture, I/O Ports, Memory Organization, Addressing Modes and Instruction set of 8051, Simple Programs, memory interfacing to 8051

UNIT -V:

8051 Real Time Control: Programming Timer Interrupts, Programming External Hardware Interrupts, Programming the Serial Communication Interrupts, Programming 8051 Timers and Counters

ARM Processor: Fundamentals, Registers , current program status register, pipeline, ~~Interrupt and the vector table.~~

TEXT BOOKS:

1. D. V. Hall, Microprocessors and Interfacing, TMGH, 2nd Edition 2006.
2. Kenneth. J. Ayala, The 8051 Microcontroller , 3rd Ed., Cengage Learning.
3. ARM System Developer's Guide: Designing and Optimizing System Software- Andrew N. Sloss, Dominic Symes, Chris Wright, Elsevier Inc., 2007

REFERENCE BOOKS:

1. Advanced Microprocessors and Peripherals – A. K. Ray and K.M. Bhurchandani, TMH, 2nd Edition 2006.
2. The 8051Microcontrollers, Architecture and Programming and Applications -K.Uma Rao, Andhe Pallavi, Pearson, 2009.
3. Micro Computer System 8086/8088 Family Architecture, Programming and Design - Liu and GA Gibson, PHI, 2nd Ed.
4. Microcontrollers and Application - Ajay. V. Deshmukh, TMGH, 2005.

OUTCOMES:

After going through this course the student will be able to

- The student will learn the internal organization of popular 8086/8051 microprocessors/microcontrollers.
- The student will learn hardware and software interaction and integration.
- The students will learn the design of microprocessors/microcontrollers-based systems

Unit I

8086 Architecture

Essence of the subject

- The microprocessor is the heart of the computer and it is a hardware component. Hence we being Electronics engineers, we need to study this subject. This is the essence of the subject.
- Various applications of microprocessor are
  - Educational field, Medical field, scientific labs, Banking sector etc.

Introduction to Microprocessors

- Intel introduced it's 4 bit microprocessor 4004 in 1971 and it's 8 bit microprocessor 8008 in 1972
- These microprocessors could not survive as general purpose microprocessors because of their design and performance limitations.
- Then the launch of a 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.

Introduction to Microprocessors cntd…

- The microprocessor 8085 followed by 8080, with a few more added features to it's architecture, which resulted in a functionally complete microprocessor.
- The main limitations of 8-bit microprocessor were
  - Low speed,
  - Low memory addressing capability
  - Limited number of general purpose registers
  - Less powerful instruction set
- All these limitations lead to the launching of 8086 microprocessor.
- In the family of 16 bit microprocessors, Intel's 8086 was the first one to be launched in 1978.

Introduction to Microprocessors cntd…

- 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 microprocessor.
- The peripheral chips designed earlier for 8085 were compatible with microprocessor 8086 with slight or no modifications.

Architecture of 8086

- The architecture of 8086 supports a 16 bit ALU, a set of 16 bit registers and provides the 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.
  - Bus interface unit
  - Execution Unit

Architecture of 8086 cntd…

- The bus interface unit contains the circuit for physical address calculations and a pre-decoding instruction byte queue (6 bytes long)
- The bus interface unit makes the system's bus signals available for external interfacing of the devices.

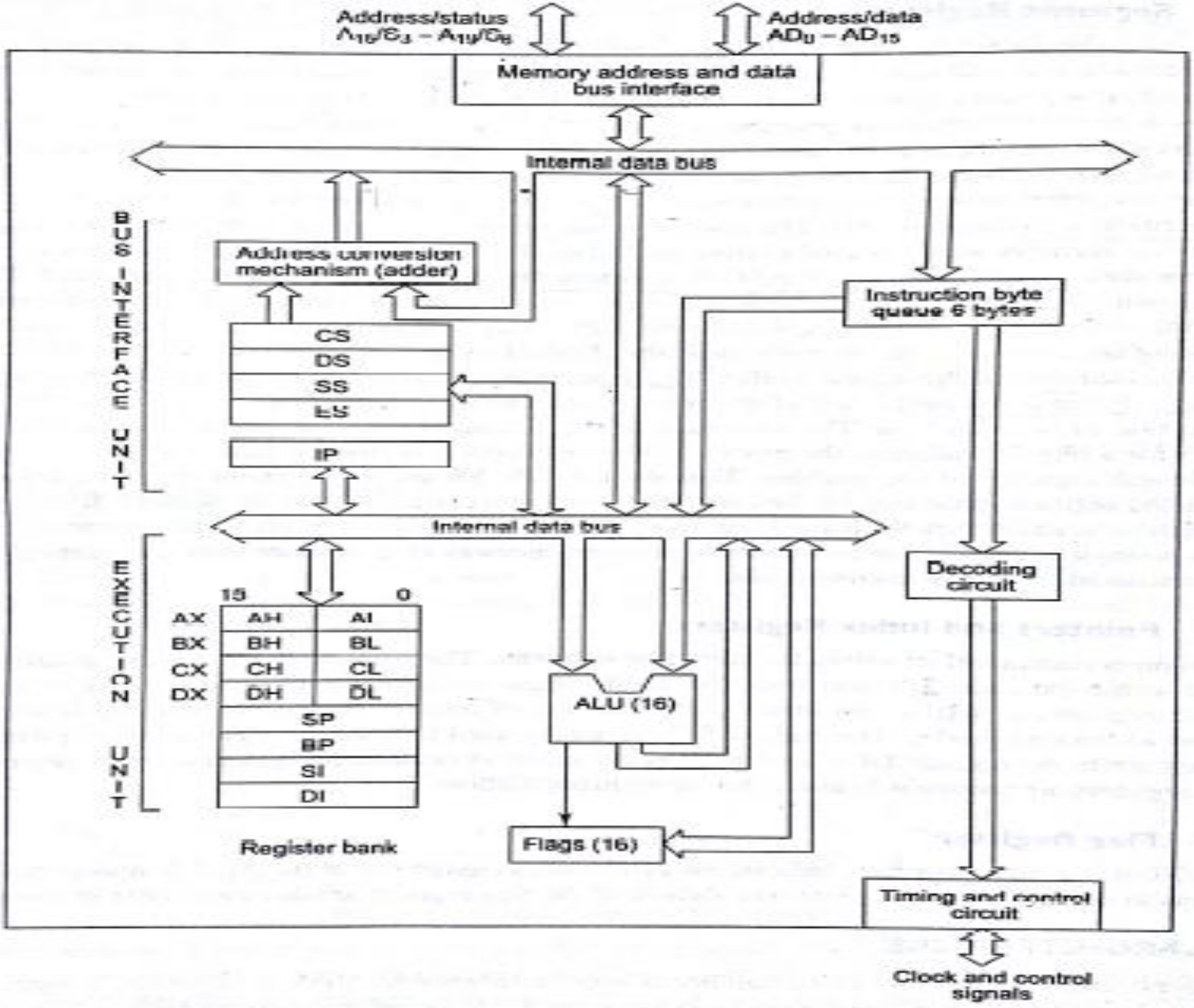


Fig. 1.2 *8086 Architecture*

Architecture of 8086 cntd…

- 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 bit long.

Architecture of 8086 cntd…

- 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.

Architecture of 8086 cntd…

- 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

Physical address = 1005 * 10 + 5555 = 155A5H

- Thus, the segment addressed by the segment value 1005H can have offset values from 0000H to FFFFH within it ie maximum 64 K locations may be accommodated in the segment.

Architecture of 8086 cntd…

- 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 a memory

Architecture of 8086 cntd…

- 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 op-code is fetched and decoded, the external bus remains free for some time, while processor internally executes the instruction.
- The time slot is utilized in 8086 to achieve the overlapped fetch and execution cycles.

Architecture of 8086 cntd…

- 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 pre-decoded instruction byte queue. It is a 6 byte long, first-in first-out structure.
- The instructions from the queue are taken for decoding sequentially.

Architecture of 8086 cntd…

- 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 op-code fetch cycle.
- While the op-code is fetched by the interface unit (BIU), the execution unit (EU) executes the previously decoded instruction concurrently.
- The BIU along with EU thus forms a pipeline.

Architecture of 8086 cntd…

- 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 register and IP.
- It has a 16-bit ALU, able to perform arithmetic and logical operation.

Architecture of 8086 cntd…

- The 16-bit flag register reflects the results of execution by the ALU.
- The decoding unit decodes the op-code bytes issued from the instruction byte queue.
- The timing and control unit derives the necessary control signals to execute the instruction op-code received from the queue, depending upon the information made available by the decoding circuit.
- The execution unit may pass the results to bus interface unit for storing them in memory.

Register organization of 8086

- 8086 has a powerful set of registers known as general purpose registers 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 other purposes like a counter or for storing offset address for some particular addressing modes etc.

Register organization of 8086 cntd…

- The special purpose registers are used as segment registers, pointers, index registers or as offset storage registers for particular addressing modes.
- The register set is categorized into four groups, as follows:
  - General data registers
  - Segment registers
  - Pointers and index registers
  - Flag register

Register organization of 8086 cntd…

General data registers:

- Figure 1.1 shows the register organization of 8086.
- The registers AX, BX, CX and DX are the general purpose 16 bit registers.
- AX is used as 16 bit accumulator → AH, AL
- AL can be used as an 8 bit accumulator for 8 bit operations. This is the most important general purpose register having multiple functions.

Register organization of 8086 cntd…

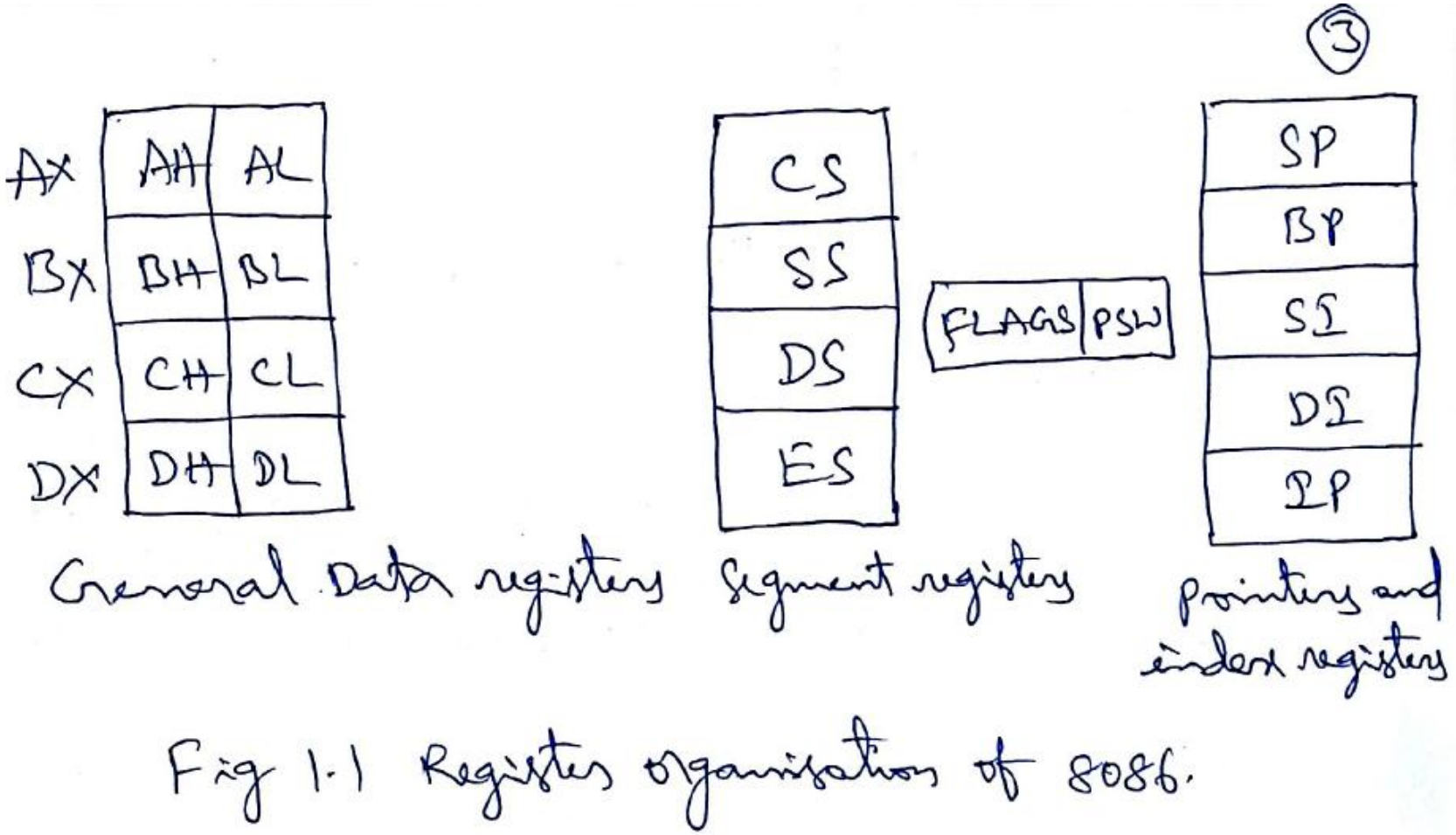


Fig 1.1 Registers organisation of 8086.

Register organization of 8086 cntd…

- Usually L and H specify the lower and higher bytes of a particular register.
- AX – accumulator,
  BX – offset storage,
  CX – counter,
  DX – to store data

Register organization of 8086 cntd…

Segment Registers:

- Unlike 8085, the 8086 addresses segmented memory.
- The complete 1 megabyte memory, which the 8086 addresses, is divided into 16 logical segments.
- Each segment thus contains 64 k bytes of memory.

Register organization of 8086 cntd…

- There are 4 segment registers, viz,
  Code segment register (CS) → Code,
  Data segment register (DS) → Data,
  Extra segment register (ES) → Data,
  Stack segment register (SS) → Stack related data
- The CPU uses the stack for temporarily storing important data.

Register organization of 8086 cntd…

- 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

Register organization of 8086 cntd…

- 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.
- It may be noted that all these segments are logical segments

Register organization of 8086 cntd…

Pointers and Index registers:

- The pointers contain offset within the particular segments.
- The pointers IP, BP and SP usually contain offsets with in the code (IP), 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

Register organization of 8086 cntd…

- The register SI is generally used to store the offset of the source data in data segment while the register DI is used to store the offset of the destination in data or extra segment.
- The index registers are particularly useful for string manipulations.

Register organization of 8086 cntd…

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 operation.

Programming Model

- An assembly language program model of 8086 is as follows

```
ASSUME DS:DATA,CS:CODE
DATA SEGMENT
.       (Declaration of data variables, constants etc)
.
DATA ENDS
CODE SEGMENT
START:
MOV AX,DATA
MOV DS,AX
.
.
CODE ENDS
END START
END
```

Memory Addresses

- As 8086 has got 20 address lines, it's addressing capability is 1 M Byte memory locations.
- The physical address is calculated from segment address and offset address as given below

Physical address=10*segment addr + offset addr

Memory Segmentation

- The memory in an 8086 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 64 Kbytes in size and is addressed by one of the segment register.
- The 16 bit contents of the segment register actually point to the starting location of a particular segment.

Memory Segmentation cntd…

- 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 64 K locations.
- To emphasize this segmented memory concept, we will consider an example of a housing colony containing say, 100 houses
  - Numbering the houses sequentially
  - Numbering the houses matrix wise (rows X columns)

10 X 10

Memory Segmentation cntd…

- 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 1 Mbytes of physical memory.
- The complete 1 Mbytes memory can be divided into 16 segments, each of 64 Kbytes size.

Memory Segmentation cntd…

- 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 1 Mbytes of physical memory.
- The complete 1 Mbytes memory can be divided into 16 segments, each of 64 Kbytes size.

Memory Segmentation cntd…

- The offset address values are from 0000H and FFFFH so that the physical addresses range from 00000H to FFFFFH.
- In the above said case, the segments are called non-overlapping segments which are shown in Figure 1.3a.

Memory Segmentation cntd…

- In some cases, however, the segments are overlapping.
- Suppose a segment starts at a particular address and its maximum size can be 64 Kbytes
- But, if another segment starts before this 64 kbytes locations of the first segment, the two segments are said to be overlapping segments

Memory Segmentation cntd…

- 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 1.3b explains the phenomenon more clearly.

Memory Segmentation cntd…

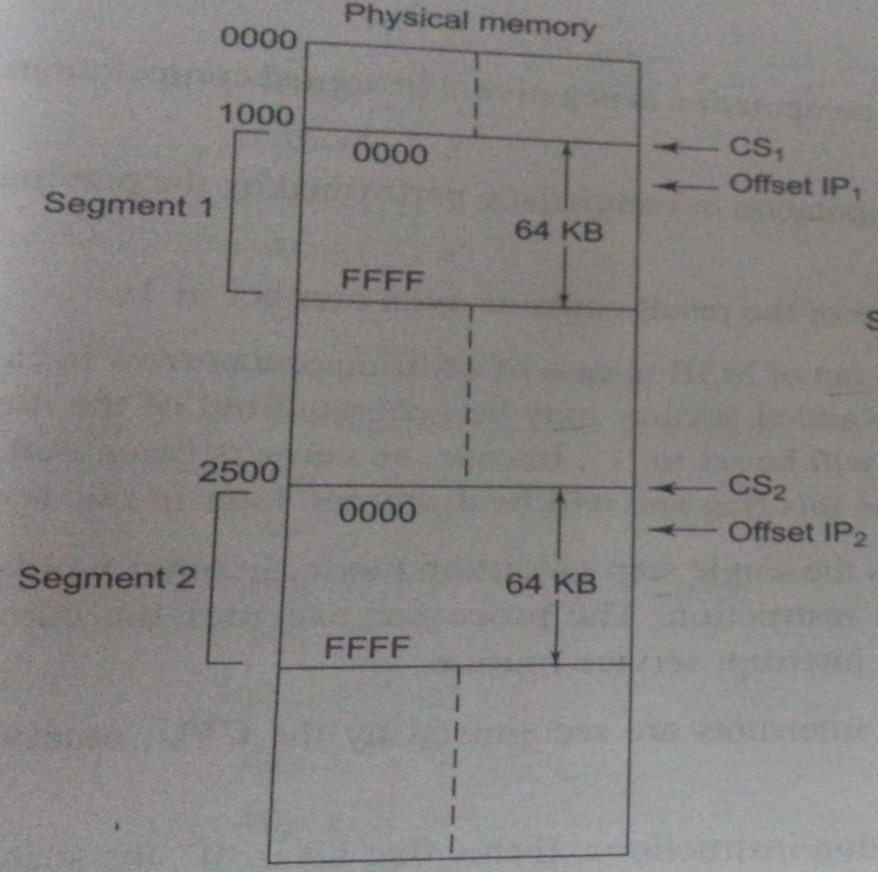


Fig. 1.3(a) *Non-overlapping Segments*

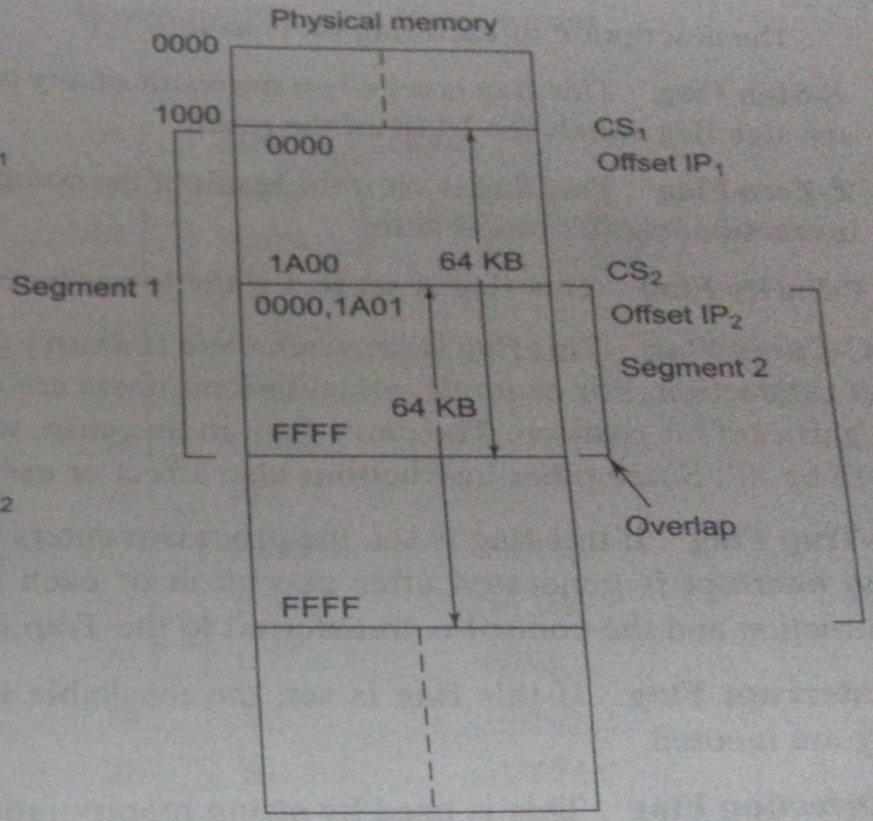


Fig. 1.3(b) *Overlapping Segments*

Memory Segmentation cntd…

- 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 1 Mbytes although the actual addresses to be handled are of 16 bit size.
  - 2 Allows the placing of the code, data, and stack portions of the same program in different parts 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 ie provision for relocation is done.
- In the overlapped Area locations physical address

  =CS1+ IP1 = CS2 + IP2 where + indicates the procedure of physical address formation

Flag Register

- 8086 has a 16-bit flag register which is divided into two parts, viz
  - Condition code or status flags
  - Machine control flags
- The condition code flag register is the lower byte of the 16bit 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

Flag Register cntd…

- 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)
  - Trap flag (T)

Flag Register cntd…

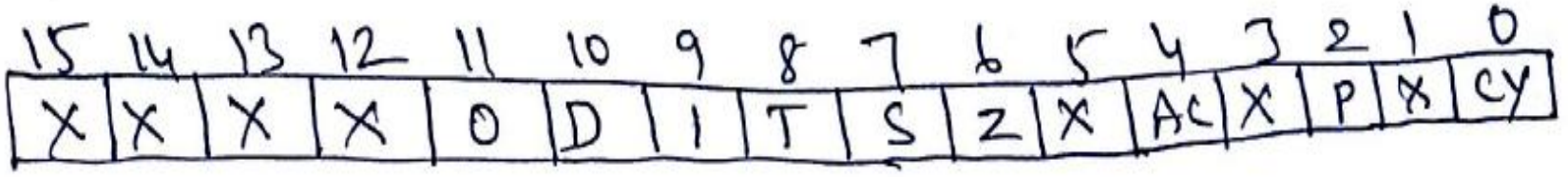


Fig 1.4 Flag register of 8086.

Flag Register cntd…

- T - Trap flag → When it is set, the processor enters the single step execution mode
- I – Interrupt Flag → If this flag is set, the maskable interrupts are recognized by the CPU, otherwise they are ignored.
- D – Direction flag → This flag is used by string manipulations instruction. If this flag bit is 0, the string is processed beginning from the lowest address to the highest address ie auto increment mode. Otherwise, the string is processed from the highest address towards the lowest address ie auto decrement mode

Flag Register cntd…

- O – Overflow flag – this flag is set if an overflow occurs ie 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 ie 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 overflow flag will be set.

Signal Descriptions of 8086

- The microprocessor 8086 is a 16 bit CPU available in 3 clock rates 5,8 and 10 MHz, packed 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

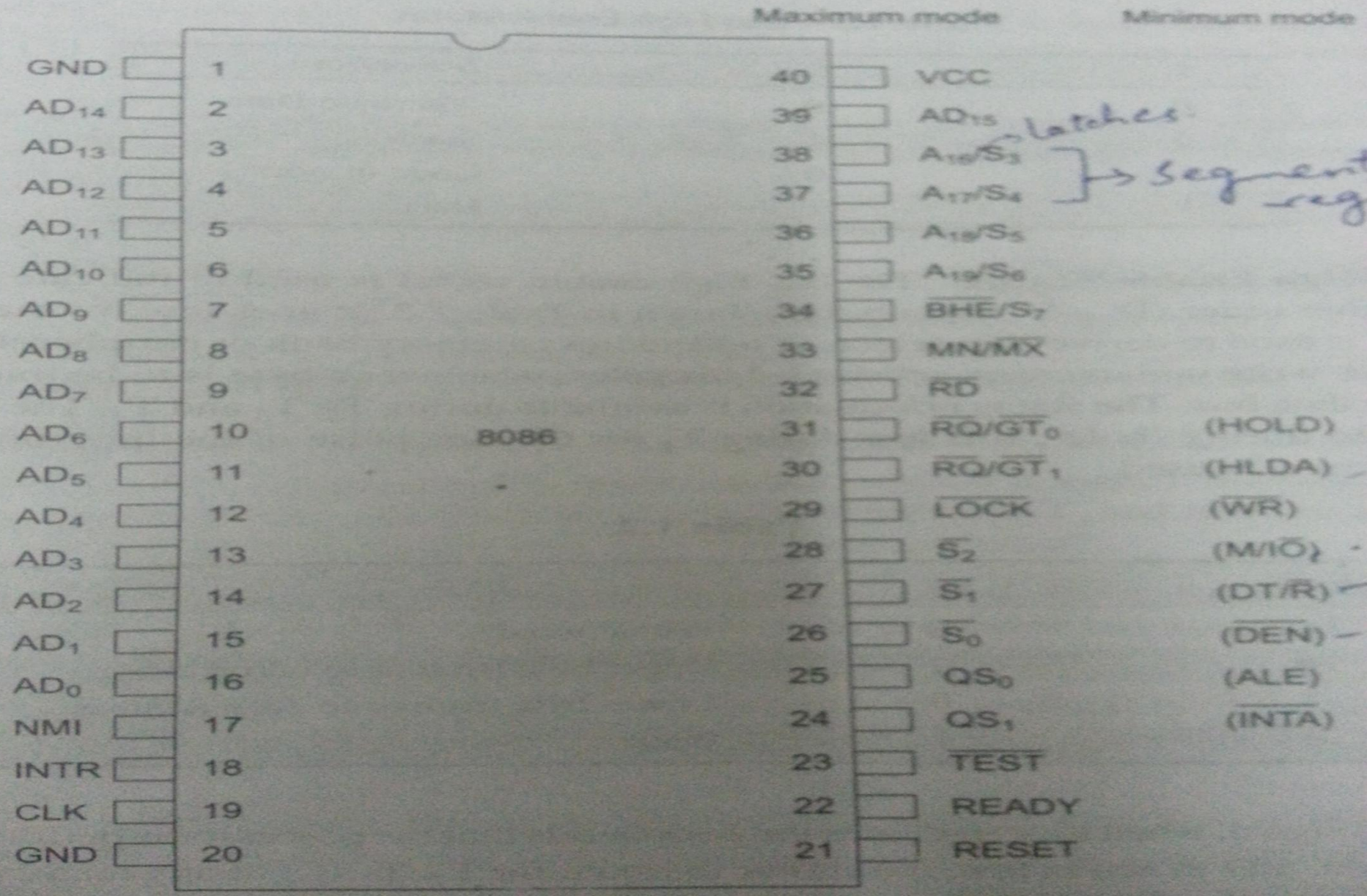


Fig. 1.5 *Pin Configuration of 8086*

Signal Descriptions of 8086 cntd…

- Some of the pins serve a particular function in minimum mode (single processor mode) and others function in maximum mode (multi processor mode) configuration
- The 8086 signals can be categorized in three groups.
  - Signals having common functions in minimum as well as maximum mode
  - Signals which have special functions for minimum mode
  - Signals which have special functions for maximum mode.

Signal Descriptions of 8086 cntd…

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

AD15 – AD0:

- These are the time multiplexed memory I/O address and data lines
- Address remains on 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.

Signal Descriptions of 8086 cntd…

- Tw is a wait state.
- These lines are active high and float to a tri-state during interrupt acknowledge and local bus hold acknowledge cycles.

A19/S6, A18/S5, A17/S4, A16/S3:

- These are the time multiplexed address and status lines.
- During T1, these are the most significant address lines for memory operations.

Signal Descriptions of 8086 cntd…

- During I/O operations, these lines are low.
- During memory or I/O operations, status information is available on those lines for T2, T3, Tw and T4.
- The status of the interrupt enable flag bit (displayed on S5) is updated at the beginning of each clock cycle.
- The S4 and S3 together indicate which segment register is presently being used for memory accesses, as shown in table 1.1.

Signal Descriptions of 8086 cntd..

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

Signal Descriptions of 8086 cntd..

S4	S3	Indications
0	0	Alternate data
0	1	Stack
1	0	Code or none
1	1	data

Signal Descriptions of 8086 cntd..

$\overline{\text{BHE}}$/S7 – Bus High Enable/Status:

- The bus high enable signal is used to indicate the transfer of data over the higher order (D15 – D8) data bus as shown in Table 1.2
- It goes low for the data transfers over D15 – D8 and is used to derive chip selects of odd address memory bank or peripherals.
- $\overline{\text{BHE}}$ 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.

Signal Descriptions of 8086 cntd..

- The status information is available during T2, T3 and T4.
- The signal is active low and is tri-stated during “Hold”.
- It is low during T1 for the first pulse of the interrupt acknowledge cycle. S7 is not currently used.

Signal Descriptions of 8086 cntd..

Table 1.2

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

Signal Descriptions of 8086 cntd..

$\overline{RD}$ – 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 T2, T3, Tw of any read cycle.
- The signal remains tri-stated during the 'HOLD Acknowledge'

Signal Descriptions of 8086 cntd..

READY

- This is the acknowledgement from the slow devices or memory that they have completed the data transfer.
- This is an input signal to the 8086.
- This 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.

Signal Descriptions of 8086 cntd..

- This can be internally masked by resetting the interrupt enable flag.
- This signal is active high.

$\overline{\text{TEST}}$

- 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.

Signal Descriptions of 8086 cntd..

NMI – Non-maskable Interrupt.

- This is an edge triggered input which causes a type 2 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.

Signal Descriptions of 8086 cntd..

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.

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 5 MHz to 10 MHz.

Signal Descriptions of 8086 cntd..

Vcc

- 8086 requires +5V power supply for the operation of the internal circuit.

GND

- This is the 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 (multi processor) mode.

Signal Descriptions of 8086 cntd..

The following pin functions are for the minimum mode operation of 8086.

$\overline{M}/\overline{IO}$ – Memory/IO

- This is a status line logically equivalent to $\overline{S2}$ in the maximum mode.
- Low – I/O operation

  High – Memory operation.
- This line becomes active in the previous T4 and remains active till final T4 of the current cycle.
- It is tri-stated during local bus “hold acknowledge”

Signal Descriptions of 8086 cntd..

$\overline{\text{INTA}}$ – 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 T2,T3 and Tw of each interrupt acknowledge cycle.

ALE – Address Latch Enable

- This output signal indicates that 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 tri-stated.

Signal Descriptions of 8086 cntd..

DT/$\overline{R}$ – Data Transmit / Receive

- This output is used to decide the direction of data flow through the transreceivers (bidirectional buffers).
- Data transmission – signal is high.
  data receiving – signal is low.
- Logically, this is equivalent to S1 in maximum mode.
- It's timing is the same as M/$\overline{I/O}$.
- This is tri-stated during 'hold acknowledge'

Signal Descriptions of 8086 cntd..

$\overline{\text{DEN}}$ – Data Enable

- This signal indicates the availability of valid data over the address / data lines.
- It is used to enable the transreceivers to separate the data from the multiplexed address / data signal.
- It is active from the middle of T2 until the middle of T4.
- $\overline{\text{DEN}}$ is tri-stated during hold acknowledge cycle

Signal Descriptions of 8086 cntd..

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 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.

Signal Descriptions of 8086 cntd..

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

S2, S1, S0 – status lines

- These are the status lines which indicate 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.
- The various operations indicated by these status lines are given in the table 1.3.

Signal Descriptions of 8086 cntd..

Table 1.3

S2	S1	S0	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

Signal Descriptions of 8086 cntd..

$\overline{\text{LOCK}}$

- This output pin indicates that other system bus masters will be prevented from gaining the system bus, while LOCK signal is low.
- The LOCK signal is activated by the ‘$\overline{\text{LOCK}}$’ prefix instruction and remains active until the completion of the next instruction.
- This floats to tri-state off during ‘hold acknowledge’
- When CPU is executing a critical instruction which requires the system bus, the LOCK prefix instruction ensure that other processors connected in the system will not gain the control of the bus.

Signal Descriptions of 8086 cntd..

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 lines indicate various operations as indicated in the table 1.4.

Signal Descriptions of 8086 cntd..

Table 1.4

QS1	QS0	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 simultaneous fetching and executing of instructions is called pipe-lining. This results in faster execution.

Signal Descriptions of 8086 cntd..

$\overline{RQ}/\overline{GT0}$, $\overline{RQ}/\overline{GT1}$ – REQUEST/GRANT

- These pins are used by other local bus masters in maximum mode, to force the processor to release local bus at the end of processor's current bus cycle.
- Each of the pins is bidirectional with $\overline{RQ}/GT0$ having the higher priority than $\overline{RQ}/GT1$.
- The request and grant pulses are active low.
- $\overline{RQ}/\overline{GT}$ pins have internal pull up resistors and may be left unconnected.

Signal Descriptions of 8086 cntd..

- The request/grant sequence is as follows.

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

Signal Descriptions of 8086 cntd..

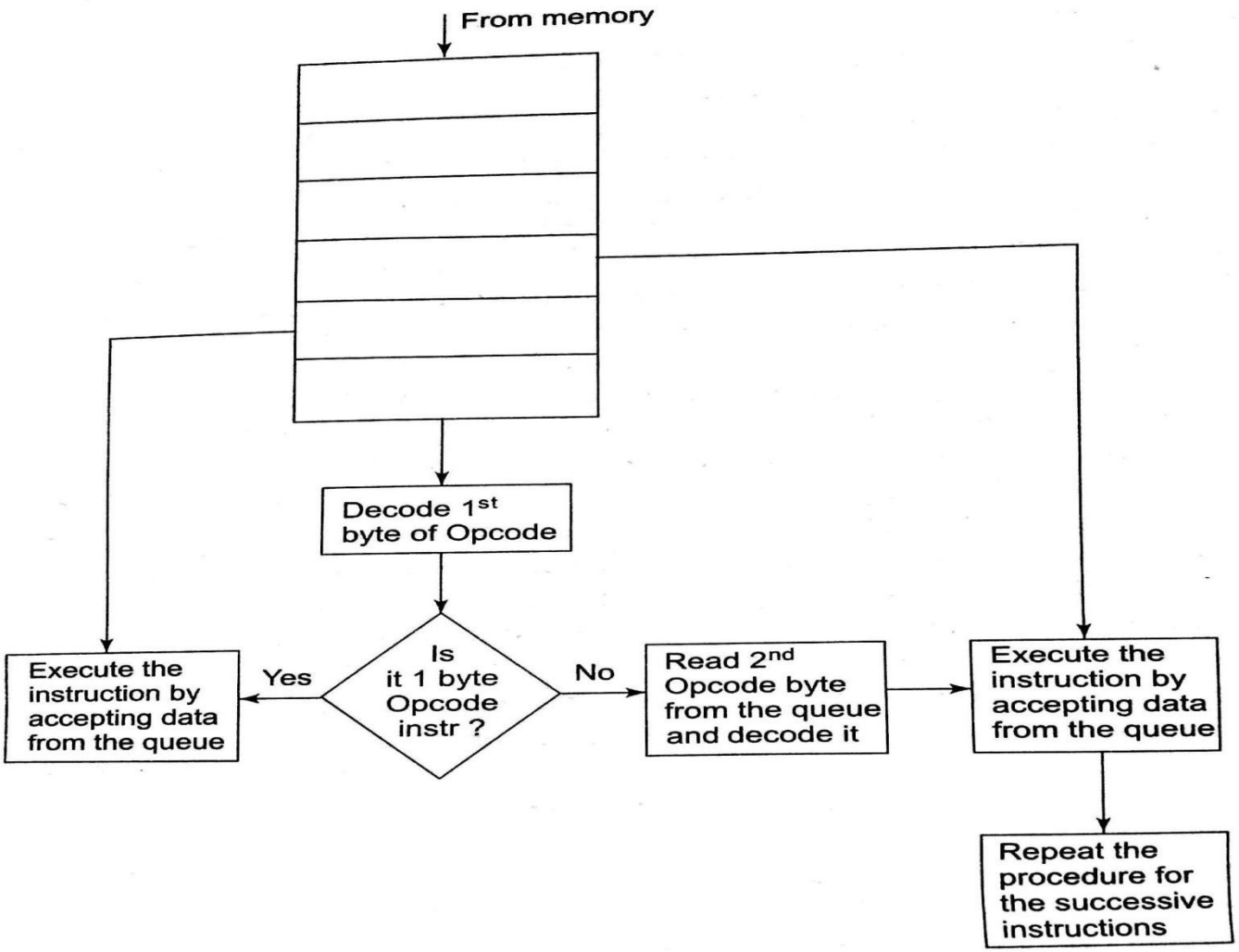


Fig. 1.6 *The Queue Operation*

Thus, each master to master exchange of the local bus is a sequence of 3 pulses. There mu

Physical Memory Organization

- In 8086, 1Mbyte memory is physically organized as an odd bank and even bank, each of 512 Kbytes.
- Data byte with an even address is transferred on D7 – D0 while the data byte with an odd address is transferred on D15 – D8 bus lines.
- If instructions are fetched from memory as words, different possibilities of these words are
  - 1 Both the bytes may be data operands
  - 2 Both the bytes may contain op-code bits
  - 3 one of the bytes may be op-code while other may be data

Physical Memory Organization cntd..

- The above possibilities are taken care by internal decoder which creates signals and sends those to Timing & Control Unit.
- This Timing and Control Unit generates the signals which executes the instructions.
- While referring to word data, BIU requires one or two memory cycles
  - Starting byte at even address ------- One cycle
  - Starting byte at odd address ------- two cycles

Physical Memory Organization cntd..

- Hence while initializing the stack, it should be initialized at an even address for efficient operation.
- A map of an 8086 memory system starts at 00000H and ends at FFFFFH.

Physical Memory Organization cntd..

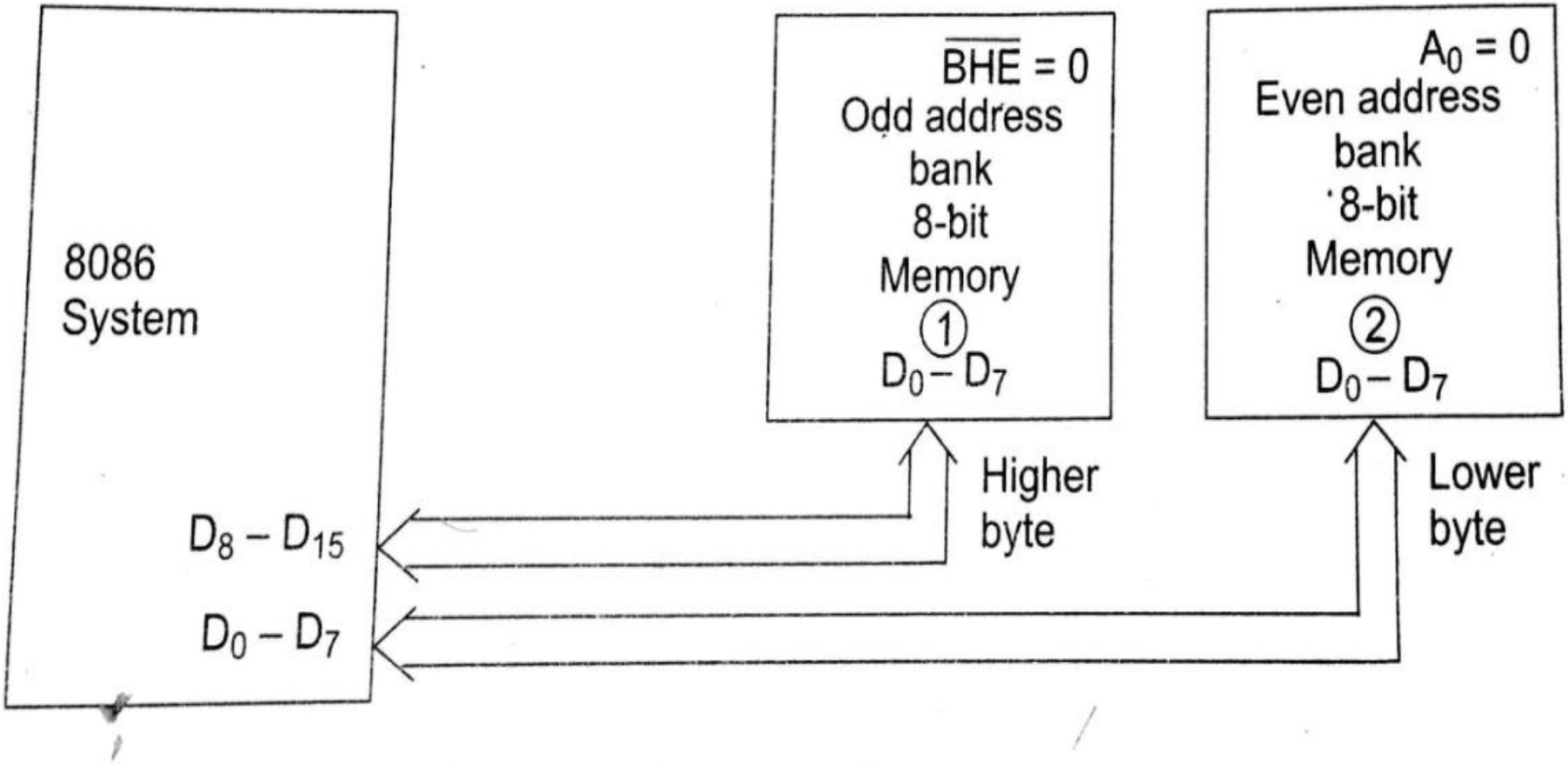


Fig. 1.7 *Physical Memory Organization*

Physical Memory Organization cntd..

- For accessing 16 bit data, both banks are to be accessed where as to access only 8 bit data, only one bank is accessed.
- Selection of memory banks depends on $\overline{BHE}$ and A0.

$\overline{BHE}$	A0	Indication
0	0	Whole word
0	1	Upper byte from or to odd address
1	0	Lower byte from or to even address
1	1	None

Minimum Mode 8086 System and Timings

- In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping it's MN/$\overline{\text{MX}}$ 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.

Minimum Mode 8086 System and Timings cntd..

- Some type of chip selection logic may be required for selecting memory or I/O devices, depending upon the address map of the system.
- 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.

Minimum Mode 8086 System and Timings cntd..

- 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 ie from or to the processor.

Minimum Mode 8086 System and Timings cntd..

- The system contains memory for the monitor and user 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 (IC 8284) 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.

Minimum Mode 8086 System and Timings cntd..

- The clock generators also synchronizes some external signals with the system clock
- The general system organization is shown in Figure 1.13.
- Since it has 20 address lines and 16 data lines the 8086 CPU requires 3 octal address latches and two octal data buffers for the complete address and the 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 operation.

Minimum Mode 8086 System and Timings cntd..

- The op-code 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 second is the timing for write cycle
- The read cycle begins in T1 with the assertion of the address latch enable (ALE) signal and M/IO signal
- During the negative going edge of this signal, the valid address is latched on the local bus

Minimum Mode 8086 System and Timings cntd..

- The $\overline{BHE}$ and A0 signals address low, high or both bytes
- From T1 to T4, the M/IO signal indicates a memory or I/O operation
- At T2, the address is removed from the local bus and is sent to the output. The bus is then tri-stated.
- The read ($\overline{RD}$) control signal is also activated in T2
- 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.

Minimum Mode 8086 System and Timings cntd..

- The addressed device will drive the READY line high.
- When the processor returns the read signal to high level, the addressed device will again tri-state its bus drivers. CS logic indicates chip select logic and ‘e’ and ‘o’ suffixes indicate even and odd address memory bank
- A write cycle also begins with the assertion of ALE and the emission of the address.

Minimum Mode 8086 System and Timings cntd..

- The M/$\overline{IO}$ signal is again asserted to indicate a memory or I/O operation.
- In T2, after sending the address in T1, the processor sends the data to be written to the addressed location.
- The data remains on the bus until the middle of T4 state.
- The $\overline{WR}$ becomes active at the beginning of T2 (unlike $\overline{RD}$ is some what delayed in T2 to provide time for floating.)

Minimum Mode 8086 System and Timings cntd..

- The $\overline{BHE}$ and A0 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

Minimum Mode 8086 System and Timings cntd..

M / IO	RD	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

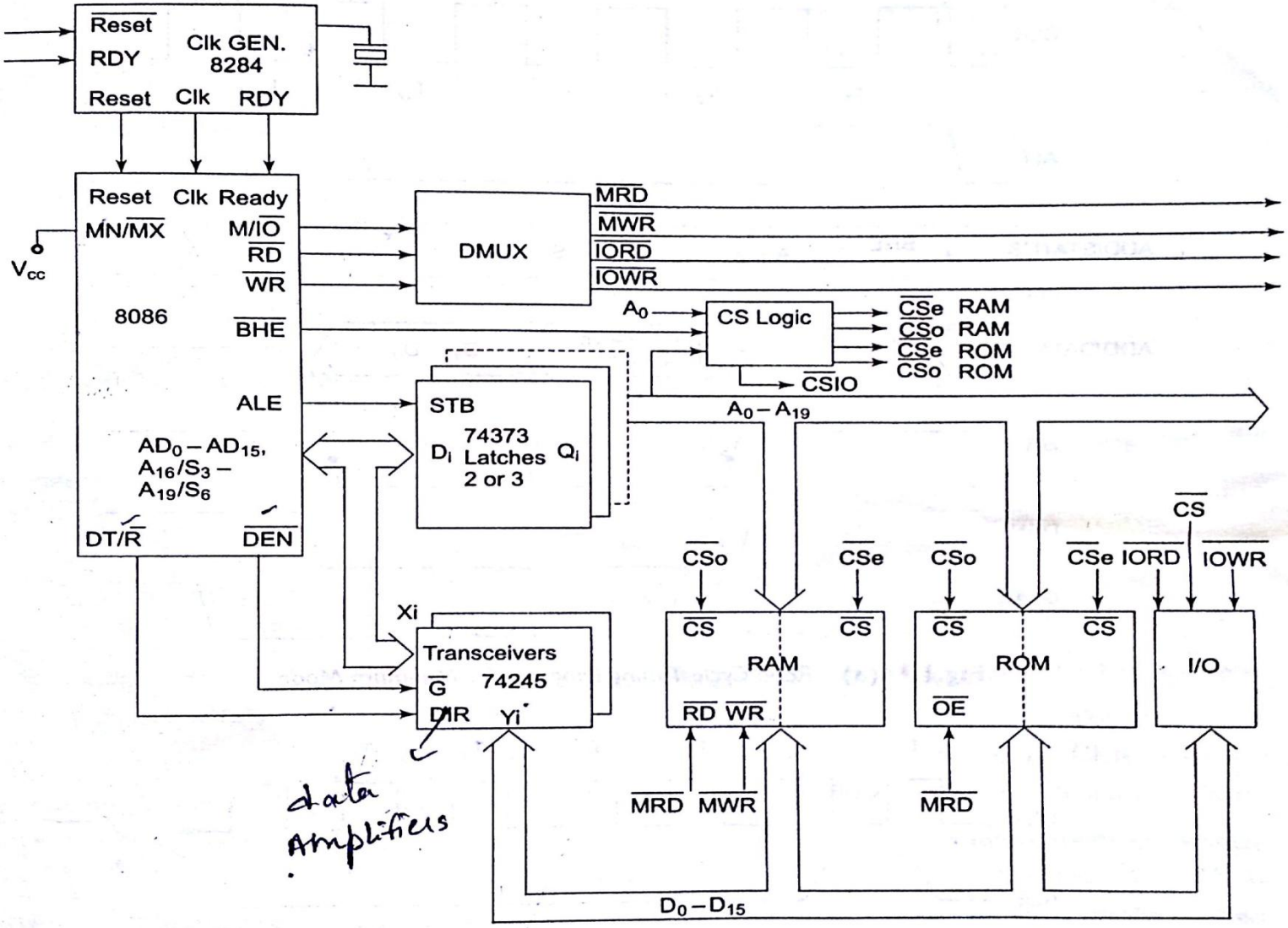


Fig. 1.13 *Minimum Mode 8086 System*

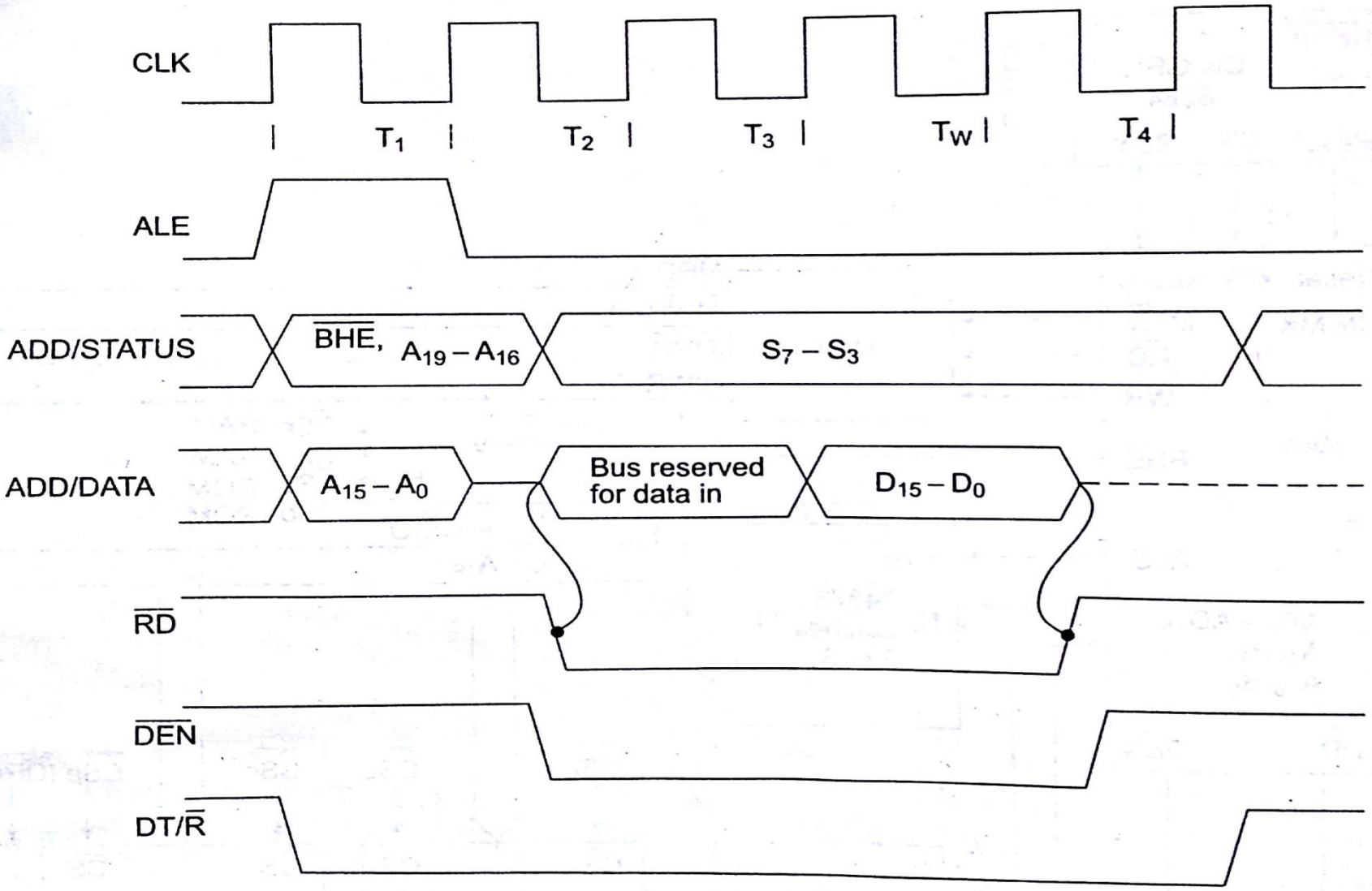


Fig.1.14(a) *Read Cycle Timing Diagram for Minimum Mode*

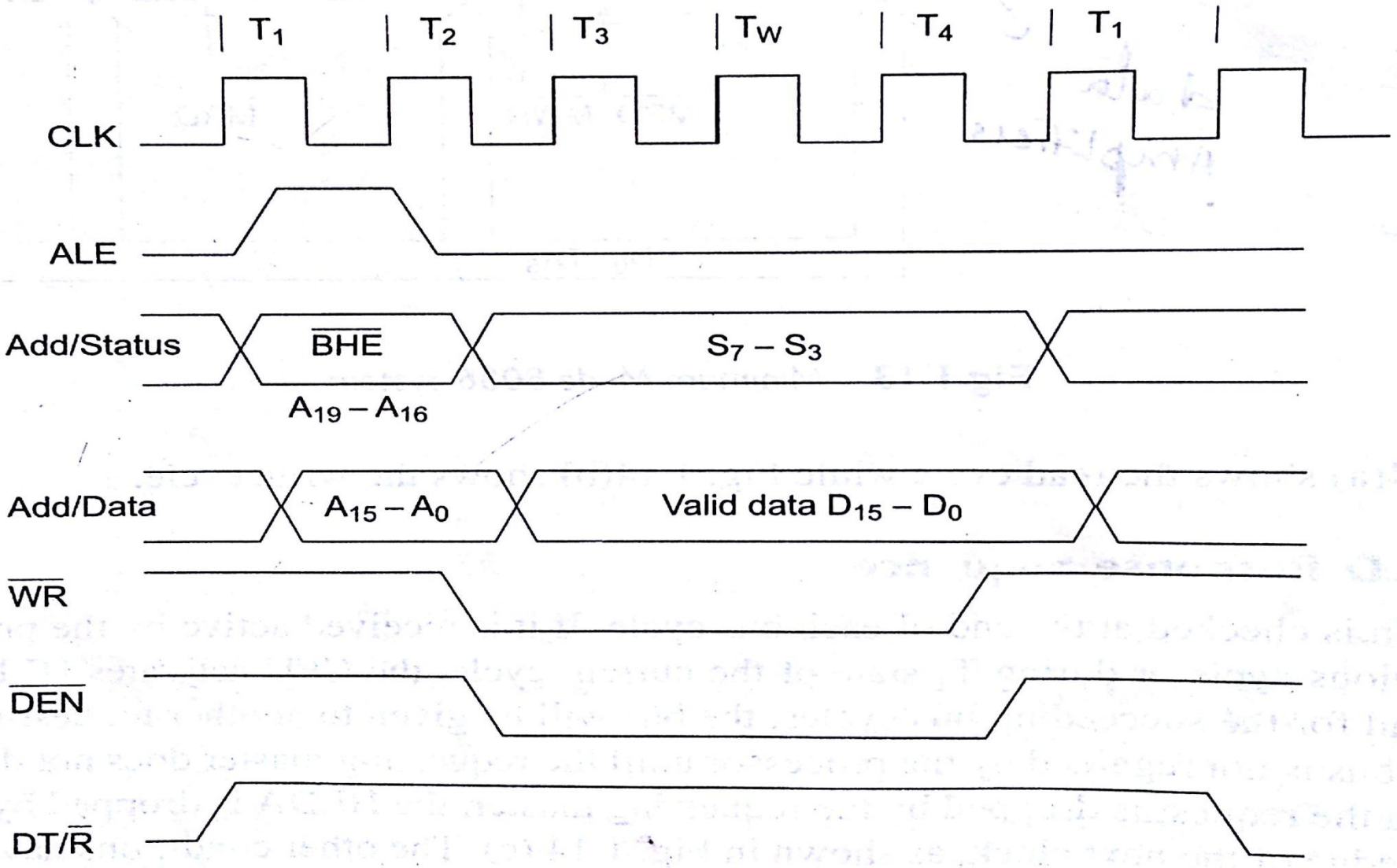


Fig. 1.14(b) *Write Cycle Timing Diagram for Minimum Mode Operation*

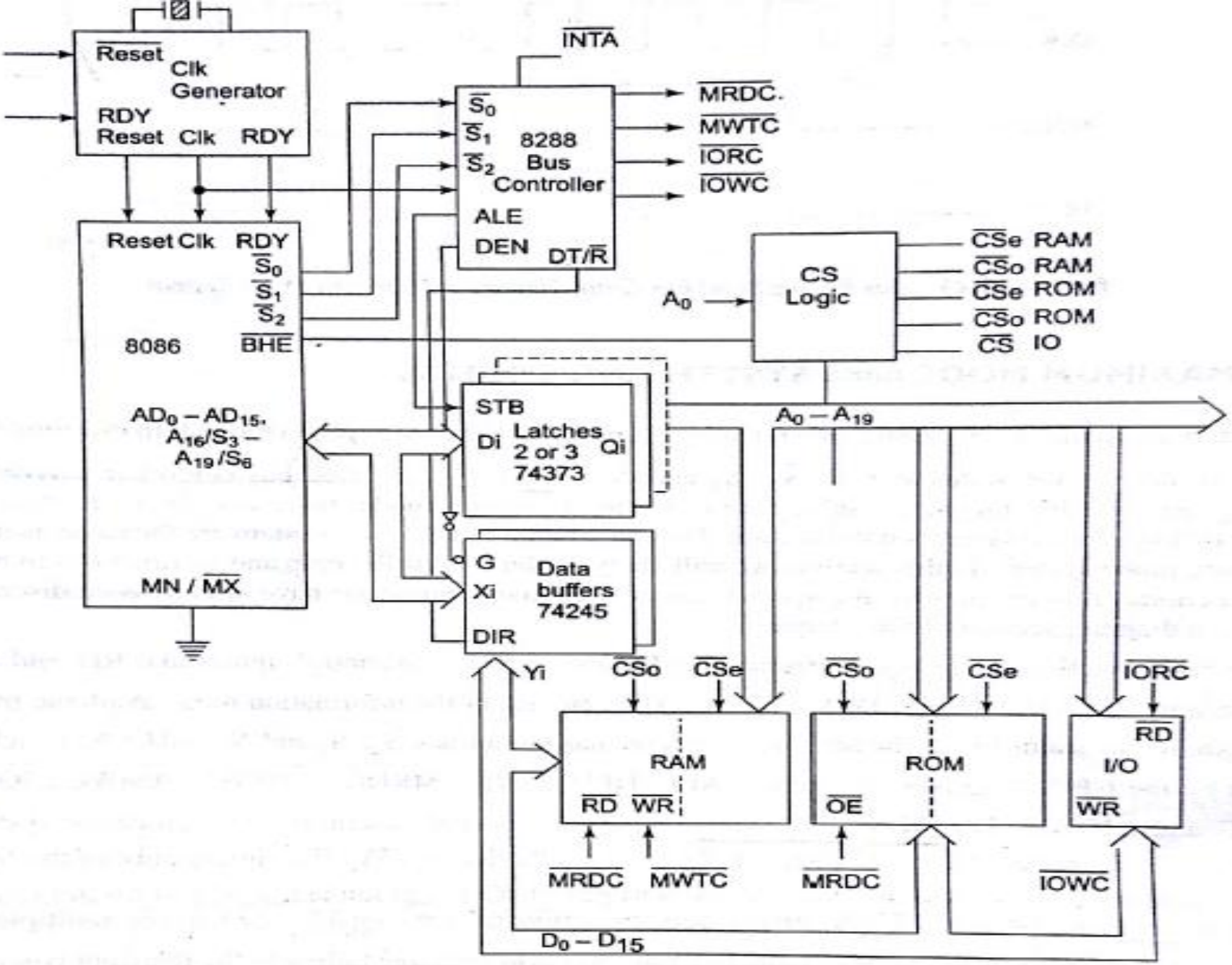


Fig. 1.15 *Maximum Mode 8086 System*

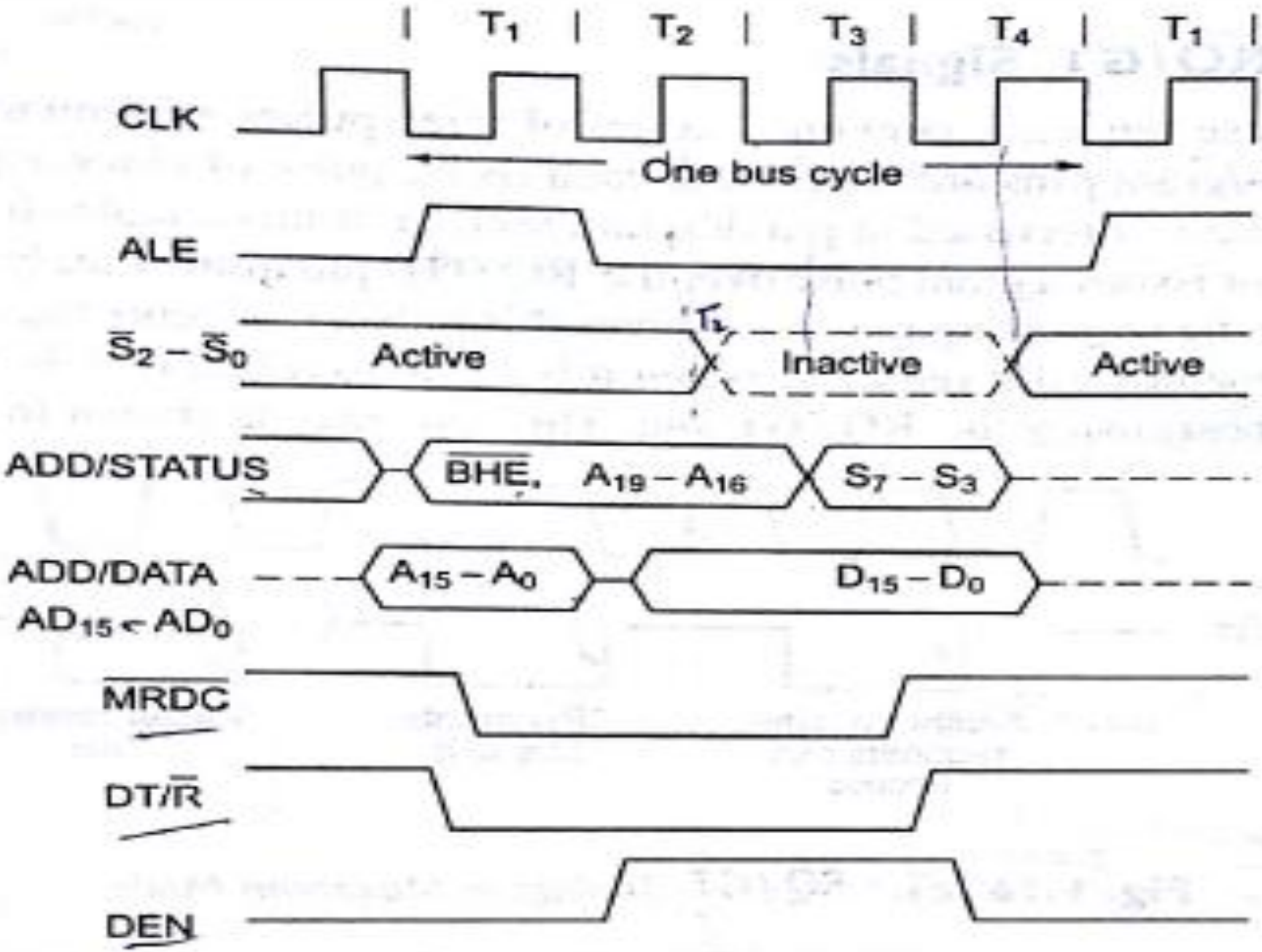


Fig. 1.16 (a) *Memory Read Timing in Maximum Mode*

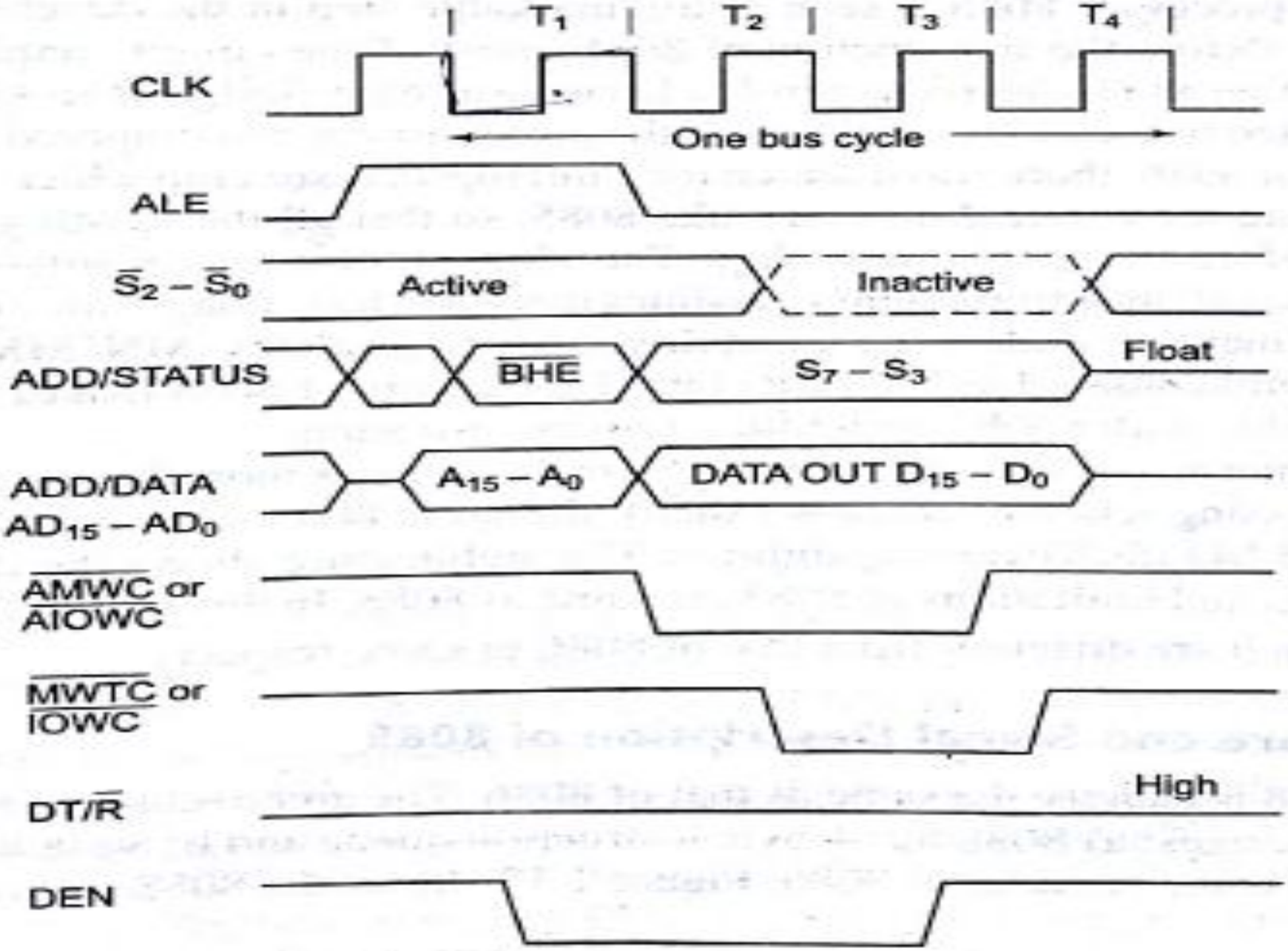


Fig. 1.16(b) *Memory Write Timing in Maximum Mode*

END of Unit I

UNIT II

Instruction Set & Assembly Language Programming of 8086

Machine Language Instruction Formats

- A machine language instruction format has one or more number of fields associated with it.
- The first field is called as operation code field or op-code field, which indicates the type of operation to be performed by the CPU
- The instruction format also contains other fields known as operand fields
- The CPU executes the instruction using the information which reside in these fields
- There are six general formats of instructions in 8086 instruction set.
- The length of an instruction may vary from 1 byte to 6 bytes. The instruction formats are described as follows

Machine Language Instruction Formats cntd..

1 One Byte Instruction:

- This format is only one byte long and may have the implied data or register operands.
- The least significant 3-bits of the opcode are used for specifying the register operand, if any.
- Otherwise, all the 8 bits form an opcode and the operands are implied

Machine Language Instruction Formats cntd..

2 Register to Register:

- This format is 2 bytes long
- The first byte of the code specifies the operation code and width of the operand specified by ‘w’ bit.
- The second byte of the code shows the register operands and R/M field, as shown below.
-

D7 D1	D0
OPCODE	W

D7 D6	D5 D4 D3	D2 D1 D0
11	REG	R/M

Machine Language Instruction Formats cntd..

- The register represented by the REG field is one of the operands.
- The R/M field specifies another register or memory location i.e. the other operand.

3 Register to/from memory with no displacement:

- This format is also 2 bytes long and similar to the Register to Register format except for the MOD field as shown.

D7 D1	D0
OPCODE	W

D7 D6	D5 D4 D3	D2 D1 D0
MOD	REG	R/M

Machine Language Instruction Formats cntd..

- The MOD field shows the mode of addressing. The MOD, R/M, REG and the ‘W’ fields are decided in Table 2.2.

Table 2.2 *Addressing Modes and the Corresponding MOD, REG and R/M Fields*

Operands	*Memory Operands*			*Register Operands*	
	No Displacement	*Displacement 8-bit*	*Displacement 16-bit*		
MOD / *R/M*	*00*	*01*	*10*	*11*	
				W = 0	*W = 1*
000	(BX) + (SI)	(BX) + (SI) + D8	(BX) + (SI) + D16	AL	AX
001	(BX) + (DI)	(BX) + (DI) + D8	(BX) + (DI) + D16	CL	CX
010	(BP) + (SI)	(BP) + (SI) + D8	(BP) + (SI) + D16	DL	DX
011	(BP) + (DI)	(BP) + (DI) + D8	(BP) + (DI) + D16	BL	BX
100	(SI)	(SI) + D8	(SI) + D16	AH	SP
101	(DI)	(DI) + D8	(DI) + D16	CH	BP
110	D16	(BP) + D8	(BP) + D16	DH	SI
111	(BX)	(BX) + D8	(BX) + D16	BH	DI

Note: 1. D8 and D16 represent 8 and 16 bit displacements respectively.

2. The default segment for the addressing modes using BP and SP is SS. For all other addressing modes the default segments are DS or ES.

Machine Language Instruction Formats cntd..

4 Register to/from Memory with Displacement:

- This type of instruction format contains 1 or 2 additional bytes for displacement along with 2 byte format of the register to/from memory without displacement. The format is as shown below.

D7 D1	D0
OPCODE	W

D7 D6	D5 D4 D3	D2 D1 D0
MOD	REG	R/M

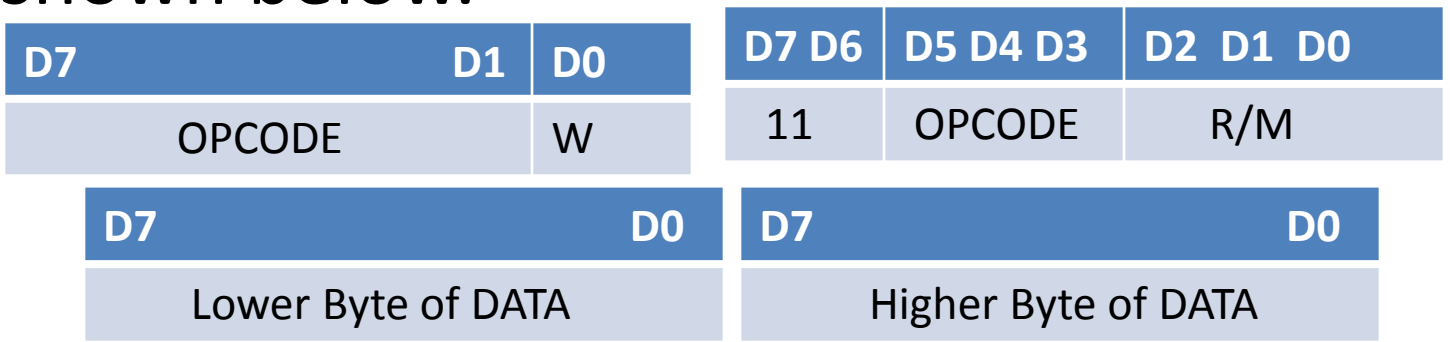
D7 D0
Lower Byte of Displacement

D7 D0
Higher Byte of Displacement

Machine Language Instruction Formats cntd..

5 Immediate Operand to Register:

- In this format, the first byte as well as the 3-bits from the second byte which are used for REG field in case of register to register format are used for opcode.
- It also contains one or two bytes of immediate data. The complete instruction format is as shown below.



Machine Language Instruction Formats cntd..

6 Immediate Operand to Memory with 16-bit displacement:

- This type of instruction format requires 5 or 6 bytes for coding.
- The first 2 bytes contain the information regarding OPCODE, MOD and R/M fields. The remaining 4 bytes contain 2 bytes of displacement and 2 bytes of data as shown.

D7	D1	D0
OPCODE		W

D7 D6	D5 D4 D3	D2 D1 D0
MOD	OPCODE	R/M

D7	D0
Lower Byte of DISPLACEMENT	

D7	D0
Higher Byte of DISPLACEMENT	

D7	D0
Lower Byte of DATA	

D7	D0
Higher Byte of DATA	

Addressing Modes of 8086

- Addressing mode indicates a way of locating data or operands.
- Depending upon the data types used in the instruction and the memory addressing modes, any instruction may belong to one or more addressing modes, or some instruction may not belong to any of the addressing modes
- Thus addressing modes describe the types of operands and the way they are accessed for executing an instruction.
- According to the flow of instruction execution, the instructions may be categorized as
  - Sequential control flow instructions
  - Control transfer instructions

Addressing Modes of 8086 cntd..

- Sequential control flow instructions are the instructions which after execution, transfer control to the next instruction appearing immediately after it in the program.
- For example, the arithmetic, logical, data transfer and processor control instructions are sequential control flow instructions.
- The control transfer instructions, on the other hand , transfer control to some predefined address or the address somehow specified in the instruction, after their execution.
- For example INT, CALL, RET and JUMP instructions fall under this category
- The addressing modes for sequential and control transfer instructions are explained as follows.

Addressing Modes of 8086 cntd..

1 Immediate:

- In this type of addressing, immediate data is a part of instruction, and appears in the form of successive byte or bytes
- Eg: MOV AX, 0005H

2 Direct:

- In the direct addressing mode, a 16-bit memory address (offset) is directly specified in the instruction as a part of it.
- Eg: MOV AX,[5000H],
  - Effective address= 10H*DS +5000H

Addressing Modes of 8086 cntd..

3 Register:

- In the register addressing mode, the data is stored in a register and it is referred using the particular register
- All the registers, except IP, may be used in this mode.
- Eg: MOV AX, BX

Addressing Modes of 8086 cntd..

4 Register Indirect:

- Some times, the address of the memory location which contains data or operand is determined in an indirect way, using the offset registers.
- This mode of addressing is known as register indirect mode
- In this addressing mode, the offset address of data is in either BX or SI or DI register.
- The default segment is either DS or ES. The data is supposed to be available at the address pointed to by the content of any of the above registers in the default data segment.
- Eg: MOV AX,[BX]
  - Effective address is 10H*DS+[BX]

Addressing Modes of 8086 cntd..

5 Indexed:

- In this addressing mode, offset of the operand is stored in one of the Index registers.
- DS is the default segment for index registers SI and DI
- In the case of string instructions DS and ES are default segments for SI and DI respectively.
- This mode is a special case of the above discussed register indirect addressing mode
- Eg: MOV AX,[SI]
  - effective address is 10H*DS+[SI]

Addressing Modes of 8086 cntd..

6 Register Relative:

- In this addressing mode, the data is available at an effective address formed by adding an 8-bit or 16-bit displacement with the content of any one of the registers BX, BP, SI and DI in the default (either DS or ES) segment.
- Eg: MOV AX,50H[BX]
  - Effective address is 10H*DS+50H+[BX]

Addressing Modes of 8086 cntd..

7 Based Indexed:

- The effective address of the data is formed, in this addressing mode, by adding the content of a base register (any one of BX or BP) to the content of an index register (any one of SI or DI)
- The default segment register may be DS or ES
  - Eg: MOV AX,[BX][SI]
    - effective address is 10H*DS +[BX]+[SI]

Addressing Modes of 8086 cntd..

8 Relative Based Indexed:

- The effective address is formed by adding an 8-bit or 16-bit displacement with the sum of contents of anyone of the base registers (BX or BP) and any one of the index registers (SI or DI), in a default segment.
- Eg: MOV AX,50H[BX][SI]
  - Effective address is 10H* DS+[BX]+[SI]+50H

Addressing Modes of 8086 cntd..

- For the control transfer instructions, the addressing modes depend upon whether the destination location is within the same segment or in a different one.
- It also depends upon the method of passing the destination address to the processor.
- Basically there are two addressing modes for the control transfer instructions, viz, intersegment and intrasegment addressing modes.
- If the location to which the control is to be transferred lies in a different segment other than the current one, the mode is called intersegment mode.

Addressing Modes of 8086 cntd..

- If the destination location lies in the same segment, the mode is called intrasegment mode

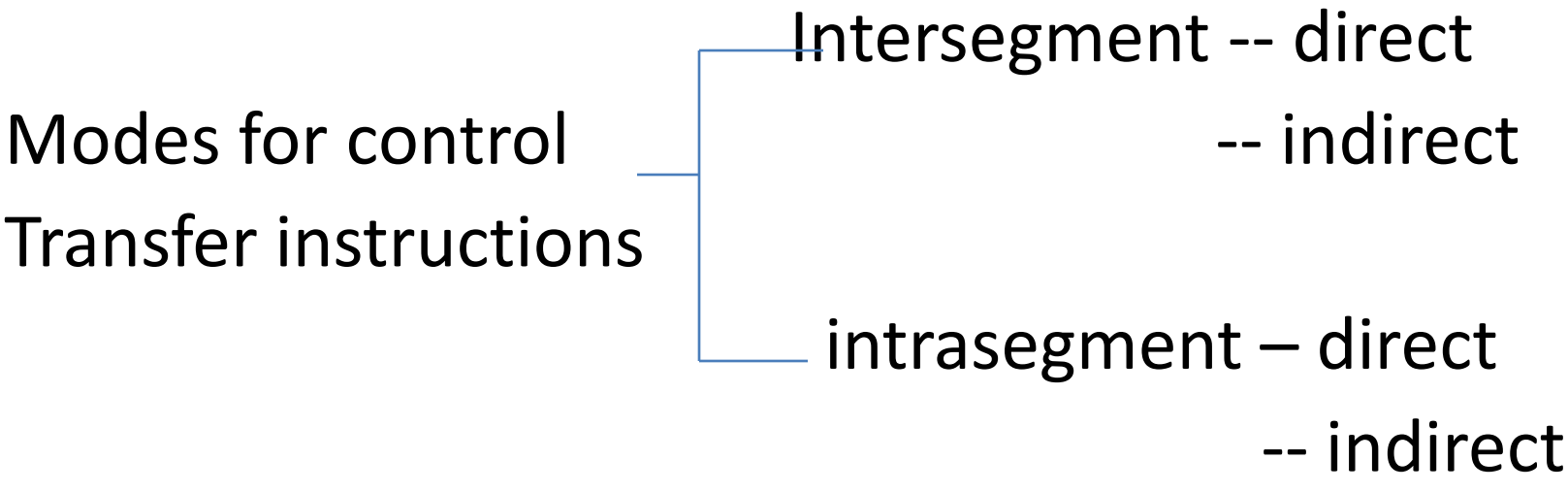


Fig 2.1 Addressing modes for control transfer instructions

Addressing Modes of 8086 cntd..

9 Intrasegment Direct mode:

- In this mode, the address to which the control is to be transferred lies in the same segment in which the control transfer instruction lies and appears directly in the instruction as an immediate displacement values.
- In this addressing mode, the displacement is computed relative to the content of the instruction pointer IP.
- The effective address to which the control will be transferred is given by the sum of 8 or 16 bit displacement and current content of IP.

Addressing Modes of 8086 cntd..

- In case of Jump instruction, if the signed displacement (d) is of 8 bits (ie -128 <d< +127) we term it as short jump and if it is of 16bits (ie -32768<d< +32767), it is termed as long jump.
- Eg: JMP SHORT LABEL; LABEL lies with in -128 to +127 from the current IP content. Thus SHORT LABEL is 8-bit signed displacement.

Addressing Modes of 8086 cntd..

10 Intrasegment Indirect Mode:

- In this mode, the displacement to which the control is to be transferred, is in the same segment in which the control transfer instruction lies, but it is passed to the instruction indirectly.
- Here, the branch address is found as the content of a register or a memory location.
- This addressing mode may be used in unconditional branch instructions.
- Eg: JMP [BX] here the effective address is stored in BX

Addressing Modes of 8086 cntd..

11 Intersegment Direct:

- In this mode, the address to which the control is to be transferred is in a different segment.
- This addressing mode provides a means of branching from one code segment to another code segment.
- Here the CS an IP of the destination address are specified directly in the instruction.
- Eg: JMP 5000H:2000H
  Jump to effective address 2000H in segment 5000H

Addressing Modes of 8086 cntd..

12 Intersegment Indirect:

- In this mode, the address to which the control is to be transferred lies in a different segment and it is passed to the instruction indirectly ie contents of a memory block containing 4 bytes ie IP(LSB), IP(MSB), CS(LSB) and CS(MSB) sequentially
- The starting address of the memory block may be referred using any of the addressing modes, except immediate mode.
- Eg: JMP [2000H] Jump to an address in other segment specified at effective address 2000H in DS, that points to the memory block as said above.

Instruction Set of 8086

Programs – Arithmetic

1) Write an assembly language program to perform the ADDition of two 8 bit numbers using 8086.

Solution:

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
NUM1 DB 12
NUM2 DB 18
SUM  DB ?
DATA ENDS
CODE SEGMENT
START: MOV AX, DATA
        MOV DS,AX
        MOV AL, NUM1
        ADD AL, NUM2
        MOV SUM, AL
        INT 03H
CODE ENDS
END START
END.
```

Programs – Arithmetic cntd..

2) Write an ALP for ADDition of two 16 bit numbers using 8086

Solution:

```
ASSUME CS: CODE, DS: DATA
  DATA SEGMENT
  NUM1 DW 1234
  NUM2 DW 4567
  SUM  DW ?
  DATA ENDS
  CODE SEGMENT
  START: MOV AX, DATA
         MOV DS,AX
         MOV AX, NUM1
         ADD AX, NUM2
         MOV SUM, AX
         INT 03H
  CODE ENDS
  END START
  END.
```

Programs – Arithmetic cntd..

3) Write an assembly language program to perform the SUBtract operation of two 8 bit numbers using 8086.

Solution:

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
NUM1 DB 18
NUM2 DB 12
DIFF   DB ?
DATA ENDS
CODE SEGMENT
START: MOV AX, DATA
        MOV DS,AX
        MOV AL, NUM1
        SUB AL, NUM2
        MOV DIFF, AL
        INT 03H
CODE ENDS
END START
END.
```

Programs – Arithmetic cntd..

4) Write an assembly language program to perform the SUBtract operation of two 16 bit numbers using 8086.

Solution:

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
NUM1 DW 1835
NUM2 DW 1735
DIFF    DW ?
DATA ENDS
CODE SEGMENT
START: MOV AX, DATA
       MOV DS,AX
       MOV AX, NUM1
       SUB AX, NUM2
       MOV DIFF, AX
       INT 03H
CODE ENDS
END START
END.
```

Programs – Logical

1) Write an assembly language program to perform the AND operation between two 16 bit numbers using 8086.

Solution:

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
NUM1 DW 1234
NUM2 DW 1856
ANDRES DW ?
DATA ENDS
CODE SEGMENT
START: MOV AX, DATA
       MOV DS,AX
       MOV AX, NUM1
       AND AX, NUM2
       MOV ANDRES, AX
       INT 03H
CODE ENDS
END START
END.
```

Programs – Logical cntd..

2) Write an assembly language program to perform the OR operation of two 16 bit numbers using 8086.

Solution:

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
NUM1 DW 1234
NUM2 DW 1898
ORRES DW ?
DATA ENDS
CODE SEGMENT
START: MOV AX, DATA
       MOV DS,AX
       MOV AX NUM1
       OR AX, NUM2
       MOV ORRES, AX
       INT 03H
CODE ENDS
END START
END.
```

Programs - Branch

1) Write an ALP to find out the larger number between two 16 bit numbers using Jump instruction using 8086.

Solution:

```
ASSUME CS:CODE, DS:DATA
DATA SEGMENT
NUM1 DW1534
NUM2 DW 2078
LARGENUM DW?
DATA ENDS
CODE SEGMENT
START: MOV AX,DATA
        MOV DS,AX
        MOV AX,NUM1
        CMP AX,NUM2
        JC  GO
        MOV LARGENUM,AX
        JMP EXIT
GO:     MOV LARGENUM,NUM2
EXIT:   INT 03H
CODE ENDS
END START
END
```

Programs – Branch cntd..

2) Write an ALP to sort the given array of 16 bit numbers using 8086.

Solution:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
ARRAYNUM DW 2378H,4567H,3498H,1289H,4298H
COUNT EQU 05H
DATA ENDS
CODE SEGMENT
START:  MOV AX,DATA
        MOV DS,AX
        MOV DX, COUNT -1
NXTITR:MOV CX,DX
        MOV SI, OFFSET ARRAYNUM
 AGAIN:MOV AX,[SI]
        CMP AX,[SI+2]
        JC  GO
        XCHG AX,[SI+2]
        XCHG AX.[SI]
GO:     INC SI
        INC SI
        LOOP AGAIN
        DEC DX
        JNZ NXTITR
        INT 03H
        CODE ENDS
        END START
        END
```

Programs – Call cntd..

Call Instruction: This instruction is used to call procedures or Subroutines in the main program.

Procedure or Subroutine: it is a group of instructions, which is called in the main program, to execute a common functionality many times.

How to pass parameters to a Procedure: There are many ways to pass parameters to a Procedure. Those are

1) Using Global declared variable
2) Using registers of CPU architecture
3) Using memory locations (reserved)
4) Using a Stack
5) Using PUBLIC and EXTRN.

Programs – Call cntd..

Example 1: Using Global declared variable

```
ASSUME CS:CODE1,DS:DATA
DATA SEGMENT
NUMBER EUQ 77H GLOBAL
DATA ENDS
CODE SEGMENT
START: MOV AX,DATA
        MOV DS,AX
              .
              .
        MOV AX,NUMBER
              .
        CODE1 ENDS
ASSUME CS:CODE2
CODE2 SEGMENT
        MOV AX,DATA
        MOV DS,AX
        MOV BX,NUMBER
CODE2 ENDS
         END START
         END
```

Programs – Call cntd..

Example 2: Using registers of CPU architecture

```
ASSUME CS:CODE
CODE SEGMENT
 START: MOV AX,5555H
        MOV BX,7272H
               .
               .
        CALL PROCEDURE1
               .
               .
               .
PROCEDURE PROCEDURE 1 NEAR
               .
               .
         ADD AX,BX
               .
               .
         RET
PROCEDURE1 ENDP
CODE ENDS
END START
```

Programs – Call cntd..

- Here the CPU registers, used in the main program, are modified in the Procedure as the same CPU registers are used here. Hence to avoid this data loss, the register contents are pushed on to a stack to save them.
- At the end of the subroutine, this saved data will be pop up from the stack and then return to the main program so that there won't be any data loss.

Programs – Call cntd..

Example 3: Using memory locations

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
NUM DB (55H)
COUNT EQU 10H
DATA ENDS
CODE SEGMENT
START:  MOV AX,DATA
        MOV DS,AX
            .
            .
        CALL ROUTINE
            .
            .
PROCEDURE ROUTINE NEAR
        MOV BX,NUM
        MOV CX,COUNT
            .
ROUTINE ENDP
CODE ENDS
END START
END
```

Programs – Call cntd..

Example4: Using Stack

```
ASSUME CS:CODE,SS:STACK
CODE SEGMENT
START:   MOV AX,STACK
         MOV SS,AX
         MOV AX,5577H
         MOV BX,2929H
              .
         PUSH AX
         PUSH BX
         CALL ROUTINE  ; decrements sp by 2 (by 4 far routine)
              .
              .
PROCEDURE  ROUTINE NEAR
              .
              .
         MOV DX,SP
         ADD SP,02
```

Programs – Call cntd..

```
    POP BX
    POP AX
    MOV SP,DX
        .
        .
        .
STACK SEGMENT
STACKDATA DB 200H DUP (?)
STACK ENDS
```

Programs – Call cntd..

Using PUBLIC & EXTRN:

- For passing parameters to procedures using the PUBLIC & EXTRN directives , the data must be declared PUBLIC (for all routines) in the main routine and the same should be declared EXTRN in the procedure.
- Thus, the main program can pass the PUBLIC parameter to a procedure in which it is declared EXTRN (external)

Programs – Call cntd..

Example 5: Using PUBLIC & EXTRN

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
PUBLIC NUMBER EQU 200H
DATA ENDS
CODE SEGMENT
START: MOV AX,DATA
        MOV DS,AX
            .
            .
        CALL ROUTINE
            .
            .
PROCEDURE ROUTINE NEAR
 EXTRN  NUMBER
         MOV AX,NUMBER
            .
            .
ROUTINE ENDP
```

Assembler Directives

- It is a direction to the assembler but not the instruction for 8086.
- The various assembler directives are given below.
- **ASSUME**:
  - It is used to tell the assembler the name of the logical segment it should use for a specified segment
  - Eg: ASSUME CS:CODE_HERE

    ASSUME DS:DATA_HERE
  - In MOV AX,[DX], it indicates that the memory location referred to by [DX], is in the logical segment DATA_HERE.

Assembler Directives cntd…

- **DB**:
  - Define byte or declare byte
  - Reserves one byte in memory and initializes to the value mentioned
  - Eg: PRICES DB 49H,98H

    NAME DB ‘MRCET’

    TEMP_STORAGE DB 100 DUP(?)
    - Reserves 100 bytes of storage in memory but initialization is not done. Program instructions will load values into these locations.

    Pressure_storage DB 20H DUP(0) ;(initialized with 0)

Assembler Directives cntd…

- **DW** : define word
- **DD** : define double word
- **DQ** : define quad word
- **DT** : define ten bytes

- **END** : Indicates end of the program
- **ENDP** : indicates end of the procedure

  Eg: SQUARE_ROOT PROC …

  SQUARE_ROOT ENDP
- **ENDS** : End segment

  Eg: CODE ENDS

Assembler Directives cntd…

- **EQU** : EQUATE

  used to give a name to some value or symbol. This name will be used in the program.

  Eg: correction_factor EQU 03H

  ADD AL, correction_factor
- **EVEN** :
  - Tells the assembler to increment the location counter to the next even address if it is not already at an even address
- **INCLUDES** : Include source code from file
- **NAME** : used to give a specific name to each assembly module when programs consisting of several modules are written.

Assembler Directives cntd…

- **ORG** : ORIGINATE
  - Used to set the location counter to a particular value.
- Eg: 1 ORG 2000H ; Sets the location counter to 2000H
  - 2 ORG $ + 100 ; tells the assembler to increment the value of the location counter by 100 from its current value.
- **SEGMENT**: Indicates the start of a logical segment. Eg: CODE SEGMENT.

Assembler Macros

- **Procedure Vs Macros**:
  - Whenever we need to use a group of instructions several times through out a program, there are two ways we can avoid having to write the group of instructions each time we want to use it.
- **Procedure**
  - Writing the group of instructions as a separate procedure
  - Then we can just call the procedure whenever we need to execute that group of instructions.
  - **Advantage:** the machine codes for the group of instructions in the procedure only have to be put in memory once
  - So, less memory usage
  - **Disadvantage**: need a stack and the overhead time is required to call the procedure and return to the calling program.

Assembler Macros cntd…

- **Macros**:
  - When the repeated group of instructions is too short or not appropriate to be written as a procedure, we use a macro.
  - A macro is a group of instructions we bracket and give a name to at the start of our program.
  - Each time we “call” the macro in our program, the assembler will insert the defined group of instructions in place of the “call”.
  - An important point here is that the assembler generates machine codes for the group of instructions each time the macro is called.
  - Replacing the macro with the instructions it represents is commonly called “expanding” the macro
  - Since the generated machine codes are right in-line with the rest of program, the processor does not have to go off to a procedure and return.

Assembler Macros cntd…

- **Advantage**: Therefore, using a macro avoids the overhead time involved in calling and returning from a procedure.
- **Disadvantage**: A disadvantage of generating in-line code each time a macro is called is that this will make the program take up more memory than using a procedure

Assembler Macros cntd…

- **Defining and Calling a Macro without parameters**:
  - Before calling a procedure, you need to save all registers on to stack. For this purpose you need to write many instructions using PUSH command.
- After executing the procedure you need to write many pop instructions to retrieve the data
- The above two cases adds more complexity to main program and is therefore not appropriate
- Two simple macros will solve the problem for us
- Here's how we write a macro to save all these registers

Assembler Macros cntd…

- PUSH_ALL MACRO
  PUSH F
  PUSH AX
  PUSH BX
  PUSH CX
  PUSH DX
  PUSH BP
  PUSH SI
  PUSH DI
  PUSH DS
  PUSH ES
  PUSH SS

  ENDM

Assembler Macros cntd…

- The PUSH_ALL Macro statement identifies the start of the Macro and gives the Macro a name. The ENDM identifies the end of the Macro.
- Now this Macro is used in the procedure as given below.

```
BREATH_RATE     PROC  FAR
ASSUME CS:PROCEDURE, DS:PATIENT_PARAMETERS
                        PUSH_ALL   ;Macro call
                        MOV AX,PATIENT_PARAMETERS
                        MOV DS,AX  ;Intialize data seg reg
```

Assembler Macros cntd…

- When the assembler assembles this program section, it will replace PUSH_ALL with the instruction that it represents and insert the machine codes for these instructions in the object code version of the program.
- The assembler listing tells you which lines were inserted by a macro call by putting a + in each program line inserted by a Macro call

Assembler Macros cntd…

- As you can see from the example here, using a Macro makes the source program much more readable because the source program does not have the long series of Push instructions cluttering it up.
- **Passing parameters to Macro**:
  - Dear Parent,

    Your ward's (Roll No.) attendance percentage is given below

    Attendance: % (percentage)

    Thank you,

Assembler Macros cntd…

- Example: To move ASCII character from one place to another.

  Source: SI, Destination: DI,

  no. of characters to be moved = CX

```
MOVE _ASCII  MACRO  NUMBER,SOURCE,DEST
             MOV CX,NUMBER
             LEA SI,SOURCE
             LEA DI,DEST
             CLD
        REP  MOVSB
             ENDM
```

Assembler Macros cntd…

- When we call the Macro, values from the calling statement will be put in the instructions in place of the dummies.
- Example: MOVE_ASCII 03DH,BLOCK_START,BLOCK_DEST

Then the assembler will expand the Macro as follows.

```
MOV CX,03DH
LEA SI,BLOCK_START
LEA DI,BLOCK_DEST
CLD
REP MOVSB
```

Instruction set of 8086 Microprocessor

Instructions

LABEL: INSTRUCTION ; COMMENT

Address identifier **Does not generate any machine code**

- Ex. START: MOV AX, BX ; copy BX into AX
- There is a one-to-one relationship between assembly and machine language instructions
- A compiled machine code implementation of a program written in a high-level language results in inefficient code

- **Two key benefits of assembly language programming**
  - **It takes up less memory**
  - **It executes much faster**

The 8086 instructions are categorized into the following main types.

1. Data Copy/Transfer Instructions:

- **These type of instructions are used to transfer data from source operand to destination operand.**
- **All the store, move, load, exchange, input and output instructions belong to this category.**

2. Arithmetic and Logical instructions:

- **All the instructions performing arithmetic, logical, increment, decrement, compare, and scan instructions belong to this category**

3. Branch Instructions:

- **These instructions transfer control of execution to the specified address.**
- **All the call, jump, interrupt and return instructions belong to this class.**

4. Loop Instructions:

- If these instructions have REP prefix with CX used as count register, they can be used to implement unconditional and conditional loops.
- The LOOP, LOOPNZ and LOOPZ instructions belong to this category.
- These are useful to implement different loop structures.

5. Machine Control Instructions:

- These instructions control the machine status.
- NOP, HLT, WAIT and LOCK instructions belong to this class.

6. Flag Manipulation Instructions:

- All the instructions which directly affect the flag register, come under this group of instructions.
- Instructions like CLD, STD, CLI, STI, etc. belong to this category of instructions.

7. Shift and Rotate Instructions:

- **These instructions involve the bitwise shifting or rotation in either direction with or without a count in CX.**

8. String Instructions:

- **These instructions involve various string manipulation operations like load, move, scan, compare , store, etc.**
- **These instructions are only to be operated upon the strings.**

Data Transfer Instructions - MOV

Mnemonic	Meaning	Format	Operation	Flags affected
MOV	Move	Mov D,S	(S) → (D)	None

Destination	Source
Memory	Accumulator
Accumulator	Memory
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Seg reg	Reg 16
Seg reg	Mem 16
Reg 16	Seg reg
Memory	Seg reg

NO MOV

Memory → Memory
Immediate → Segment Register
Segment Register → Segment Register

EX: MOV AL, BL

Data Transfer Instructions - XCHG

Mnemonic	Meaning	Format	Operation	Flags affected
XCHG	Exchange	XCHG D,S	(S) ⇆ (D)	None

Destination	Source
Accumulator	Reg 16
Memory	Register
Register	Register
Register	Memory

Example: XCHG [1234h], BX

NO XCHG

MEMs
SEG REGs

Data Transfer Instructions – LEA, LDS, LES

Mnemonic	Meaning	Format	Operation	Flags affected
LEA	Load Effective Address	LEA Reg16,EA	EA → [Reg16]	None
LDS	Load Register And DS	LDS Reg16,MEM32	[MEM32] → [Reg16] [Mem32+2] → [DS]	None
LES	Load Register and ES	LES Reg16,MEM32	[MEM32] → [Reg16] [Mem32+2] → [ES]	None

LEA SI DATA (or) MOV SI Offset DATA

The XLAT Instruction

Mnemonic	Meaning	Format	Operation	Flags
XLAT	Translate	XLAT	([AL]+[BX]+[DS]0) → [AL]	None

Used to find out the codes in case of code conversion problems, using look-up-table technique.

Example:

Assume [DS] = 0300H, [BX]=0100H, and [AL]=0DH

XLAT replaces contents of AL by contents of memory location with

PA=[DS]0 +[BX] +[AL]

= 03000H + 0100H + 0DH = 0310DH

Thus

[0310DH] → [AL]

Arithmetic Instructions: ADD, ADC, INC, AAA, DAA

Mnemonic	Meaning	Format	Operation	Flags affected
ADD	Addition	ADD D,S	[S]+[D] → [D] carry → [CF]	ALL
ADC	Add with carry	ADC D,S	[S]+[D]+[CF] → [D] carry → [CF]	ALL
INC	Increment by one	INC D	[D]+1 → [D]	ALL but CY
DAA	Decimal adjust for addition	DAA	Adjust AL for decimal Packed BCD	ALL

Examples:

Ex.1 **ADD AX,2**
ADC AX,2

Ex.2 **INC BX**
INC WORD PTR [BX]

Ex.3 AL contains 25 (packed BCD)
BL contains 56 (packed BCD)

```
ADD AL, BL
DAA
```

```
  25
+ 56
--------
7B   81
```

Arithmetic Instructions – SUB, SBB, DEC, AAS, DAS, NEG

Mnemonic	Meaning	Format	Operation	Flags affected
SUB	Subtract	SUB D,S	[D] – [S] → [D] Borrow → (CF)	All
SBB	Subtract with borrow	SBB D,S	[D] – [S] – [CF] → [D]	All
DEC	Decrement by one	DEC D	[D] - 1 → [D]	All but CF
NEG	Negate	NEG D		All
DAS	Decimal adjust for subtraction	DAS	Convert the result in AL to packed decimal format	All

Examples: DAS

```
MOV BL, 28H
MOV AL, 83H
SUB AL,BL       ;  AL=5BH
DAS             ;  adjust as AL=55H
```

Multiplication and Division, S(Source) – R8, R16, M8, M16

Mnemonic	Meaning	Format	Operation	Flags affected
MUL	Multiply (Unsigned)	MUL S	[AL].S8 → AX [AX].S16 → DX, AX	ALL
DIV	Division (Unsigned)	DIV S	1) Q([AX]/S8) → AL R([AX]/S8) → AH 2) Q([DX, AX]/S16 → AX R([DX, AX]/S16 → DX	ALL
IMUL	Integer Multiply (Signed)	IMUL S	[AL].S8 → AX [AX].S16 → DX, AX	ALL
IDIV	Integer Division (Signed)	IDIV S	1) Q([AX]/S8) → AL R([AX]/S8) → AH 2) Q([DX, AX]/S16) → AX R([DX, AX]/S16) → DX	ALL
CBW	Convert signed byte to word	CBW	MSB of AL → All bits of AH	No Flags
CWD	Convert signed word to d. word	CWD	MSB of AX → All bits of DX	No Flags

Multiplication and Division

Multiplication (MUL or IMUL)	Multiplicant	Operand (Multiplier)	Result
Byte * Byte	AL	Register or memory	AX
Word * Word	AX	Register or memory	DX :AX
Dword * Dword	EAX	Register or Memory	EDX :EAX

Division (DIV or IDIV)	Dividend	Operand (Divisor)	Quotient : Remainder
Word / Byte	AX	Register or memory	AL : AH
Dword / Word	DX:AX	Register or memory	AX : DX
Qword / Dword	EDX: EAX	Register or Memory	EAX : EDX

Multiplication and Division Examples

Ex1: Assume that each instruction starts from these values:
AL = 85H, BL = 35H, AH = 0H

1. MUL BL → AL . BL = 85H * 35H = 1B89H → AX = 1B89H

2. IMUL BL → AL . BL = 2'S AL * BL = 2'S (85H) * 35H
= 7BH * 35H = 1977H→ 2's comp → E689H → AX.

3. DIV BL → $\frac{AX}{BL} = \frac{0085H}{35H}$ = 02 (85-02*35=1B) →

AH	AL
1B	02

4. IDIV BL → $\frac{AX}{BL} = \frac{0085H}{35H} =$

AH	AL
1B	02

Ex2: AL = F3H, BL = 91H, AH = 00H

1. MUL BL → AL * BL = F3H * 91H = 89A3H → AX = 89A3H

2. IMUL BL → AL * BL = 2'S AL * 2'S BL = 2'S (F3H) * 2'S(91H) =
   0DH * 6FH = 05A3H → AX.

3.IDIV BL → $\frac{AX}{BL} = \frac{00F3H}{2'S(91H)} = \frac{00F3H}{6FH} = 2$ → (00F3 – 2*6F=15H)

AH	AL
15	02
R	**Q**

→ $\frac{POS}{NEG} = NEG$ → 2's(02) = FEH →

AH	AL
15	FE

4. DIV BL → $\frac{AX}{BL} = \frac{00F3H}{91H} = 01$ → (F3-1*91=62) →

AH	AL
62	01
R	Q

Ex3: AX= F000H, BX= 9015H, DX= 0000H

1. MUL BX = F000H * 9015H =

DX	AX
8713	B000

2. IMUL BX = 2'S(F000H) * 2'S(9015H) = 1000 * 6FEB =

DX	AX
06FE	B000

3. DIV BL = $\frac{F000H}{15H}$ = B6DH → More than FFH → Divide Error.

4. IDIV BL → $\frac{2'S(F000H)}{15H} = \frac{1000H}{15H}$ = C3H > 7F → Divide Error.

Ex4: AX= 1250H, BL= 90H

1. IDIV BL → $\frac{AX}{BL} = \frac{1250H}{90H} = \frac{POS}{NEG} = \frac{POS}{2'sNEG} = \frac{1250H}{2's(90H)} = \frac{1250H}{70H}$

= 29H (Q) → (1250 – 29 * 70) = 60H (REM)

29H (*POS*) → 2'S (29H) = D7H →

R	Q
60H	D7H

2. DIV BL → $\frac{AX}{BL} = \frac{1250H}{90H}$ = 20H→1250-20*90 =50H →

R	Q
50H	20H
AH	AL

Exercise:

1. Write a program to add two 16 bit numbers
2. Write a program to subtract a two 16 bit numbers
3. Write a program to multiply two 16 bit numbers
4. Write a program to divide a 16 bit number by an 8-bit number.

Logical Instructions

Mnemonic	Meaning	Format	Operation	Flags Affected
AND	Logical AND	AND D,S	(S) · (D) → (D)	SF, ZF, PF
OR	Logical Inclusive OR	OR D,S	(S)+(D) → (D)	SF, ZF, PF
XOR	Logical Exclusive OR	XOR D,S	(S) ⊕ (D)→(D)	SF, ZF, PF
NOT	LOGICAL NOT	NOT D	$(\overline{D}) \rightarrow (D)$	None

Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate

Destination
Register
Memory

LOGICAL Instructions

- AND
  - Uses any addressing mode except memory-to-memory and segment registers
  - Especially used in clearing certain bits (masking)

    xxxx xxxx **AND** 0000 1111 = 0000 xxxx

    (clear the upper four bits)
  - Examples: AND BL, 0FH

    AND AL, [345H]

- OR
  - Used in setting certain bits

    xxxx xxxx **OR** 0000 1111 = xxxx 1111

    (Set the lower four bits)

- XOR
  - Used in Inverting bits

xxxx xxxx XOR 0000 1111 = xxxxx'x'x'x'

-Example: Clear bits 0 and 1, set bits 6 and 7, invert bit 5 of register CL:

```
AND CL, 0FCH   ;        1111 1100B
OR CL, 0C0H    ;        1100 0000B
XOR CL, 020H   ;         0010 0000B
```

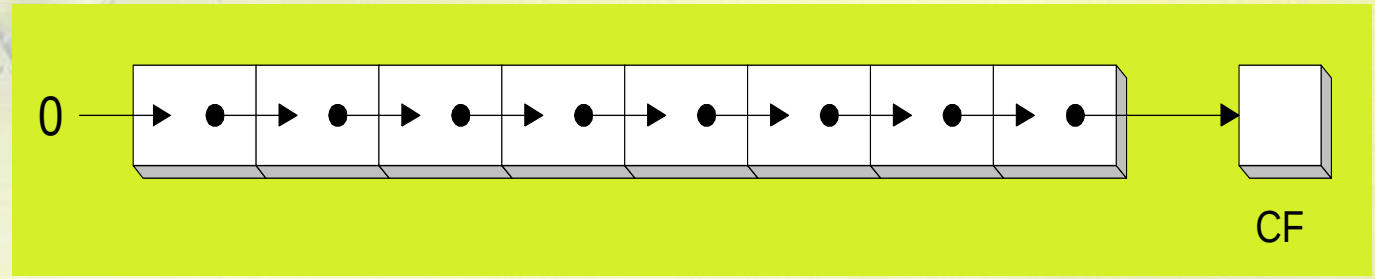
- **Exercise:** Clear bits 3 and 6, set bits 1 and 4, invert bit 0 of register BL

Shift and Rotate Instructions

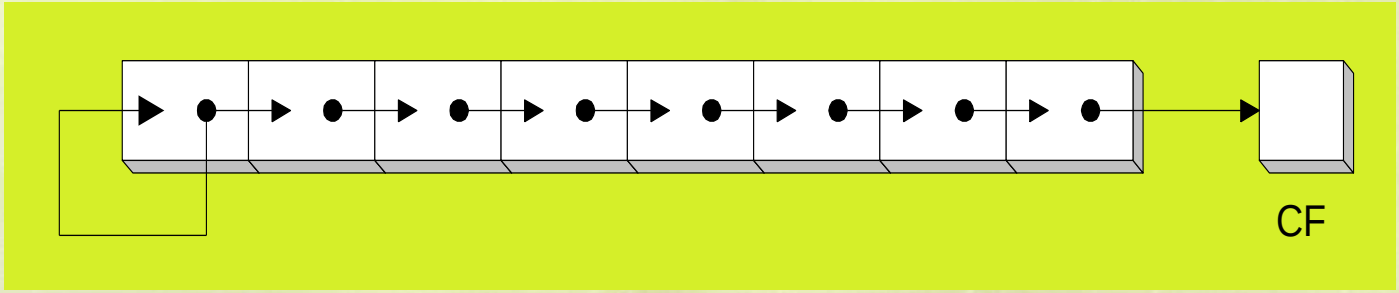
- **SHL/SAL: shift logical left/shift arithmetic left**
- **SHR: shift logical right**
- **SAR: shift arithmetic right**
- **ROL: rotate left**
- **ROR: rotate right**
- **RCL: rotate left through carry**
- **RCR: rotate right through carry**

Logical vs Arithmetic Shifts

- A logical shift fills the newly created bit position with zero:



- An arithmetic shift fills the newly created bit position with a copy of the number's sign bit:

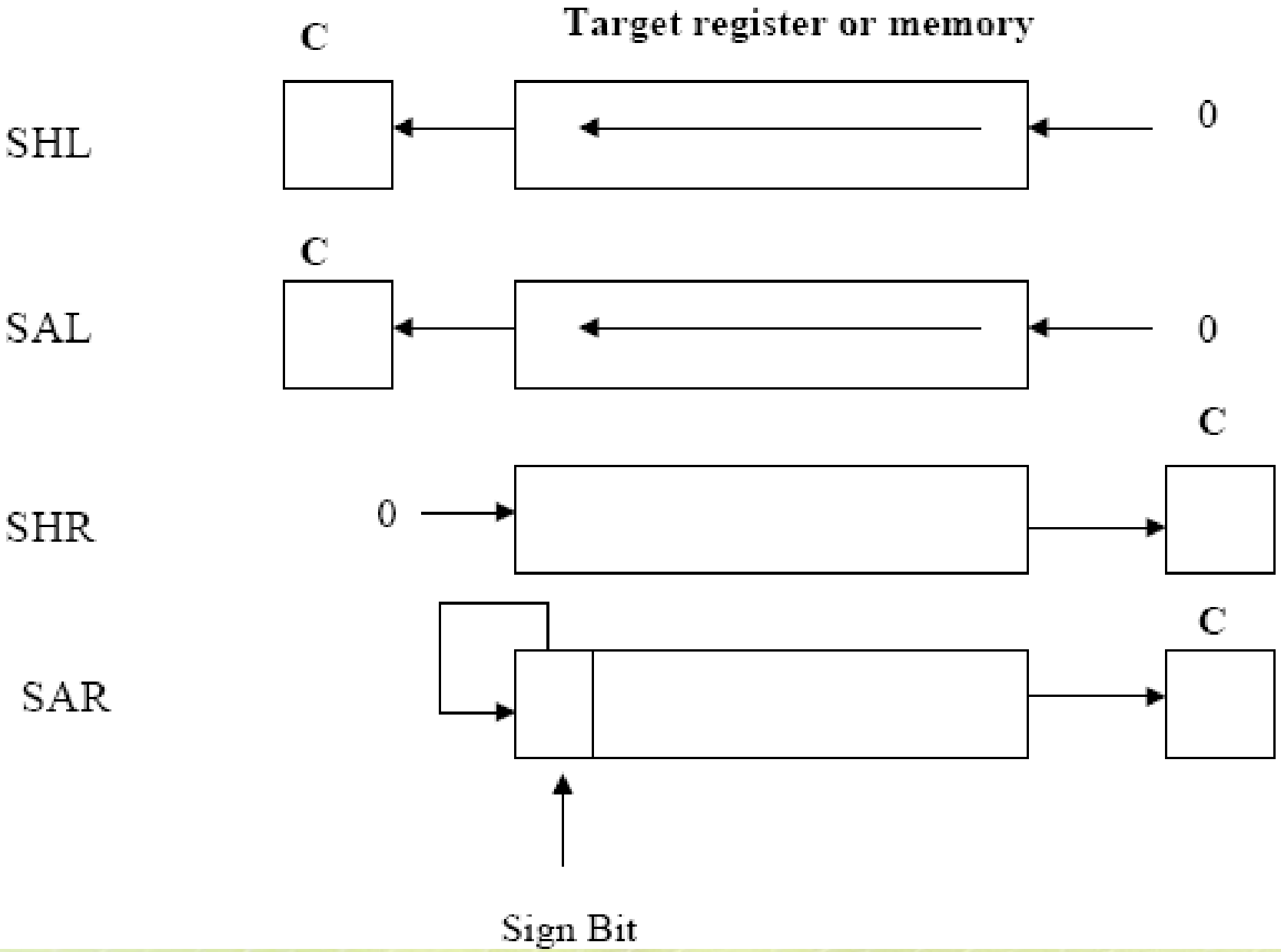


Shift Instructions

Mnemo-nic	Meaning	Format	Operation	Flags Affected
SAL/SHL	Shift arithmetic Left/shift Logical left	SAL/SHL D, Count	Shift the (D) left by the number of bit positions equal to count and fill the vacated bits positions on the right with zeros	All
SHR	Shift logical right	SHR D, Count	Shift the (D) right by the number of bit positions equal to count and fill the vacated bits positions on the left with zeros	All
SAR	Shift arithmetic right	SAR D, Count	Shift the (D) right by the number of bit positions equal to count and fill the vacated bits positions on the left with the original most significant bit	IF, DF & TF

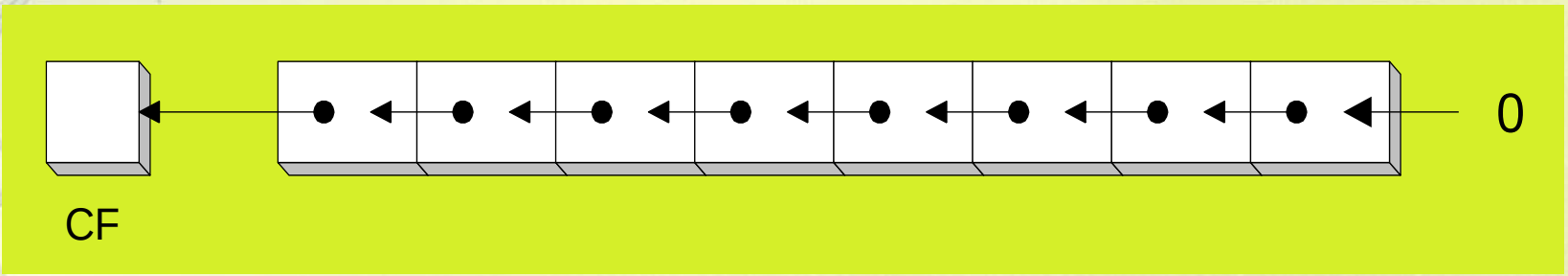
Allowed operands

Destination	Count
Register	1
Register	CL
Memory	1
Memory	CL


Target register or memory
C
SHL
0
C
SAL
0
C
SHR
0
C
SAR
Sign Bit

SHL Instruction

- The SHL (shift left) instruction performs a logical left shift on the destination operand, filling the lowest bit with 0.



- Operand types:

```
SHL reg,imm8
SHL mem,imm8
SHL reg,CL
SHL mem,CL
```

Fast Multiplication

Shifting left 1 bit multiplies a number by 2

```
mov dl,5

shl dl,1
```

Before: 0 0 0 0 0 1 0 1 = 5

After: 0 0 0 0 1 0 1 0 = 10

Shifting left *n* bits multiplies the operand by 2^n

For example, $5 * 2^2 = 20$

```
mov dl,5
shl dl,2                ; DL = 20
```

Ex. Let AL=10H, what is AL after the following code?

```
SHL AL, 1
MOV BL, AL
MOV CL,2
SHL AL,CL
ADD AL, BL
```

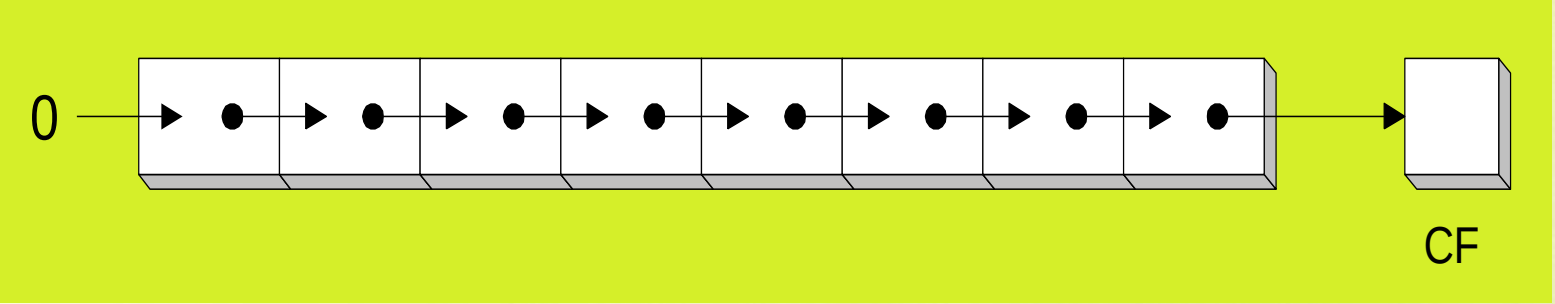
Exercise: Let AL=10H, what is AL after the following code

1.
```
SHL AL,1
MOV CL, AL
MOV CL,4
SHL AL,CL
ADD AL,BL
```

2.
```
SHL AL,1
MOV BL,AL
MOV CL,3
SHL AL,CL
ADD AL,BL
```

SHR Instruction

- The SHR (shift right) instruction performs a logical right shift on the destination operand. The highest bit position is filled with a zero.

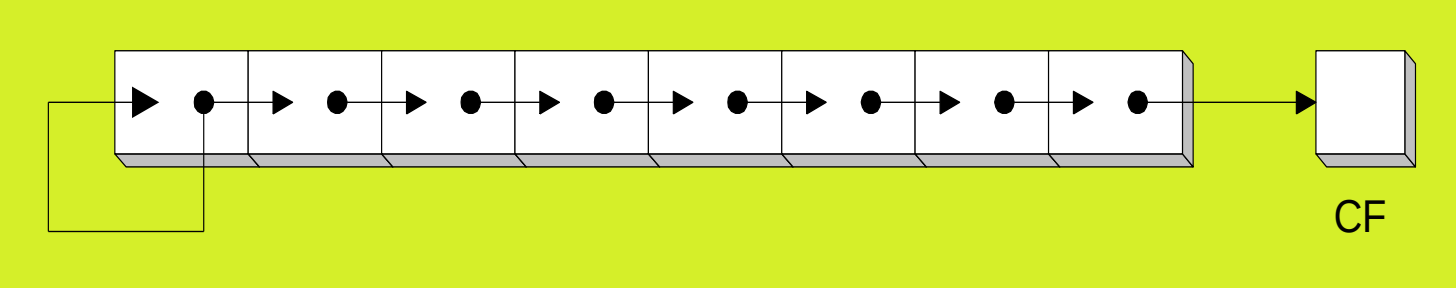


Shifting right *n* bits divides the operand by 2^n

```
MOV DL,80
SHR DL,1                    ; DL = 40
SHR DL,2                    ; DL = 10
```

SAR Instruction

- **SAR (shift arithmetic right) performs a right arithmetic shift on the destination operand.**



An arithmetic shift preserves the number's sign.

```
MOV DL,-80
SAR DL,1                    ; DL = -40
SAR DL,2                    ; DL = -10
```

- **Example:**

1. Let AL=B0H, what is AL after the following code

```
SHR AL, 1
MOV BL, AL
MOV CL,2
SHR AL,CL
ADD AL, BL
```

Exercise:

Let AL=B0H, what is AL after the following code

1.
```
SHR AL,1
MOV CL, AL
MOV CL,2
SHR AL,4
ADD AL,BL
```

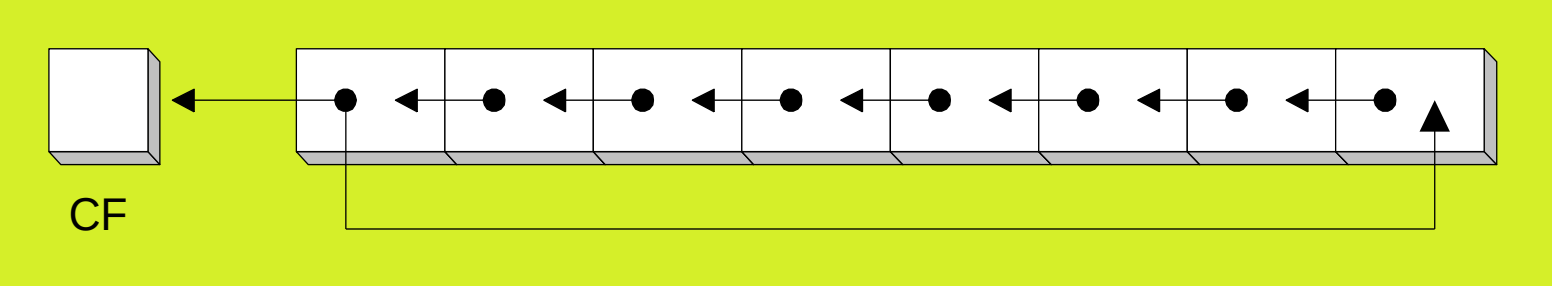
2.
```
SHR AL,1
MOV BL,AL
MOV CL,3
SHR AL,CL
ADD AL,BL
```

Rotate Instructions

Mnem-onic	Meaning	Format	Operation	Flags Affected
ROL	Rotate Left	ROL D,Count	Rotate the (D) left by the number of bit positions equal to Count. Each bit shifted out from the left most bit goes back into the rightmost bit position.	PF, SF, and ZF are left unchanged
ROR	Rotate Right	ROR D,Count	Rotate the (D) right by the number of bit positions equal to Count. Each bit shifted out from the rightmost bit goes back into the leftmost bit position.	PF, SF, and ZF are left unchanged
RCL	Rotate Left through Carry	RCL D,Count	Same as ROL except carry is attached to (D) for rotation.	PF, SF, and ZF are left unchanged
RCR	Rotate right through Carry	RCR D,Count	Same as ROR except carry is attached to (D) for rotation.	PF, SF, and ZF are left unchanged

ROL Instruction

- ROL (rotate) shifts each bit to the left
- The highest bit is copied into both the Carry flag and into the lowest bit
- No bits are lost

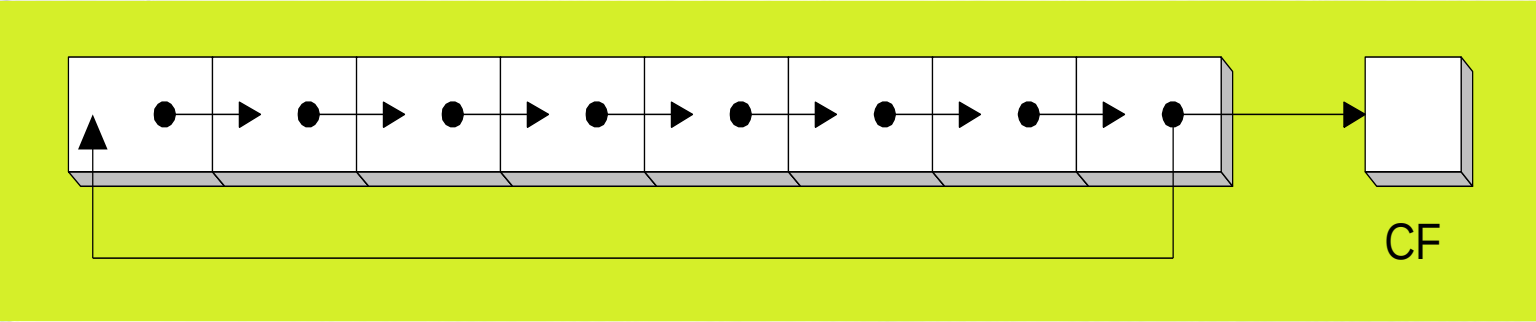



```
MOV Al,11110000b
ROL Al,1                    ; AL = 11100001b

MOV Dl,3Fh
ROL Dl,4                    ; DL = F3h
```

ROR Instruction

- ROR (rotate right) shifts each bit to the right
- The lowest bit is copied into both the Carry flag and into the highest bit
- No bits are lost

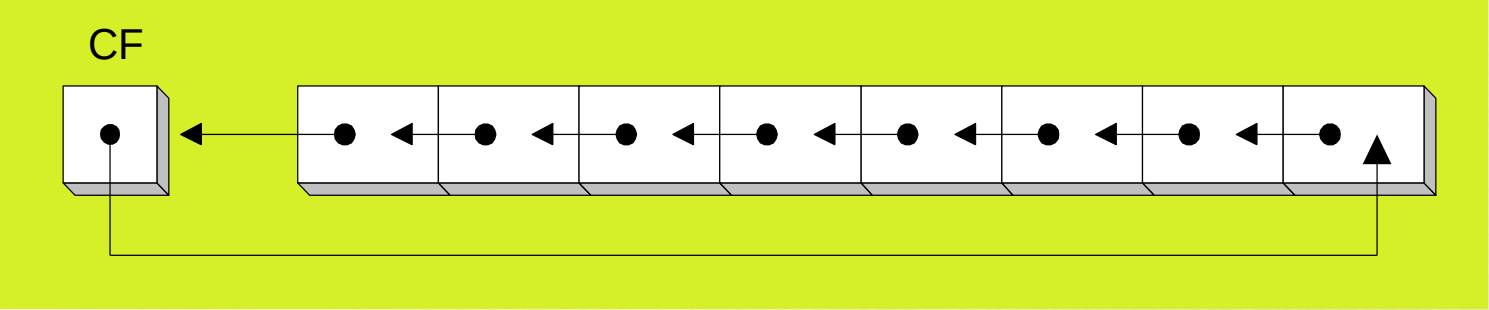



```
MOV AL,11110000b
ROR AL,1                    ; AL = 01111000b

MOV DL,3Fh
ROR DL,4                    ; DL = F3h
```

RCL Instruction

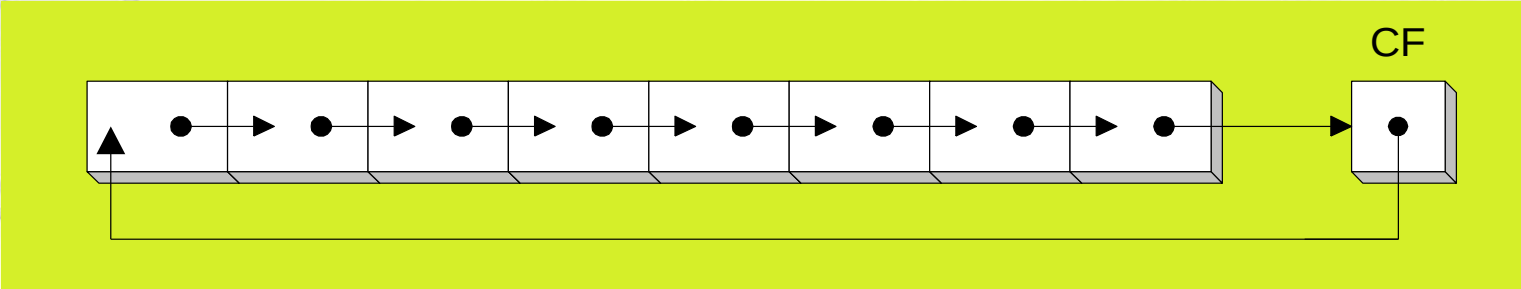
- RCL (rotate left thru carry) shifts each bit to the left
- Copies the Carry flag to the least significant bit
- Copies the most significant bit to the Carry flag




```
CLC                 ; CF = 0
MOV BL,88H          ; CF,BL = 0 10001000b
RCL BL,1            ; CF,BL = 1 00010000b
RCL BL,1            ; CF,BL = 0 00100001b
```

RCR Instruction

- RCR (rotate carry right) shifts each bit to the right
- Copies the Carry flag to the most significant bit
- Copies the least significant bit to the Carry flag




```
STC                 ; CF = 1
MOV AH,10H          ; CF,AH = 00010000 1
RCR AH,1            ; CF,AH = 10001000 0
```

Rotate Instructions

Destination	Count
Register	1
Register	CL
Memory	1
Memory	CL

Flag control instructions

MNEM-ONIC	MEANING	OPERATION	Flags Affected
CLC	Clear Carry Flag	(CF) ← 0	CF
STC	Set Carry Flag	(CF) ← 1	CF
CMC	Complement Carry Flag	(CF) ← $(CF)^{I}$	CF
CLD	Clear Direction Flag	(DF) ← 0 SI & DI will be auto incremented while string instructions are executed.	DF
STD	Set Direction Flag	(DF) ← 1 SI & DI will be auto decremented while string instructions are executed.	DF
CLI	Clear Interrupt Flag	(IF) ← 0	IF
STI	Set Interrupt Flag	(IF) ← 1	IF

Compare Instruction, CMP

Mnemonic	Meaning	Format	Operation	Flags Affected
CMP	Compare	CMP D,S	**[D] – [S]** is used in setting or resetting the flags	CF, AF, OF, PF, SF, ZF

[D] = [S] ; **ZF=1**

[D] > [S] ; **ZF=0, CF=0**

[D] < [S] ; **ZF=0, CF=1**

Allowed Operands

Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate

Strings

- An array of bytes or words located in memory
- Supported String Operations
  - Copy (move, load)
  - Search (scan)
  - Store
  - Compare

String Instruction Basics

- **Source DS:SI, Destination ES:DI**
  - You must ensure DS and ES are correct
  - You must ensure SI and DI are offsets into DS and ES respectively
  - You must move the count to CX register before executing the instruction

- Direction Flag (0 = Up, 1 = Down)
  - CLD - Increment addresses (left to right)
  - STD - Decrement addresses (right to left)

String Instructions

Instruction prefixes

Prefix	Used with	Meaning
REP	MOVS STOS	Repeat while not end of string CX ≠ 0
REPE/REPZ	CMPS SCAS	Repeat while not end of string and strings are equal. CX ≠ 0 and ZF = 1
REPNE/REPNZ	CMPS SCAS	Repeat while not end of string and strings are not equal. CX ≠ 0 and ZF = 0

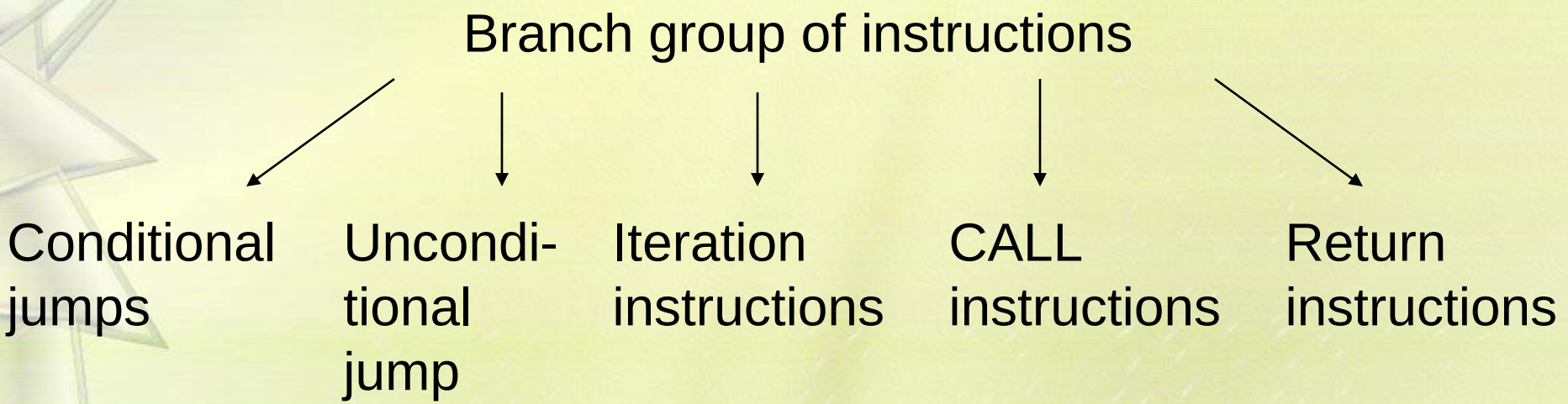
Instructions

Mnemo-Nic	meaning	format	Operation	Flags effect-ed
MOVS	Move string DS:SI →ES:DI	MOVSB/ MOVSW	[[ES]0+[DI]] ← [[DS]0+[SI]] [SI] ← [SI] ± 1 or 2 [DI] ← [DI] ± 1 or 2	none
CMPS	Compare string DS:SI →ES:DI	CMPSB/ CMPSW	Set flags as per [[ES]0+[DI]] – [[DS]0+[SI]] [SI] ← [SI] ± 1 or 2 [DI] ← [DI] ± 1 or 2	All status flags

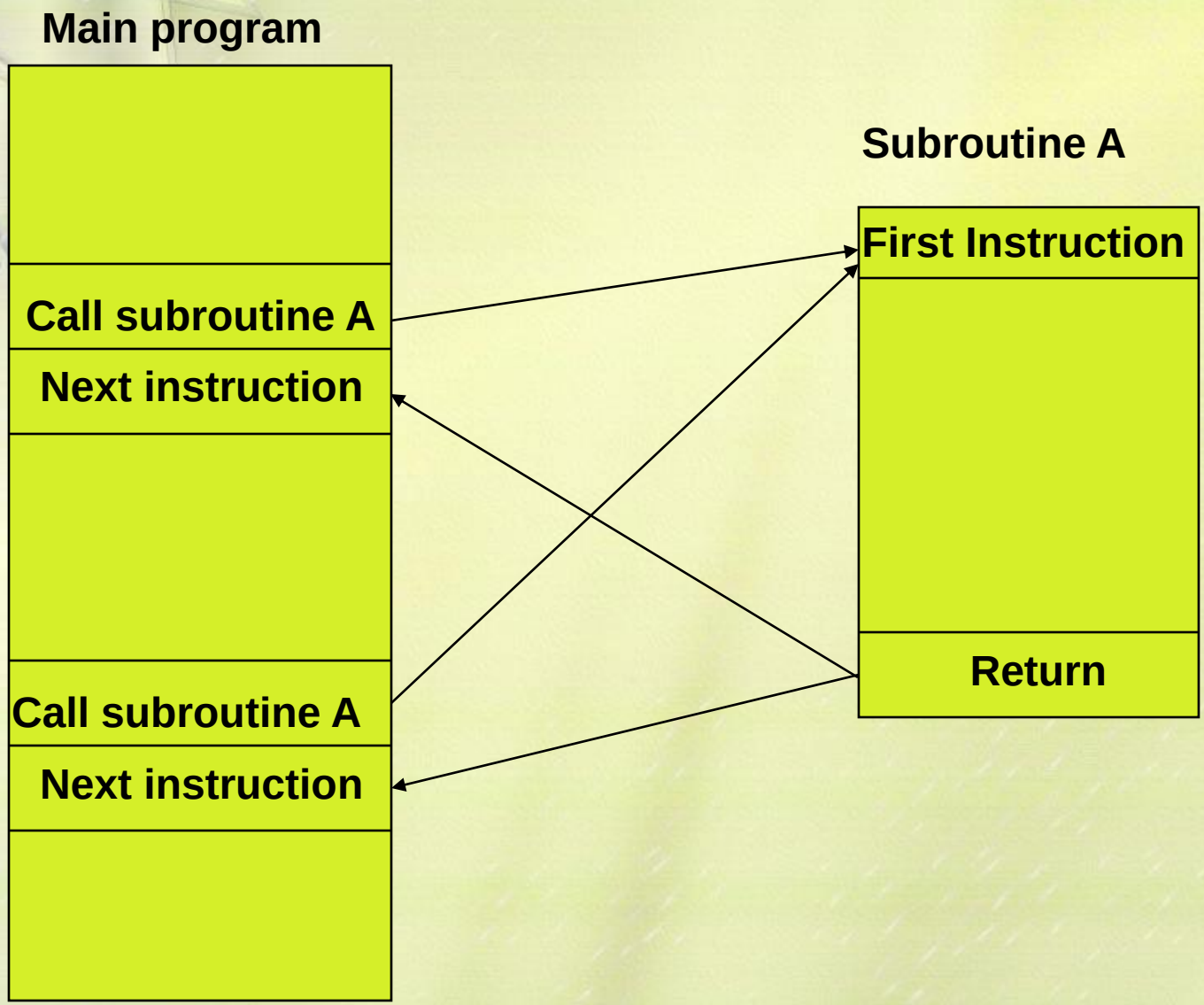
Mnemo-Nic	meaning	format	Operation
SCAS	Scan string AX – ES:DI	SCASB/ SCASW	Set flags as per [AL or AX] – [[ES]0+[DI]] [DI] ← [DI] ± 1 or 2
LODS	Load string DS:SI → AX	LODSB/ LODSW	[AL or AX] ← [[DS]0+[SI]] [SI] ← [SI] ± 1 or 2
STOS	Store string ES:DI ← AX	STOSB/ STOSW	[[ES]0+[DI]] ← [AL or AX] ± 1 or 2 [DI] ← [DI] ± 1 or 2

Branch group of instructions

Branch instructions provide lot of convenience to the programmer to perform operations selectively, repetitively etc.



SUBROUTINE & SUBROUTINE HANDILING INSTRUCTIONS



- A subroutine is a special segment of program that can be called for execution from any point in a program.
- An assembly language subroutine is also referred to as a “procedure”.
- Whenever we need the subroutine, a single instruction is inserted in to the main body of the program to call subroutine.
- To branch a subroutine the value in the IP or CS and IP must be modified.
- After execution, we want to return the control to the instruction that immediately follows the one called the subroutine i.e., the original value of IP or CS and IP must be preserved.
- Execution of the instruction causes the contents of IP to be saved on the stack. (this time [SP] ← [SP] -2)
- A new 16-bit (near-proc, mem16, reg16 i.e., Intra Segment) value which is specified by the instructions operand is loaded into IP.
- Examples: CALL 1234H

  CALL BX

  CALL [BX]

- **Inter Segment**
  - **At starting CS and IP placed in a stack.**
  - **New values are loaded in to CS and IP given by the operand.**
  - **After execution original CS, IP values placed as it is.**

Far-proc

Memptr32

These two words (32 bits) are loaded directly into IP and CS with execution at CALL instruction.

First 16 → IP

Next 16 → CS

Mnem-onic	Meaning	Format	Operation	Flags Affected
CALL	Subroutine call	CALL operand	Execution continues from the address of the subroutine specified by the operand. Information required to return back to the main program such as IP and CS are saved on the stack.	none

Operand

Near-proc

Far – proc

Memptr 16

Regptr 16

Memptr 32

RETURN

- Every subroutine must end by executing an instruction that returns control to the main program. This is the return (RET) instruction.
- By execution the value of IP or IP and CS that were saved in the stack to be returned back to their corresponding regs. (this time (SP) ← (SP)+2)

Mnem-onic	Meaning	Format	Operation	Flags Affected
RET	Return	RET or RET operand	Return to the main program by restoring IP (and CS for far-proc). If operands is present, it is added to the contents of SP.	None

Operand

None

Disp16

Loop Instructions

- These instructions are used to repeat a set of instructions several times.
- Format: LOOP Short-Label
- Operation: [CX] ← [CX]-1
- Jump is initialized to location defined by short label if CX≠0. otherwise, execute next sequential instruction.
- Instruction LOOP works w.r.t contents of CX. CX must be preloaded with a count that represents the number of times the loop is to be repeated.
- Whenever the loop is executed, contents at CX are first decremented then checked to determine if they are equal to zero.
- If CX=0, loop is complete and the instruction following loop is executed.
- If CX ≠ 0, control returns to the instruction at the label specified in the loop instruction.

LOOP Instruction contd.

General format : LOOP r8 ; r8 is 8-bit signed value.

It is a 2 byte instruction.

Used for backward jump only.

Maximum distance for backward jump is only 128 bytes.

LOOP AGAIN is almost same as:

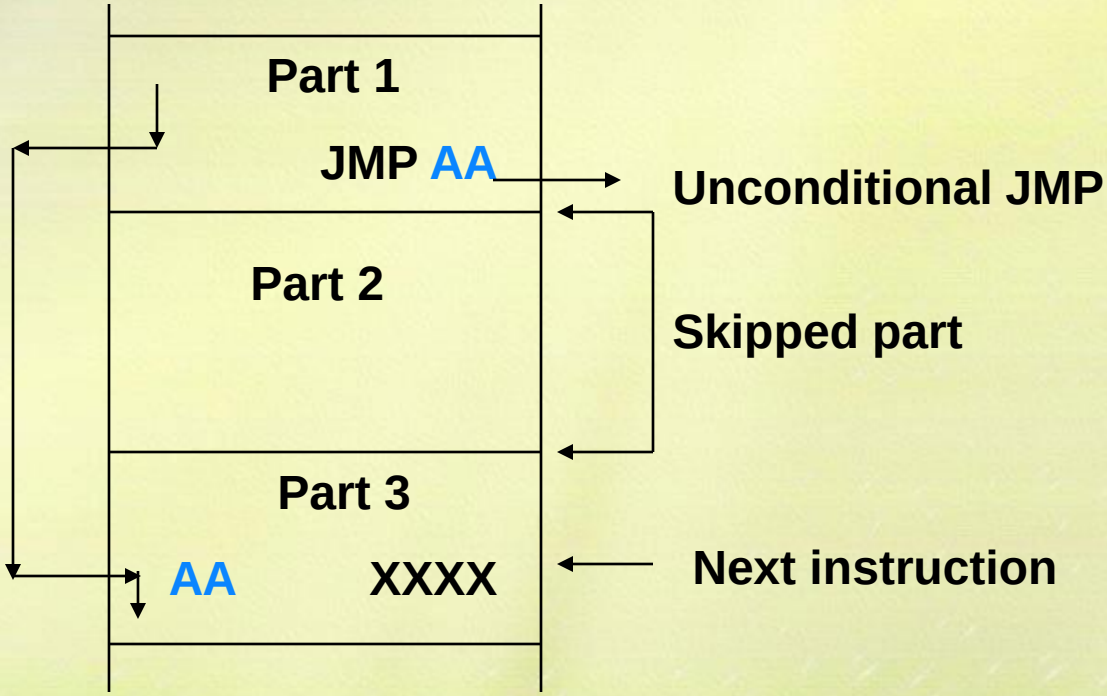
```
DEC CX
JNZ AGAIN
```

LOOP instruction does not affect any flags.

Mnemonic	meaning	format	Operation
LOOP	Loop	Loop short-label	[CX] ← [CX] – 1 Jump to location given by short-label if CX ≠ 0
LOOPE/ LOOPZ	Loop while equal/ loop while zero	LOOPE/LOOPZ short-label	[CX] ← [CX] – 1 Jump to location given by short-label if CX ≠ 0 and ZF=1
LOOPNE/ LOOPNZ	Loop while not equal/ loop while not zero	LOOPNE/LOOPNZ short-label	[CX] ← [CX] – 1 Jump to location given by short-label if CX ≠ 0 and ZF=0

Control flow and JUMP instructions

Unconditional Jump



JMP → unconditional jump

JMP Operand

Unconditional Jump

Unconditional Jump Instruction

Near Jump or
Intra segment Jump
(Jump within the segment)

Is limited to the address with in the current segment. It is achieved by modifying value in IP

Far Jump or
Inter segment Jump
(Jump to a different segment)

Permits jumps from one code segment to another. It is achieved by modifying CS and IP

Operands

Short label

Near label

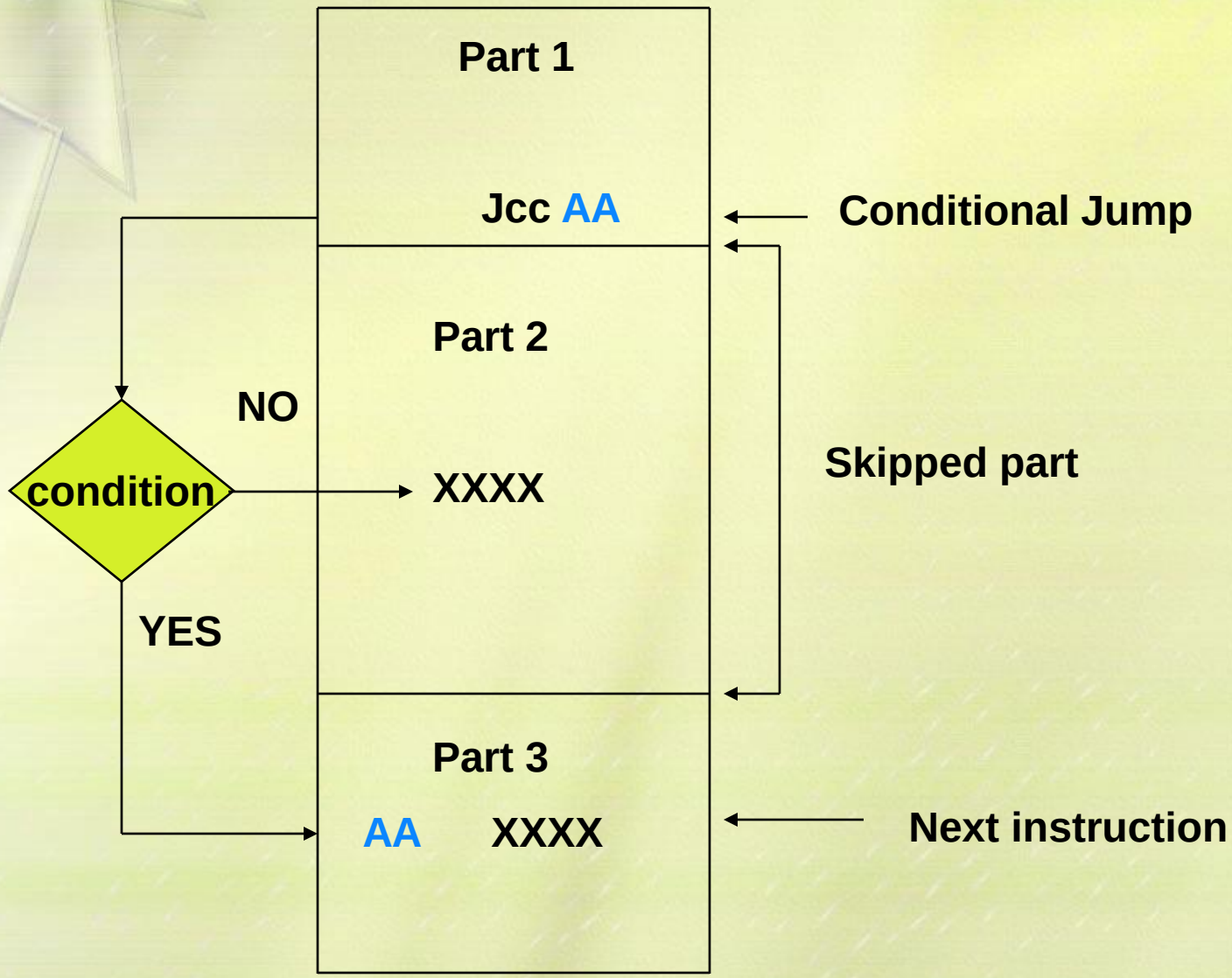
Far label ← Inter Segment Jump

Memptr16

Regptr16

memptr32 ← Inter Segment Jump

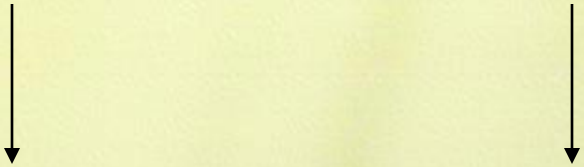
Conditional Jump



Conditional Jump instructions

Conditional Jump instructions in 8086 are just 2 bytes long. 1-byte opcode followed by 1-byte signed displacement (range of –128 to +127).

Conditional Jump Instructions



Jumps based on a single flag

Jumps based on more than one flag

Conditional Jump Instructions

Mnemonic : **Jcc**

Meaning : **Conditional Jump**

Format : **Jcc operand**

Operation : **If condition is true jump to the address specified by operand. Otherwise the next instruction is executed.**

Flags affected : **None**

TYPES

Mnemonic	meaning	condition
JA	Above	CF=0 or ZF=0
JAE	Above or Equal	CF=0
JB	Below	CF=1
JBE	Below or Equal	CF=1 or ZF=1
JC	Carry	CF=1
JCXZ	CX register is Zero	[CF or ZF]=0
JE	Equal	ZF=1
JG	Greater	ZF=0 and SF=OF
JGE	Greater or Equal	Neither SF=0 nor OF=0
JL	Less	Neither SF=1 nor OF=1

Mnemonic	meaning	condition
JLE	Less or Equal	ZF=1 or neither SF nor OF is 1
JNA	Not Above	CF =1 or Zf=1
JNAE	Not Above nor Equal	CF = 1
JNB	Not Below	CF = 0
JNBE	Not Below nor Equal	CF = 0 and ZF = 0
JNC	Not Carry	CF = 0
JNE	Not Equal	ZF = 0
JNG	Not Greater	[[SF XOR OF] or ZF]=1
JNGE	Not Greater nor Equal	Neither SF=1 nor OF=1
JNL	Not Less	Neither SF=0 nor OF=0

Mnemonic	meaning	condition
JNLE	Not Less nor Equal	ZF = 0 or both SF and OF are not 0
JNO	Not Overflow	OF = 0
JNP	Not Parity	PF = 0
JNZ	Not Zero	ZF = 0
JNS	Not Sign	SF = 0
JO	Overflow	OF = 1
JP	Parity	PF = 1
JPE	Parity Even	PF = 1
JPO	Parity Odd	PF = 0
JS	Sign	SF = 1
JZ	Zero	ZF = 1

Jumps Based on a single flag

```
JZ      r8   ;Jump if zero flag set  to 1 (Jump if result is zero)
JNZ     r8   ;Jump if Not Zero (Z flag = 0 i.e. result is nonzero)
-------------------------------------------------------------------
JS      r8   ;Jump if Sign flag set to 1 (result is negative)
JNS     r8   ;Jump if Not Sign (result is positive)
-------------------------------------------------------------------
JC      r8   ;Jump if Carry flag set to 1
JNC     r8   ;Jump if No Carry
-------------------------------------------------------------------
JP      r8   ;Jump if Parity flag set to 1 (Parity is even)
JNP     r8   ;Jump if No Parity (Parity is odd)
-------------------------------------------------------------------
JO      r8   ;Jump if Overflow flag set to 1 (result is wrong)
JNO     r8   ;Jump if No Overflow (result is correct)
```

There is no jump based on AC flag

JZ r8 ; JE (Jump if Equal) also means same.

JNZ r8 ; JNE (Jump if Not Equal) also means same.

JC r8 ;JB (Jump if below) and JNAE (Jump if Not Above or Equal) also mean same.

JNC r8 ;JAE (Jump if Above or Equal) and JNB (Jump if Not Above) also mean same.

JZ, JNZ, JC and JNC used after arithmetic operation

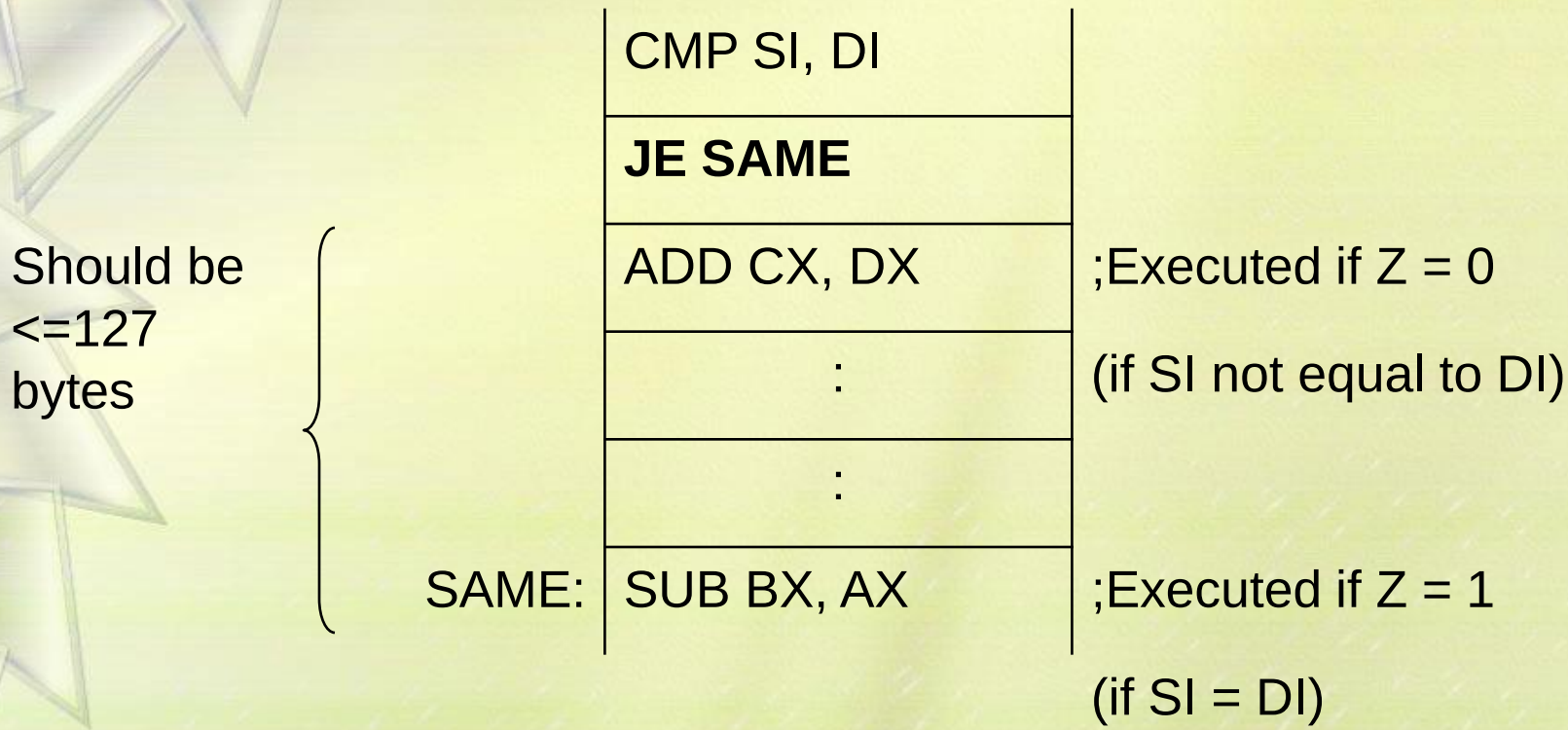
JE, JNE, JB, JNAE, JAE and JNB are used after a compare operation.

JP r8 ; JPE (Jump if Parity Even) also means same.

JNP r8 ; JPO (Jump if Parity Odd) also means same.

Examples for JE or JZ instruction

Ex. for forward jump (Only examples for JE given)



Examples for JE or JZ instruction

Ex. for backward jump

Should be <= 128 bytes

BACK:	SUB BX, AX	; executed if Z = 1
	:	(if SI = DI)
	:	
	CMP SI, DI	
	JE BACK	
	ADD CX, DX	;executed if Z = 0
		(if SI not equal to DI)

Jumping beyond -128 to +127?

What if >127 bytes

	Requirement
	CMP SI, DI
	JE SAME
	ADD CX, DX
	:
	:
SAME:	SUB BX, AX

	Then do this!
	CMP SI, DI
	JNE NEXT
	JMP SAME
NEXT:	ADD CX, DX
	:
	:
SAME:	SUB BX, AX

Range for JMP (unconditional jump) can be $\pm 2^{15} = \pm$ 32K JMP instruction discussed in detail later

Terms used in comparison

Above and Below used for comparing Unsigned nos.

Greater than and less than used with signed numbers.

All Intel microprocessors use this convention.

95H is above 65H	Unsigned comparison - True
95H is less than 65H	Signed comparison - True
	95H is negative, 65H is positive
65H is below 95H	Unsigned comparison - True
65H is greater than 95H	Signed comparison - True

Jump on multiple flags

Conditional Jumps based on more than one flag are used after a CMP (compare) instruction.

JBE or — Jump if Below or Equal

JNA — Jump if Not Above

Jump if	*No Jump if*	*Ex.*
Cy = 1 OR Z= 1	Cy = 0 AND Z = 0	CMP BX, CX
Below OR Equal	Surely Above	JBE BX_BE

BX_BE (BX is Below or Equal) is a symbolic location

Jump on multiple flags contd.

JNBE or JA — Jump if Not (Below or Equal) / Jump if Above

Jump if	*No Jump if*	*Ex.*
Cy = 0 AND Z= 0	Cy = 1 OR Z = 1	CMP BX, CX
Surely Above	Below OR Equal	JA BXabove

BXabove (BX is above) is a symbolic location

Jump on multiple flags contd.

JLE or JNG — Jump if Less than OR Equal / Jump if Not Greater than

Jump if	*No Jump if*
S = 1 AND V = 0 (surely negative) OR (S = 0 AND V = 1) (wrong answer positive!) OR Z = 1 (equal)	S = 0 AND V = 0 (surely positive) OR (S = 1 AND V = 1) (wrong answer negative!) AND Z = 0 (not equal)
i.e. S XOR V = 1 OR Z = 1	i.e. S XOR V = 0 AND Z = 0

Jump on multiple flags contd.

JNLE or Jump if Not (Less than OR Equal)

JG Jump if Greater than

Jump if	*No Jump if*
S = 0 AND V = 0 (surely positive) OR (S = 1 AND V = 1) (wrong answer negative!) AND Z = 0 (not equal)	S = 1 AND V = 0 (surely negative) OR (S = 0 AND V = 1) (wrong answer positive!) OR Z = 1 (equal)
i.e. S XOR V = 0 AND Z = 0	i.e. S XOR V = 1 OR Z = 1

Jump on multiple flags contd.

JL or JNGE — Jump if Less than / Jump if Not (Greater than OR Equal)

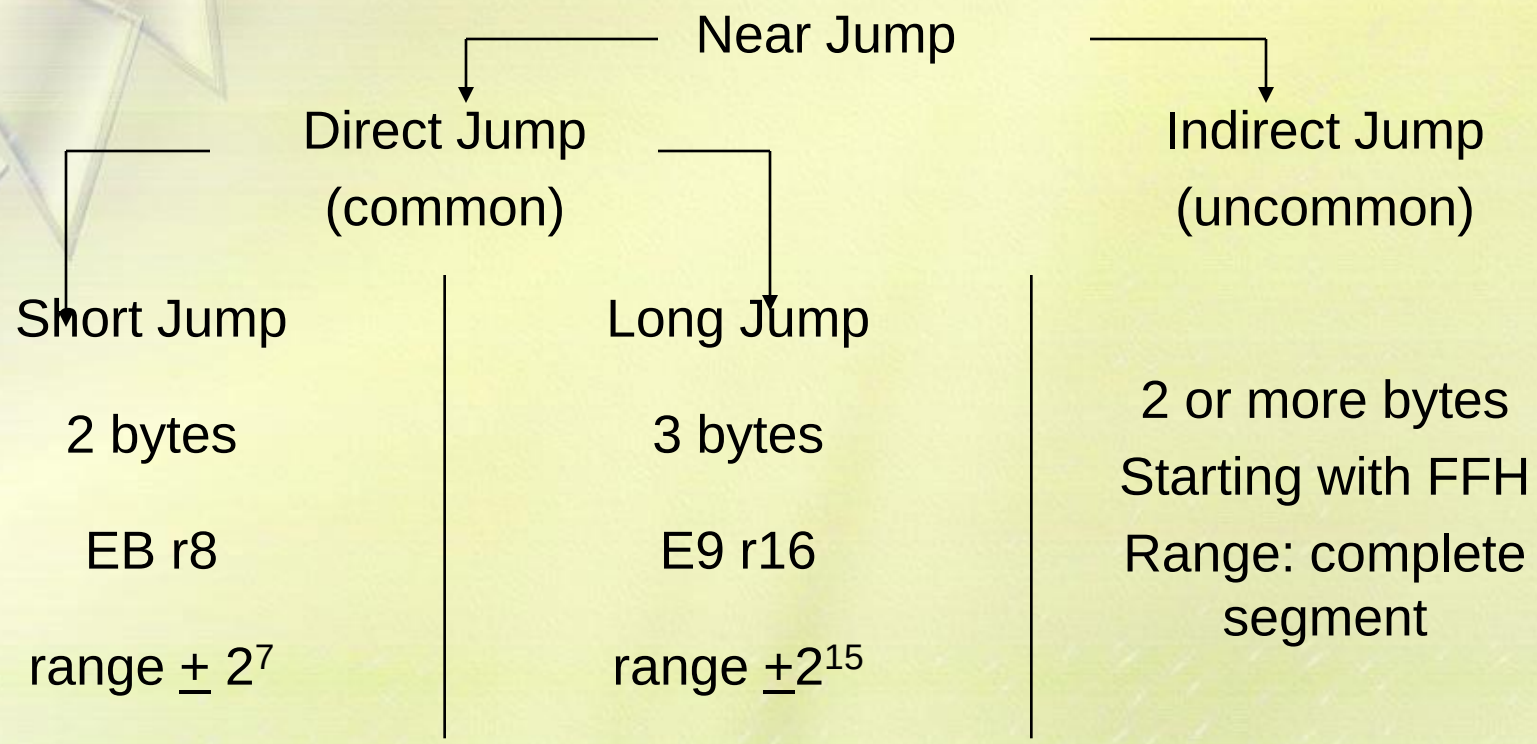
Jump if	*No Jump if*
S = 1 AND V = 0 (surely negative) OR (S = 0 AND V = 1) (wrong answer positive!) i.e. S XOR V = 1 When S = 1, result cannot be 0	S = 0 AND V = 0 (surely positive) OR (S = 1 AND V = 1) (wrong answer negative!) i.e. S XOR V = 0 When S = 0, result can be 0

Jump on multiple flags contd.

JNL or JGE — Jump if Not Less than / Jump if Greater than OR Equal

Jump if	*No Jump if*
S = 0 AND V = 0 (surely positive) OR (S = 1 AND V = 1) (wrong answer negative!)	S = 1 AND V = 0 (surely negative) OR (S = 0 AND V = 1) (wrong answer positive!)
i.e. S XOR V = 0 When S = 0, result can be 0	i.e. S XOR V = 1 When S = 1, result cannot be 0

Near Jump



3 Near Jump and 2 Far Jump instructions have the same mnemonic JMP but different opcodes

Short Jump

2 byte (EB r8) instruction Range: -128 to +127 bytes

Backward jump: Assembler knows the quantum of jump.

Generates Short Jump code if <=128 bytes is the required jump

Generates code for Long Jump if >128 bytes is the required jump

---

Forward jump: Assembler doesn't know jump quantum in pass 1.

Assembler reserves 3 bytes for the forward jump instruction.

If jump distance turns out to be >128 bytes, the instruction is coded as E9 r16 (E9H = Long jump code).

If jump distance becomes <=128 bytes, the instruction is coded as EB r8 followed by code for NOP (E8H = Short jump code).

Short Jump contd.

SHORT Assembler Directive

Assembler generates only 2 byte Short Jump code for forward jump, if the SHORT assembler directive is used.

Programmer should ensure that the Jump distance is <=127 bytes

```
        JMP SHORT   SAME
          :
          :
SAME:   MOV CX, DX
```

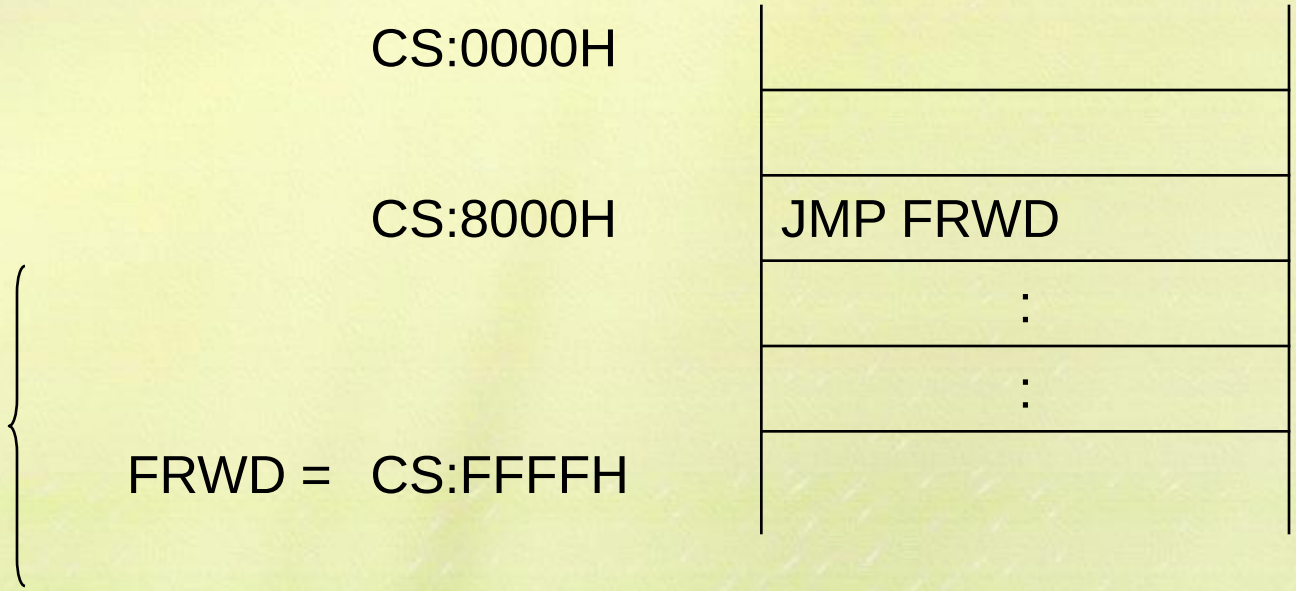
Long Jump

3-byte (E9 r16) instruction Range: -32768 to +32767 bytes

Long Jump can cover entire 64K bytes of Code segment

Long Jump can handle it as jump quantum is <=32767

FRWD = CS:FFFFH



Long Jump contd.

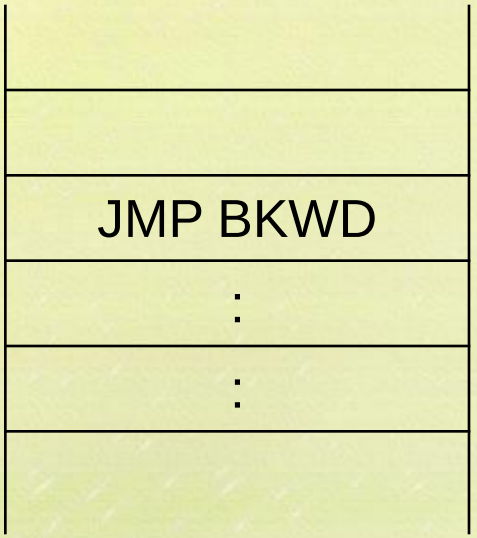
It can cover entire 64K bytes of Code segment

Long Jump can handle it as jump quantum is <=32768

BKWD = CS:0000H

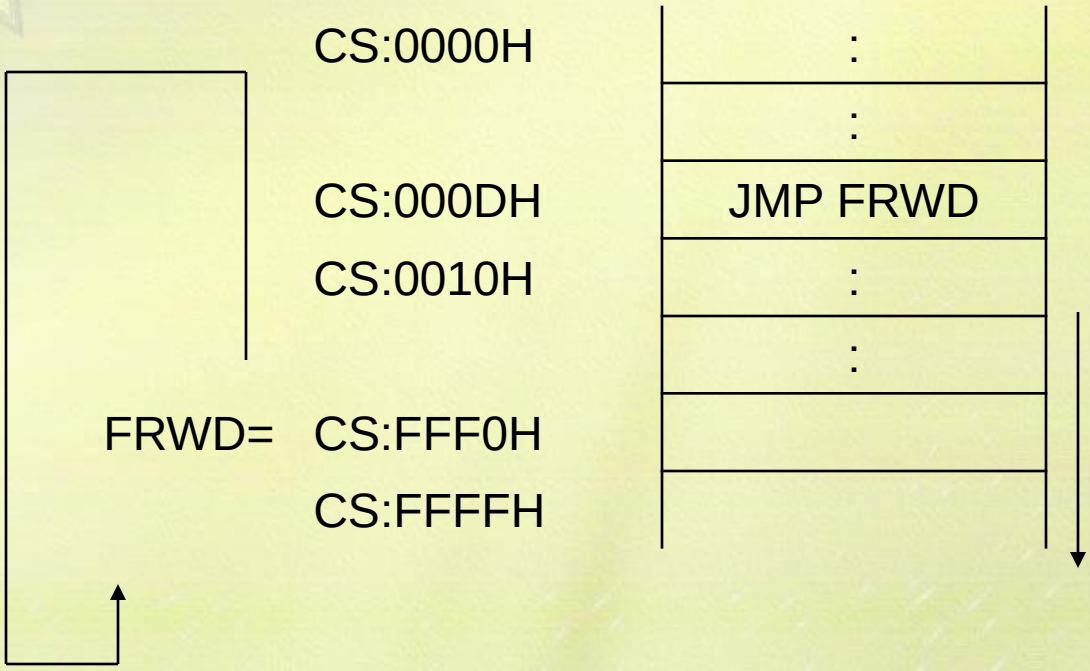
CS:8000H

CS:FFFFH



Long Jump or Short Jump?

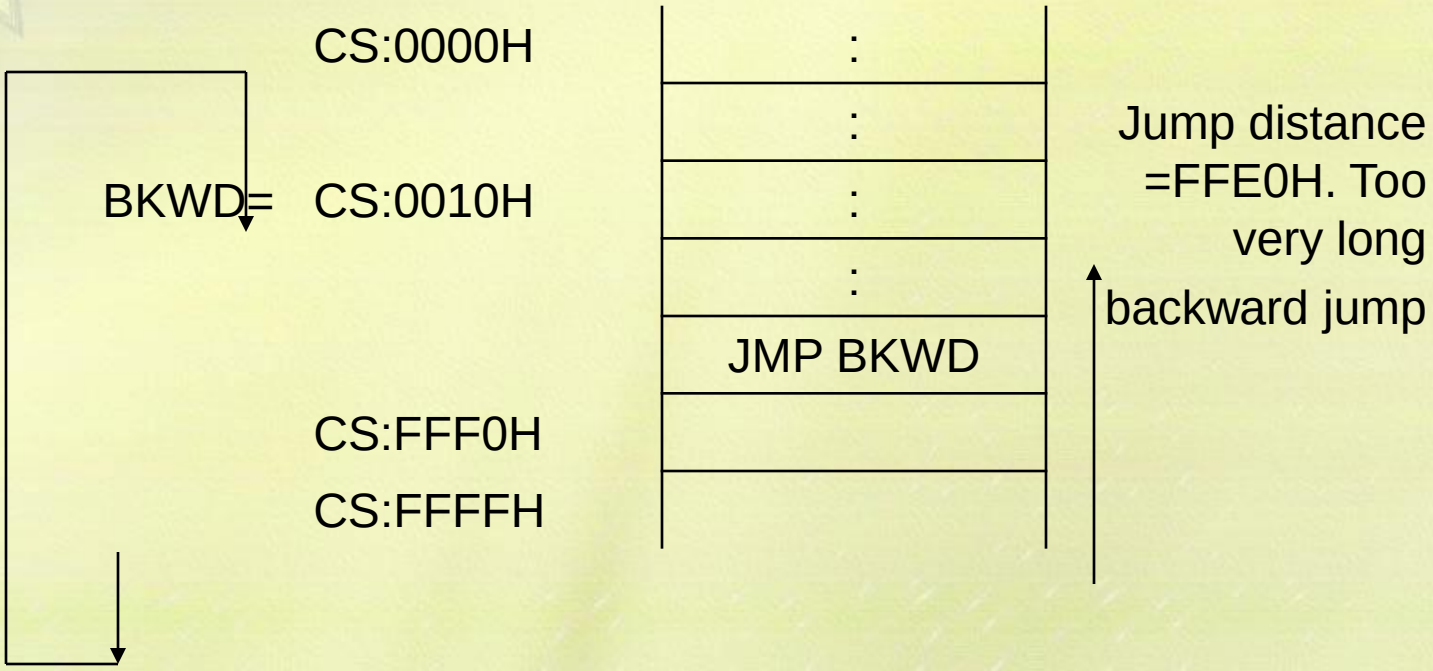
Can be treated as a small (20H) backward branch!



Jump distance =FFE0H. Too very long forward jump

Long Jump or Short Jump?

Can be treated as a small (20H) forward branch!



Intra segment indirect Jump

Near Indirect Jump is uncommon.

Instruction length: 2 or more bytes

Range: complete segment

Ex.1: JMP DX

If DX = 1234H, branches to CS:1234H

1234H is not signed relative displacement

Ex. 2: JMP wordptr 2000H[BX]

BX	1234H	DS:3234H	5678H	Branches to
		DS:3236H	AB22H	CS:5678H

Far Jump

Far Jump

Direct Jump
(common)

5 bytes

EA,2 byte offset, 2 byte segment

Range: anywhere

Indirect Jump
(uncommon)

2 or more bytes
Starting with FFH
Range: anywhere

3 Near Jump and 2 Far Jump instructions have the same mnemonic JMP but different opcodes

Inter segment Direct Jump

Also called Far Direct Jump

It is the common inter segment jump scheme

It is a 5 byte instruction

1 byte opcode (EAH)

2 byte offset value

2 byte segment value

Ex. JMP Far ptr LOC

Inter segment Indirect Jump

Instruction length depends on the way jump location is specified

It can be a minimum of 2 bytes

Ex. JMP DWORD PTR 2000H[BX]

Inter segment Indirect Jump

Also called Far Indirect Jump

It is not commonly used

Instruction length is a minimum of 2 bytes.

It depends on the way jump location is specified

Ex. JMP DWORD PTR 2000H[BX]

BX	1234H

DS:3234H	5678H
DS:3236H	ABCDH

Branches to

ABCDH:5678H

It is a 4-byte instruction

Machine control instructions

HLT instruction – HALT processing

the HLT instruction will cause the 8086 to stop fetching and executing instructions. The 8086 will enter a halt state. The only way to get the processor out of the halt state are with an interrupt signal on the INTR pin or an interrupt signal on NMI pin or a reset signal on the RESET input.

NOP instruction

this instruction simply takes up three clock cycles and does no processing. After this, it will execute the next instruction. This instruction is normally used to provide delays in between instructions.

ESC instruction

whenever this instruction executes, the microprocessor does NOP or access a data from memory for coprocessor. This instruction passes the information to 8087 math processor. Six bits of ESC instruction provide the opcode to coprocessor.

when 8086 fetches instruction bytes, co-processor also picks up these bytes and puts in its queue. The co-processor will treat normal 8086 instructions as NOP. Floating point instructions are executed by 8087 and during this 8086 will be in WAIT.

Machine control instructions contd

LOCK instruction

this is a prefix to an instruction. This prefix makes sure that during execution of the instruction, control of system bus is not taken by other microprocessor.

in multiprocessor systems, individual microprocessors are connected together by a system bus. This is to share the common resources. Each processor will take control of this bus only when it needs to use common resource.

the lock prefix will ensure that in the middle of an instruction, system bus is not taken by other processors. This is achieved by hardware signal 'LOCK' available on one of the CPU pin. This signal will be made active during this instruction and it is used by the bus control logic to prevent others from taking the bus.

once this instruction is completed, lock signal becomes inactive and microprocessors can take the system bus.

WAIT instruction

this instruction takes 8086 to an idle condition. The CPU will not do any processing during this. It will continue to be in idle state until TEST pin of 8086 becomes low or an interrupt signal is received on INTR or NMI. On valid interrupt, ISR is executed and processor enters the idle state again.

End of Unit II

Unit III

I/O Interface

Interfacing with Advanced devices

Communication Interface

I/O Interface

8255 PPI

8255

D:\College\
MP & MC\2017-18

DAC and ADC Converters

DAC and ADC:

M3L8 8255 ADC & DAC.pdf

Stepper Motor and 8257:

Microsoft Office
Word Document

Interfacing with Advanced Devices

Memory Interfacing to 8086:

D:\College\
MP & MC\2017-18

Interrupts:

Microsoft Office
werPoint Presentati

8259:

Microsoft Office
Word Document

Serial Communication Interface

Microsoft Office
Word Document

PIO 8255 (cont..)

- The parallel input-output port chip 8255 is also called as programmable ***peripheral input-output port.*** The Intel's 8255 is designed for use with Intel's 8-bit, 16-bit and higher capability microprocessors. It has 24 input/output lines which may be individually programmed in two groups of twelve lines each, or three groups of eight lines. The two groups of I/O pins are named as Group A and Group B. Each of these two groups contains a subgroup of eight I/O lines called as 8-bit port and another subgroup of four lines or a 4-bit port. Thus Group A contains an 8-bit port A along with a 4-bit port. C upper.

PIO 8255 (cont..)

- The port A lines are identified by symbols PA_0-PA_7 while the port C lines are identified as PC_4-PC_7. Similarly, Group B contains an 8-bit port B, containing lines PB_0-PB_7 and a 4-bit port C with lower bits PC_0- PC_3. The port C upper and port C lower can be used in combination as an 8-bit port C.
- Both the port C are assigned the same address. Thus one may have either three 8-bit I/O ports or two 8-bit and two 4-bit ports from 8255. All of these ports can function independently either as input or as output ports. This can be achieved by programming the bits of an internal register of 8255 called as control word register (CWR).

PIO 8255 (cont..)

- The internal block diagram and the pin configuration of 8255 are shown in fig.
- The 8-bit data bus buffer is controlled by the read/write control logic. The read/write control logic manages all of the internal and external transfers of both data and control words.
- $\overline{RD}$, $\overline{WR}$, A_1, A_0 and RESET are the inputs provided by the microprocessor to the READ/ WRITE control logic of 8255. The 8-bit, 3-state bidirectional buffer is used to interface the 8255 internal data bus with the external system data bus.

PIO 8255 (cont..)

- This buffer receives or transmits data upon the execution of input or output instructions by the microprocessor. The control words or status information is also transferred through the buffer.
- The signal description of 8255 are briefly presented as follows :
- **PA_7-PA_0**: These are eight port A lines that acts as either latched output or buffered input lines depending upon the control word loaded into the control word register.
- **PC_7-PC_4** : Upper nibble of port C lines. They may act as either output latches or input buffers lines.

PIO 8255 (cont..)

- This port also can be used for generation of handshake lines in mode 1 or mode 2.
- **PC_3-PC_0** : These are the lower port C lines, other details are the same as PC_7-PC_4 lines.
- **PB_0-PB_7** : These are the eight port B lines which are used as latched output lines or buffered input lines in the same way as port A.
- $\overline{\textbf{RD}}$: This is the input line driven by the microprocessor and should be low to indicate read operation to 8255.
- $\overline{\textbf{WR}}$: This is an input line driven by the microprocessor. A low on this line indicates write operation.

PIO 8255 (cont..)

- $\overline{\mathbf{CS}}$: This is a chip select line. If this line goes low, it enables the 8255 to respond to RD and WR signals, otherwise RD and WR signal are neglected.
- $\mathbf{A_1}$-$\mathbf{A_0}$: These are the address input lines and are driven by the microprocessor. These lines A_1-A_0 with $\overline{RD}$, $\overline{WR}$ and $\overline{CS}$ from the following operations for 8255. These address lines are used for addressing any one of the four registers, i.e. three ports and a control word register as given in table below.
- In case of 8086 systems, if the 8255 is to be interfaced with lower order data bus, the A_0 and A_1 pins of 8255 are connected with A_1 and A_2 respectively.

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	Input (Read) cycle
0	1	0	0	0	Port A to Data bus
0	1	0	0	1	Port B to Data bus
0	1	0	1	0	Port C to Data bus
0	1	0	1	1	CWR to Data bus

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	Output (Write) cycle
1	0	0	0	0	Data bus to Port A
1	0	0	0	1	Data bus to Port B
1	0	0	1	0	Data bus to Port C
1	0	0	1	1	Data bus to CWR

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	Function
X	X	1	X	X	Data bus tristated
1	1	0	X	X	Data bus tristated

Control Word Register

PIO 8255.

- **D_0-D_7** : These are the data bus lines those carry data or control word to/from the microprocessor.
- **RESET** : A logic high on this line clears the control word register of 8255. All ports are set as input ports by default after reset.

Block Diagram of 8255 (Architecture) (cont..)

- It has a 40 pins of 4 groups.

1. Data bus buffer
2. Read Write control logic
3. Group A and Group B controls
4. Port A, B and C

- ***Data bus buffer***: This is a tristate bidirectional buffer used to interface the 8255 to system databus. Data is transmitted or received by the buffer on execution of input or output instruction by the CPU.
- Control word and status information are also transferred through this unit.

Block Diagram of 8255 (Architecture) (cont..)

- ***Read/Write control logic***: This unit accepts control signals ($\overline{RD}$, $\overline{WR}$) and also inputs from address bus and issues commands to individual group of control blocks (Group A, Group B).
- It has the following pins.

a) **$\overline{CS}$ –** Chipselect : A low on this PIN enables the communication between CPU and 8255.

b) **$\overline{RD}$** (Read) **–** A low on this pin enables the CPU to read the data in the ports or the status word through data bus buffer.

Block Diagram of 8255 (Architecture) (cont..)

c) **$\overline{WR}$** (Write) : A low on this pin, the CPU can write data on to the ports or on to the control register through the data bus buffer.

d) **RESET**: A high on this pin clears the control register and all ports are set to the input mode

e) **A_0** and **A_1** (Address pins): These pins in conjunction with $\overline{RD}$ and $\overline{WR}$ pins control the selection of one of the 3 ports.

- ***Group A and Group B controls*** : These block receive control from the CPU and issues commands to their respective ports.

Block Diagram of 8255 (Architecture) (cont..)

- Group A - PA and PCU (PC_7 –PC_4)
- Group B - PCL (PC_3 – PC_0)
- Control word register can only be written into no read operation of the CW register is allowed.
- a) **Port A**: This has an 8 bit latched/buffered O/P and 8 bit input latch. It can be programmed in 3 modes – mode 0, mode 1, mode 2.

  b) **Port B**: This has an 8 bit latched / buffered O/P and 8 bit input latch. It can be programmed in mode 0, mode1.

Block Diagram of 8255 (Architecture).

c) **Port C** : This has an 8 bit latched input buffer and 8 bit out put latched/buffer. This port can be divided into two 4 bit ports and can be used as control signals for port A and port B. it can be programmed in mode 0.

Modes of Operation of 8255 (cont..)

- These are two basic modes of operation of 8255. I/O mode and Bit Set-Reset mode (BSR).
- In I/O mode, the 8255 ports work as programmable I/O ports, while in BSR mode only port C (PC_0-PC_7) can be used to set or reset its individual port bits.
- Under the I/O mode of operation, further there are three modes of operation of 8255, so as to support different types of applications, mode 0, mode 1 and mode 2.

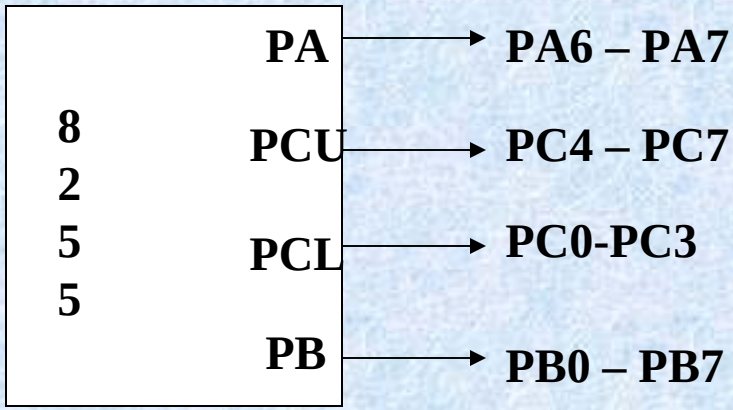
Modes of Operation of 8255 (cont..)

- ***BSR Mode***: In this mode any of the 8-bits of port C can be set or reset depending on D_0 of the control word. The bit to be set or reset is selected by bit select flags D_3, D_2 and D_1 of the CWR as given in table.
- **I/O Modes** :

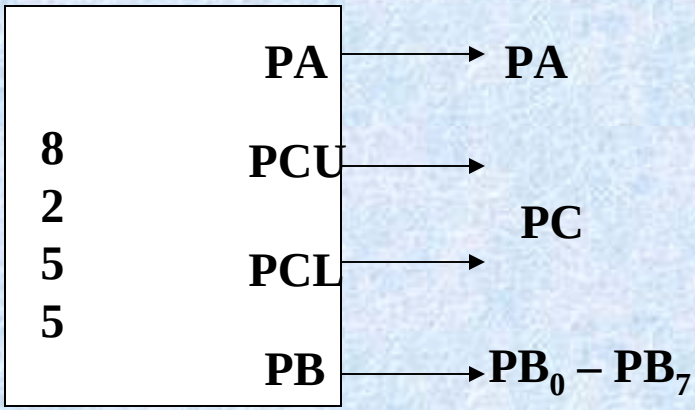
  a) ***Mode 0 (Basic I/O mode):*** This mode is also called as basic input/output mode. This mode provides simple input and output capabilities using each of the three ports. Data can be simply read from and written to the input and output ports respectively, after appropriate initialisation.

D_3	D_2	D_1	Selected bits of port C
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

BSR Mode : CWR Format



All Output



Port A and Port C acting as O/P. Port B acting as I/P

Mode 0

Modes of Operation of 8255 (cont..)

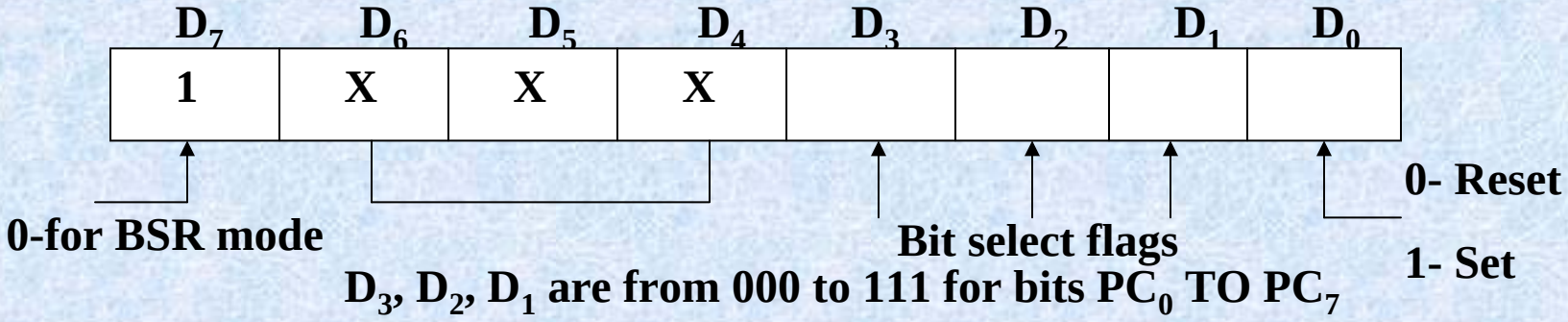
- The salient features of this mode are as listed below:

1. Two 8-bit ports (port A and port B)and two 4-bit ports (port C upper and lower) are available. The two 4-bit ports can be combinedly used as a third 8-bit port.
2. Any port can be used as an input or output port.
3. Output ports are latched. Input ports are not latched.
4. A maximum of four ports are available so that overall 16 I/O configuration are possible.

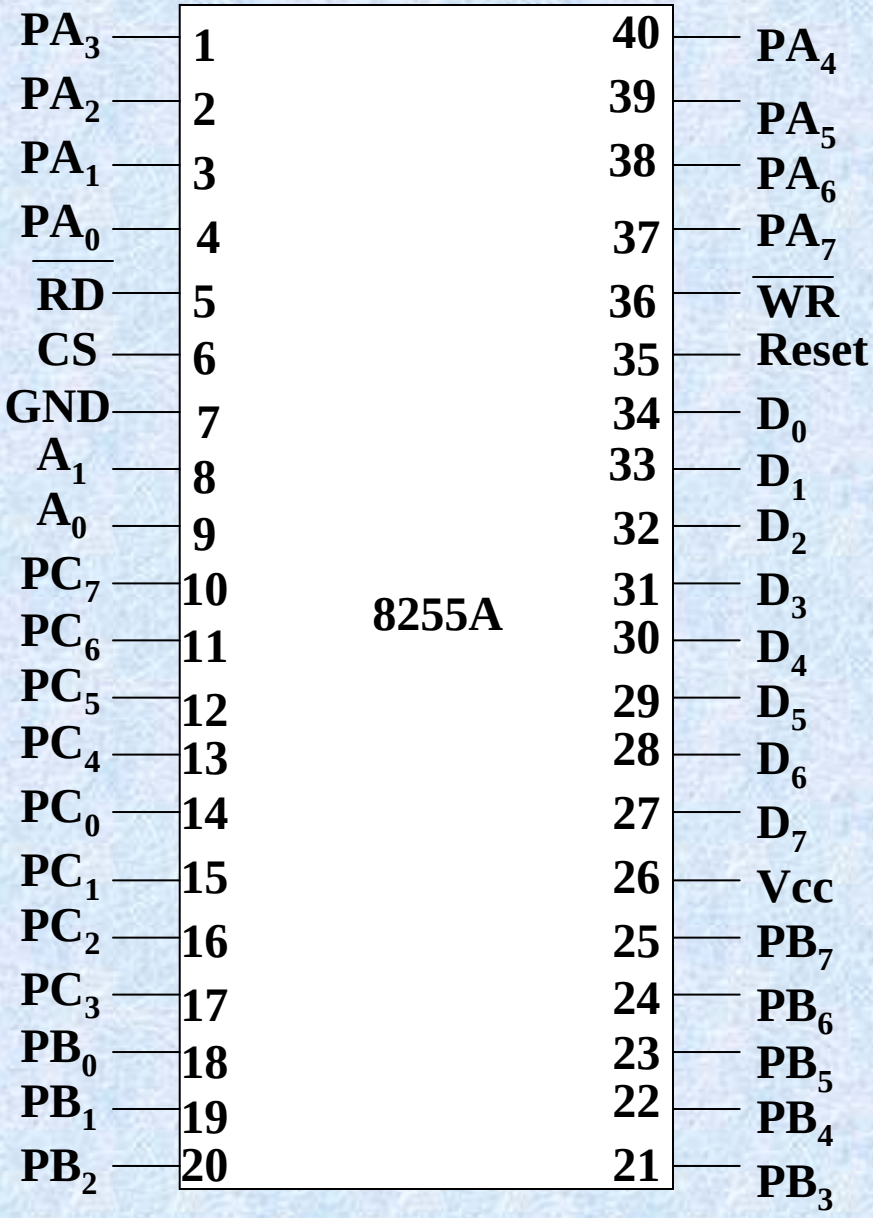
- All these modes can be selected by programming a register internal to 8255 known as CWR.

Modes of Operation of 8255 (cont..)

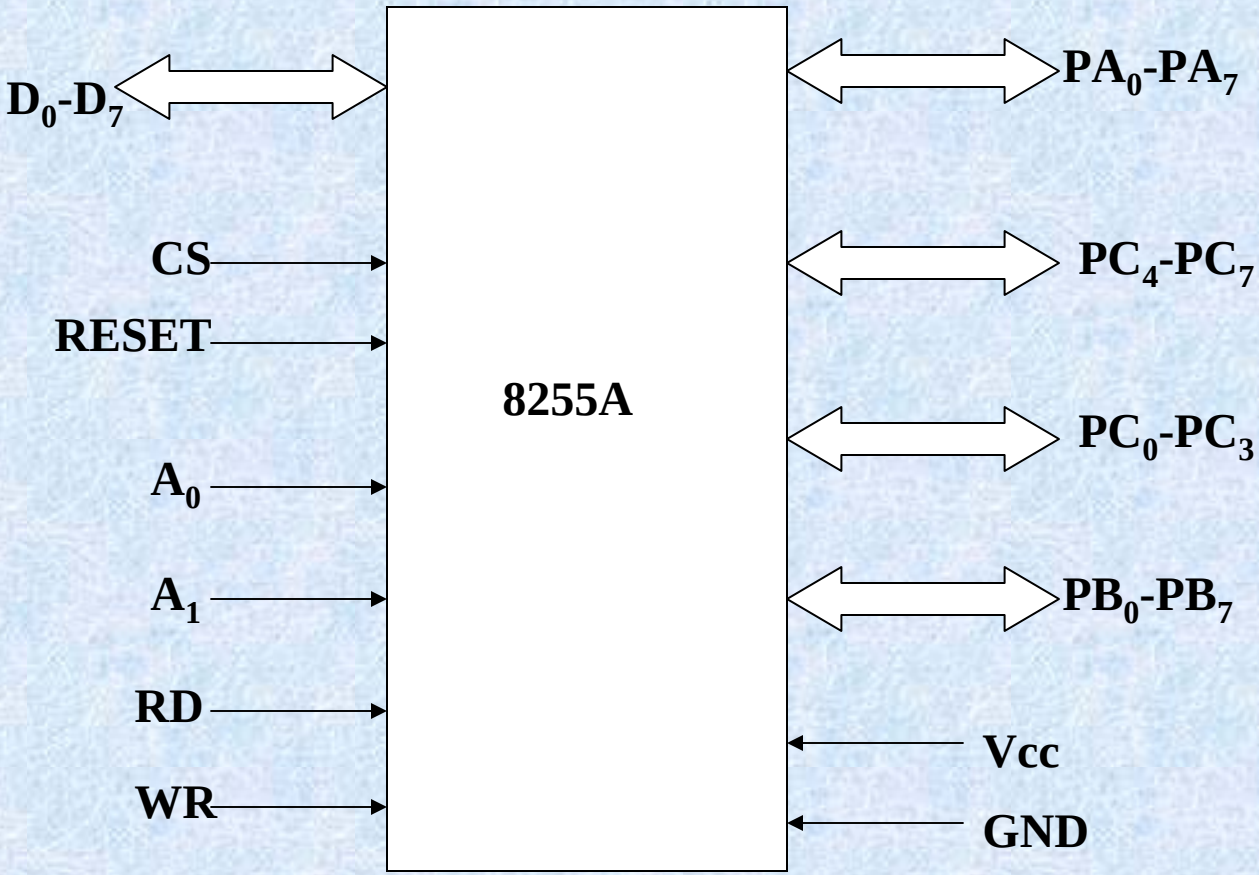
- The control word register has two formats. The first format is valid for I/O modes of operation, i.e. modes 0, mode 1 and mode 2 while the second format is valid for bit set/reset (BSR) mode of operation. These formats are shown in following fig.



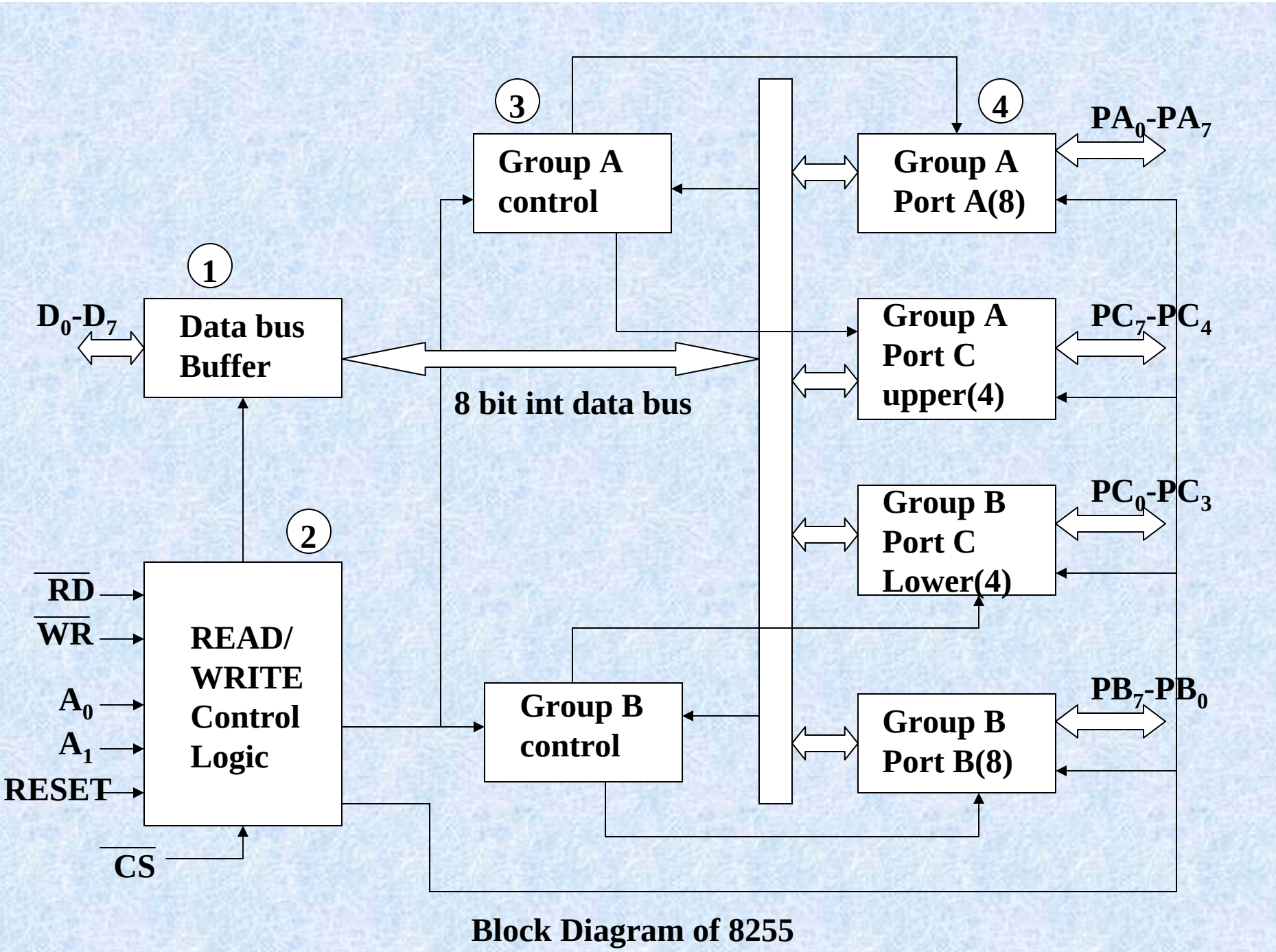
I/O Mode Control Word Register Format and
BSR Mode Control Word Register Format



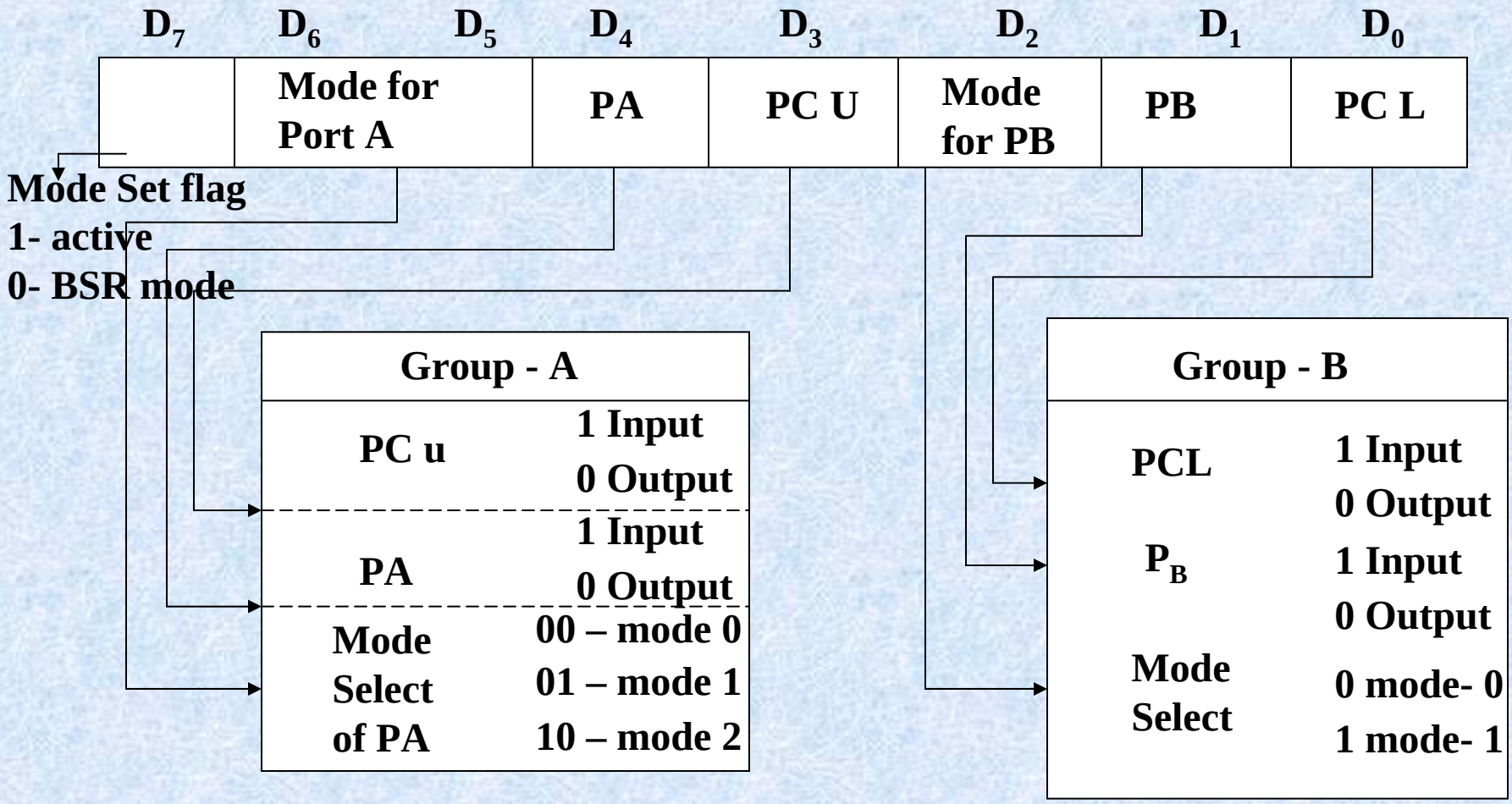
8255A Pin Configuration



Signals of 8255



Block Diagram of 8255



Control Word Format of 8255

Modes of Operation of 8255 (cont..)

b) Mode 1: ***(Strobed input/output mode)*** In this mode the handshaking control the input and output action of the specified port. Port C lines PC_0-PC_2, provide strobe or handshake lines for port B. This group which includes port B and PC_0-PC_2 is called as group B for Strobed data input/output. Port C lines PC_3-PC_5 provide strobe lines for port A. This group including port A and PC_3-PC_5 from group A. Thus port C is utilized for generating handshake signals. The salient features of mode 1 are listed as follows:

Modes of Operation of 8255 (cont..)

1. Two groups – group A and group B are available for strobed data transfer.
2. Each group contains one 8-bit data I/O port and one 4-bit control/data port.
3. The 8-bit data port can be either used as input and output port. The inputs and outputs both are latched.
4. Out of 8-bit port C, PC_0-PC_2 are used to generate control signals for port B and PC_3-PC_5 are used to generate control signals for port A. the lines PC_6, PC_7 may be used as independent data lines.

Modes of Operation of 8255 (cont..)

- The control signals for both the groups in input and output modes are explained as follows:

Input control signal definitions (mode 1):

- **$\overline{\text{STB}}$**(Strobe input) **–** If this lines falls to logic low level, the data available at 8-bit input port is loaded into input latches.
- **IBF** (Input buffer full) **–** If this signal rises to logic 1, it indicates that data has been loaded into latches, i.e. it works as an acknowledgement. IBF is set by a low on $\overline{\text{STB}}$ and is reset by the rising edge of $\overline{\text{RD}}$ input.

Modes of Operation of 8255 (cont..)

- **INTR** (Interrupt request) – This active high output signal can be used to interrupt the CPU whenever an input device requests the service. INTR is set by a high STB pin and a high at IBF pin. INTE is an internal flag that can be controlled by the bit set/reset mode of either $PC_4(INTE_A)$ or $PC_2(INTE_B)$ as shown in fig.
- INTR is reset by a falling edge of RD input. Thus an external input device can be request the service of the processor by putting the data on the bus and sending the strobe signal.

Modes of Operation of 8255 (cont..)

Output control signal definitions (mode 1) :

- **OBF** (Output buffer full) **–** This status signal, whenever falls to low, indicates that CPU has written data to the specified output port. The OBF flip-flop will be set by a rising edge of $\overline{WR}$ signal and reset by a low going edge at the $\overline{ACK}$ input.
- **$\overline{ACK}$** (Acknowledge input) **–** $\overline{ACK}$ signal acts as an acknowledgement to be given by an output device. $\overline{ACK}$ signal, whenever low, informs the CPU that the data transferred by the CPU to the output device through the port is received by the output device.

Modes of Operation of 8255 (cont..)

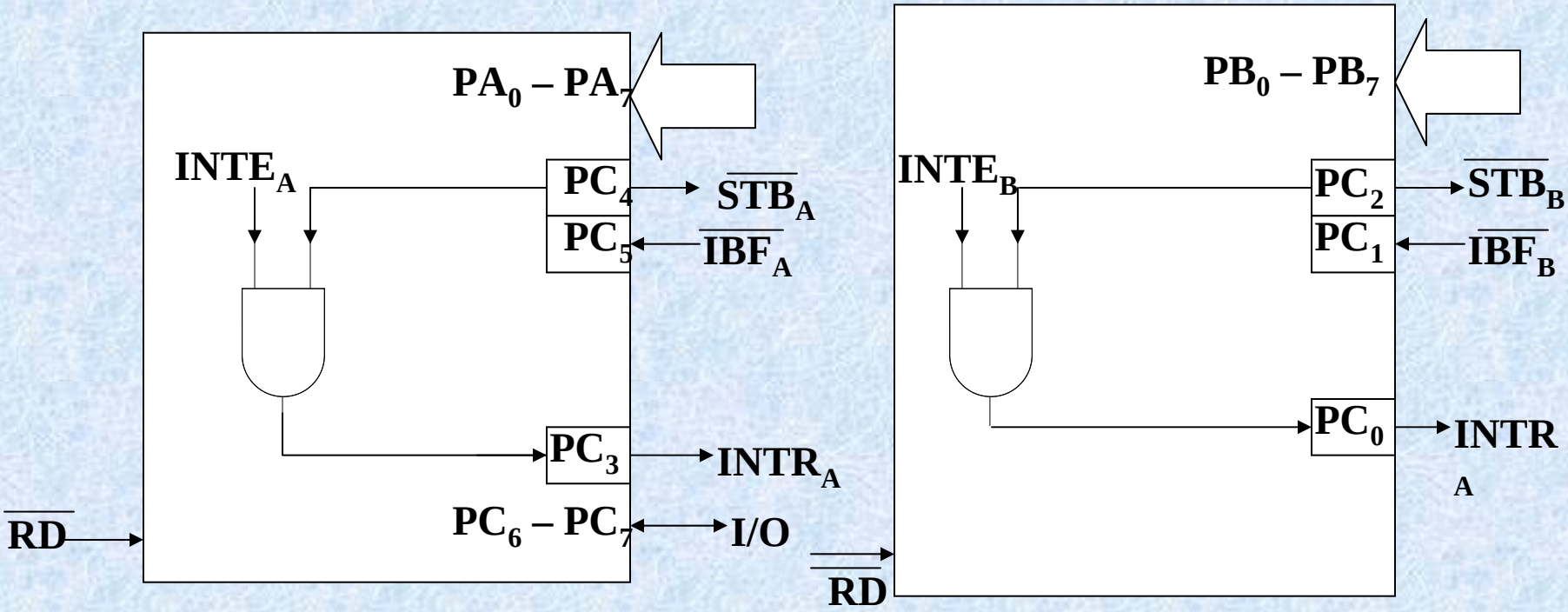
- **INTR** (Interrupt request) – Thus an output signal that can be used to interrupt the CPU when an output device acknowledges the data received from the CPU. INTR is set when ACK, OBF and INTE are 1. It is reset by a falling edge on WR input. The INTEA and INTEB flags are controlled by the bit set-reset mode of PC_6 and PC_2 respectively.

Input control signal definitions in Mode 1

1	0	1	0	1/0	X	X	X
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

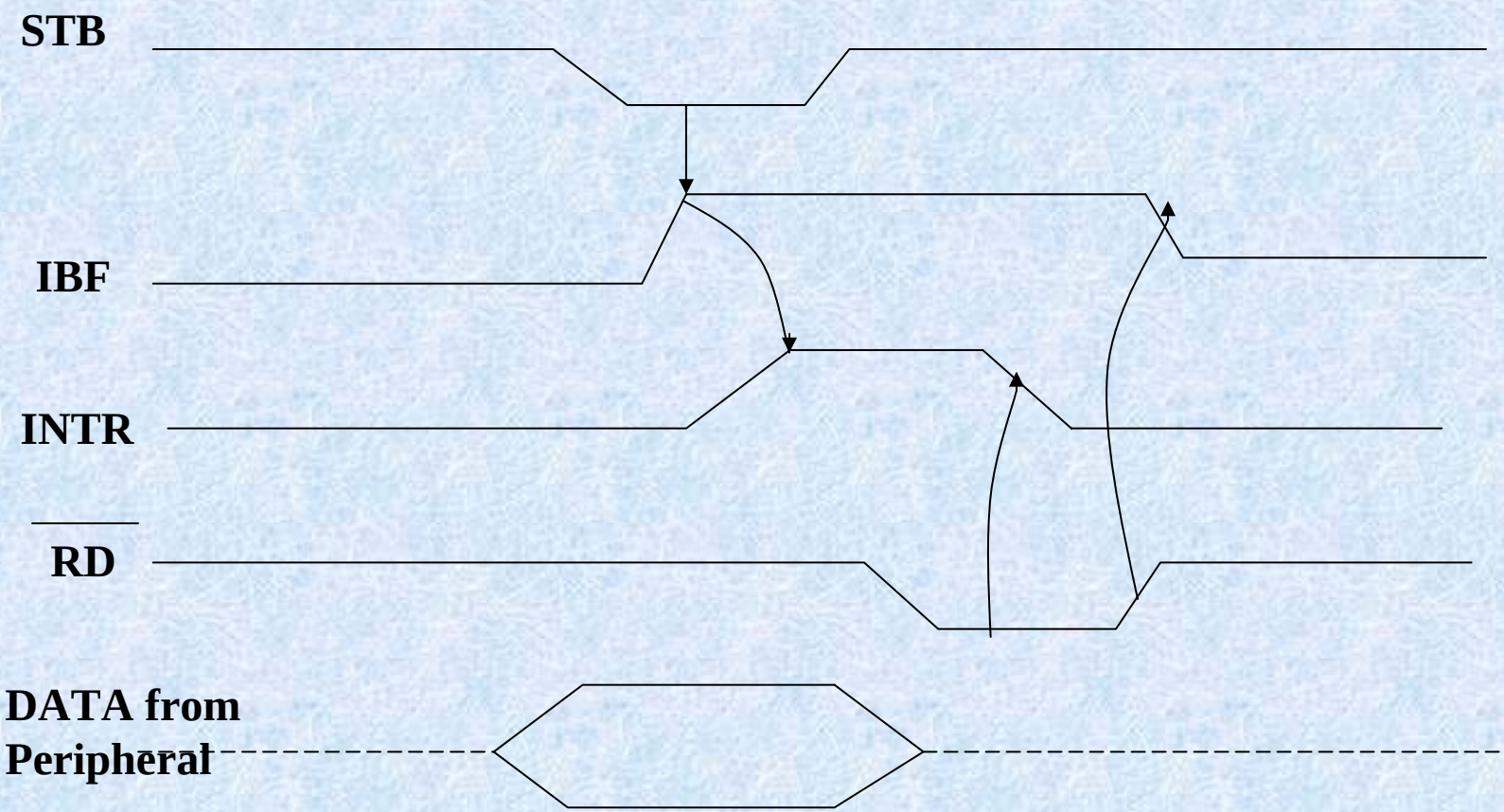
1 - Input
0 - Output
For $PC_6 – PC_7$

1	X	X	X	X	1	1	X
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

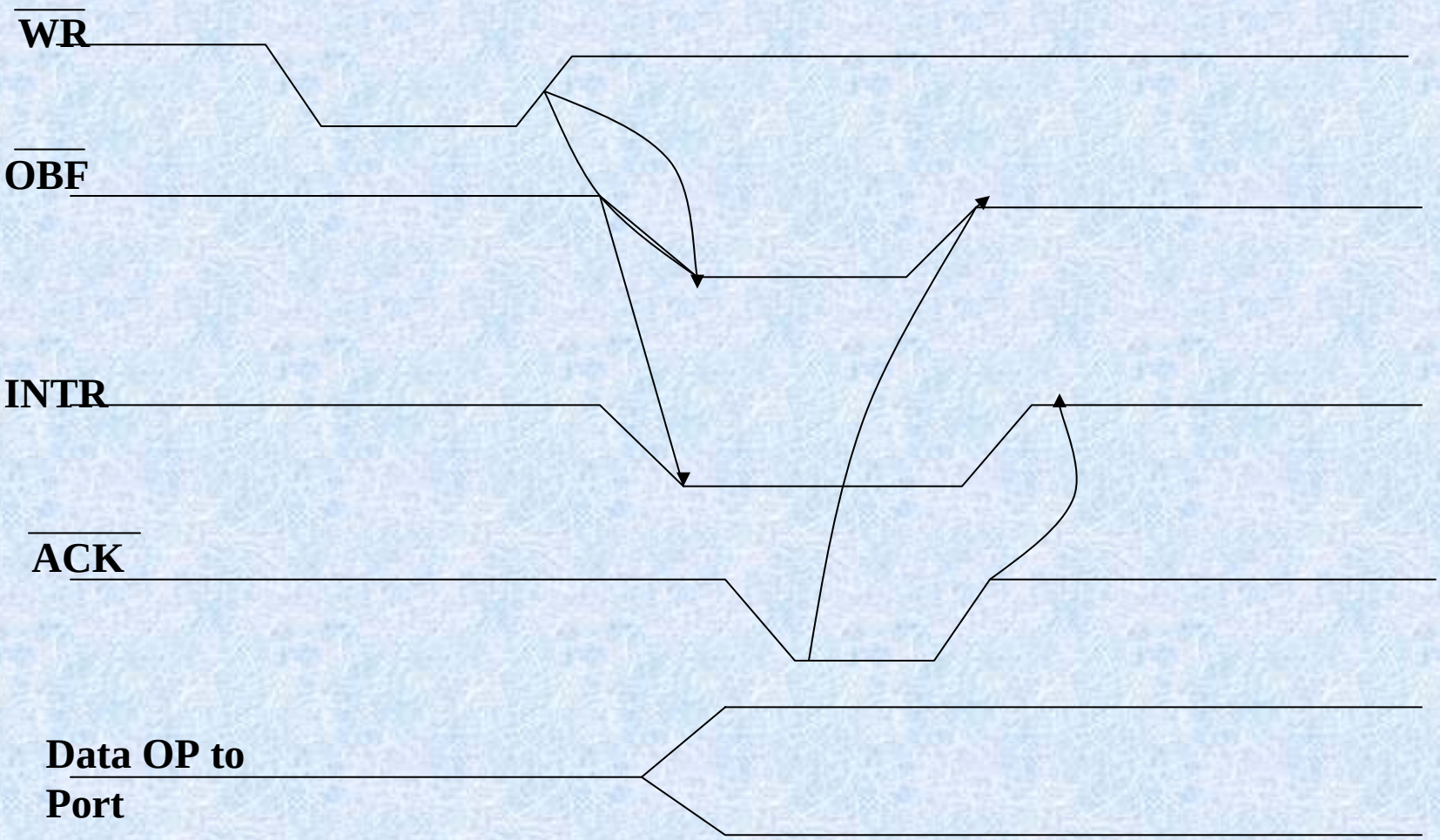


Mode 1 Control Word Group A I/P

Mode 1 Control Word Group B I/P



Mode 1 Strobed Input Data Transfer



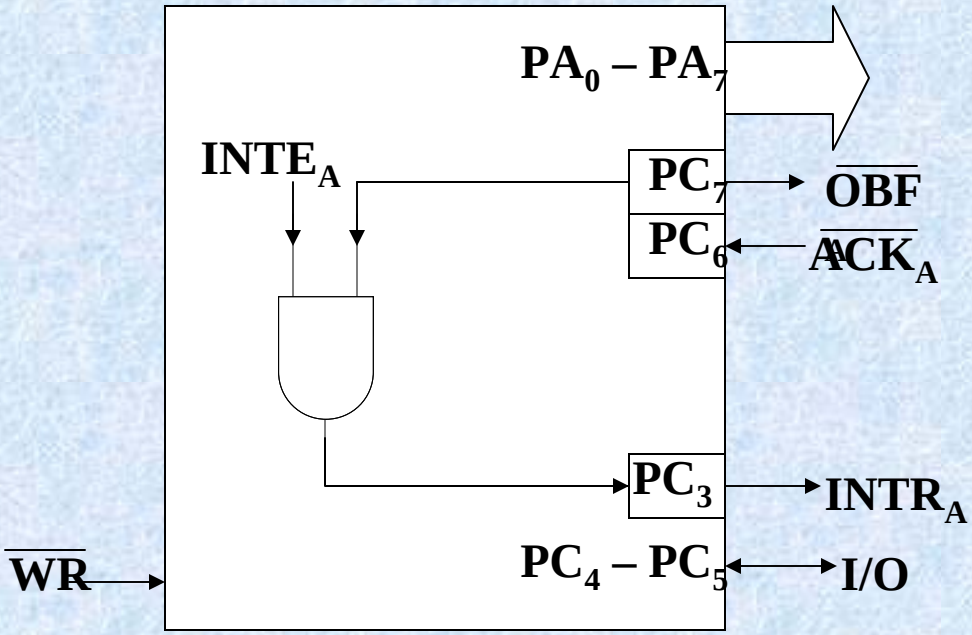
Mode 1 Strobed Data Output

Output control signal definitions Mode 1

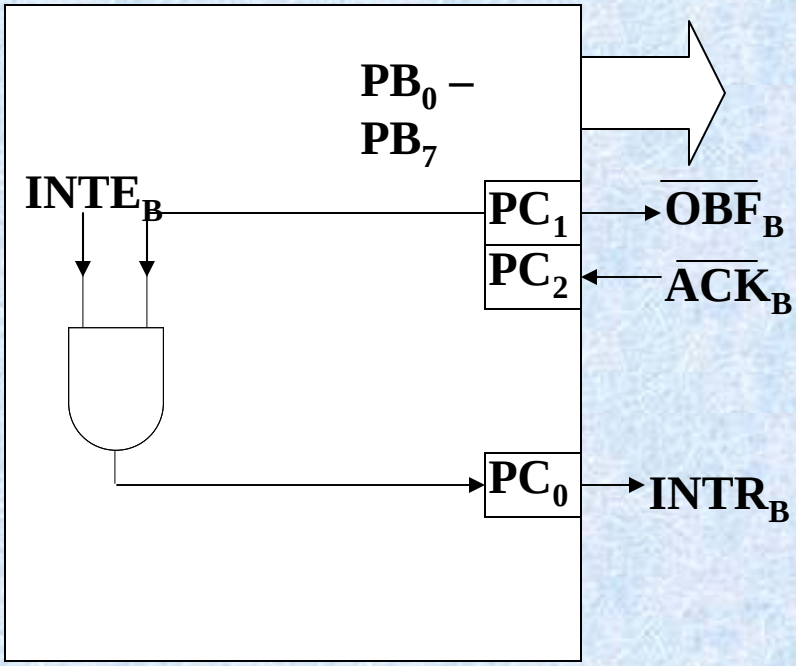
1	0	1	0	1/0	X	X	X
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

1 - Input
0 - Output
For $PC_4 – PC_5$

1	X	X	X	X	1	0	X
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0



Mode 1 Control Word Group A



Mode 1 Control Word Group B

Modes of Operation of 8255 (cont..)

- **Mode 2** ***(Strobed bidirectional I/O):*** This mode of operation of 8255 is also called as strobed bidirectional I/O. This mode of operation provides 8255 with an additional features for communicating with a peripheral device on an 8-bit data bus. Handshaking signals are provided to maintain proper data flow and synchronization between the data transmitter and receiver. The interrupt generation and other functions are similar to mode 1.
- In this mode, 8255 is a bidirectional 8-bit port with handshake signals. The RD and WR signals decide whether the 8255 is going to operate as an input port or output port.

Modes of Operation of 8255 (cont..)

- The Salient features of Mode 2 of 8255 are listed as follows:

1. The single 8-bit port in group A is available.
2. The 8-bit port is bidirectional and additionally a 5-bit control port is available.
3. Three I/O lines are available at port C.(PC_2 – PC_0)
4. Inputs and outputs are both latched.
5. The 5-bit control port C (PC_3-PC_7) is used for generating / accepting handshake signals for the 8-bit data transfer on port A.

Modes of Operation of 8255 (cont..)

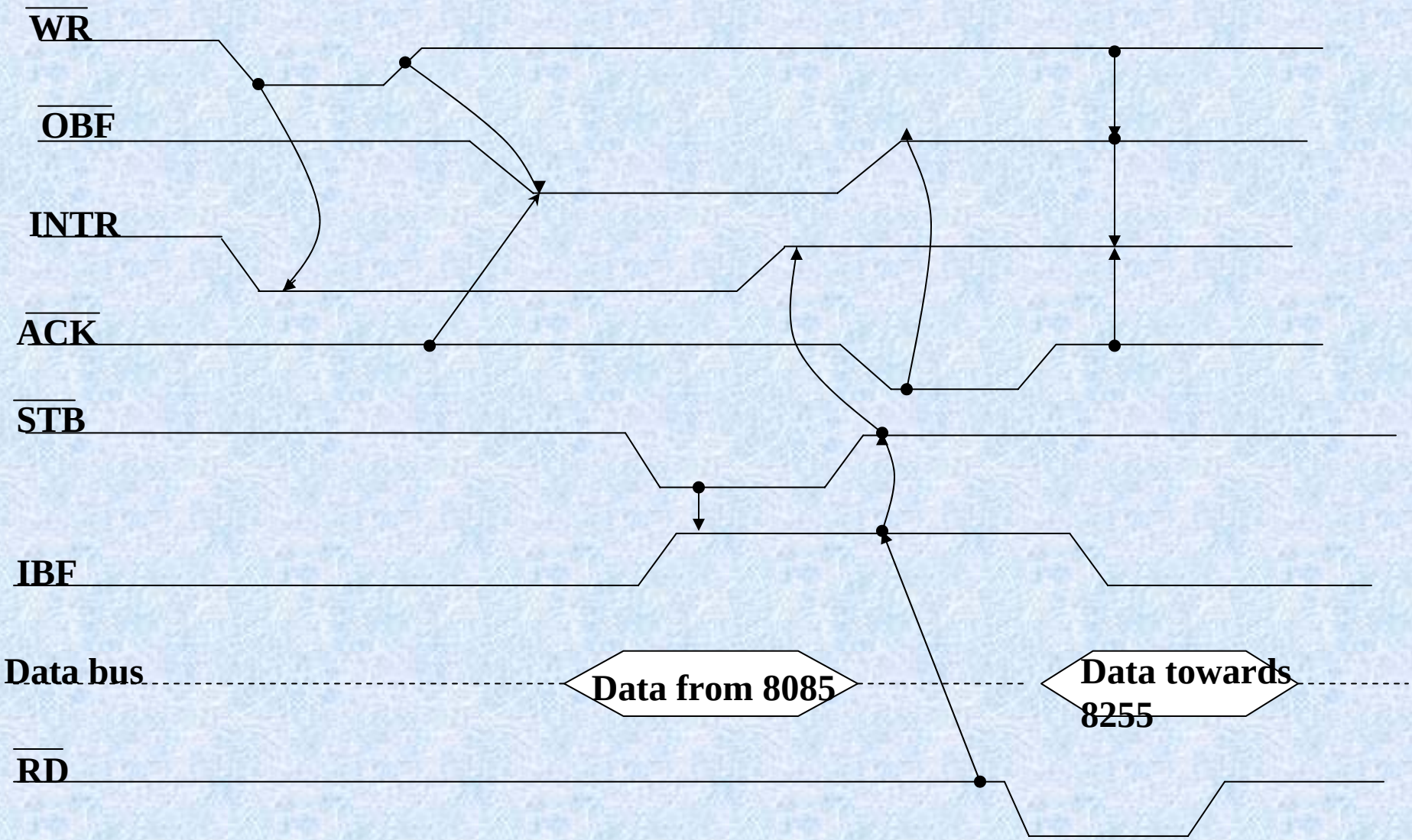
- ***Control signal definitions in mode 2***:
- **INTR –** (Interrupt request) As in mode 1, this control signal is active high and is used to interrupt the microprocessor to ask for transfer of the next data byte to/from it. This signal is used for input (read) as well as output (write) operations.
- ***Control Signals for Output operations***:
- $\overline{\textbf{OBF}}$ (Output buffer full) **–** This signal, when falls to low level, indicates that the CPU has written data to port A.

Modes of Operation of 8255 (cont..)

- $\overline{\textbf{ACK}}$ (Acknowledge) This control input, when falls to logic low level, acknowledges that the previous data byte is received by the destination and next byte may be sent by the processor. This signal enables the internal tristate buffers to send the next data byte on port A.
- **INTE1** (A flag associated with $\overline{\text{OBF}}$) This can be controlled by bit set/reset mode with PC_6.
- ***Control signals for input operations :***
- $\overline{\textbf{STB}}$ (Strobe input) A low on this line is used to strobe in the data into the input latches of 8255.

Modes of Operation of 8255 (cont..)

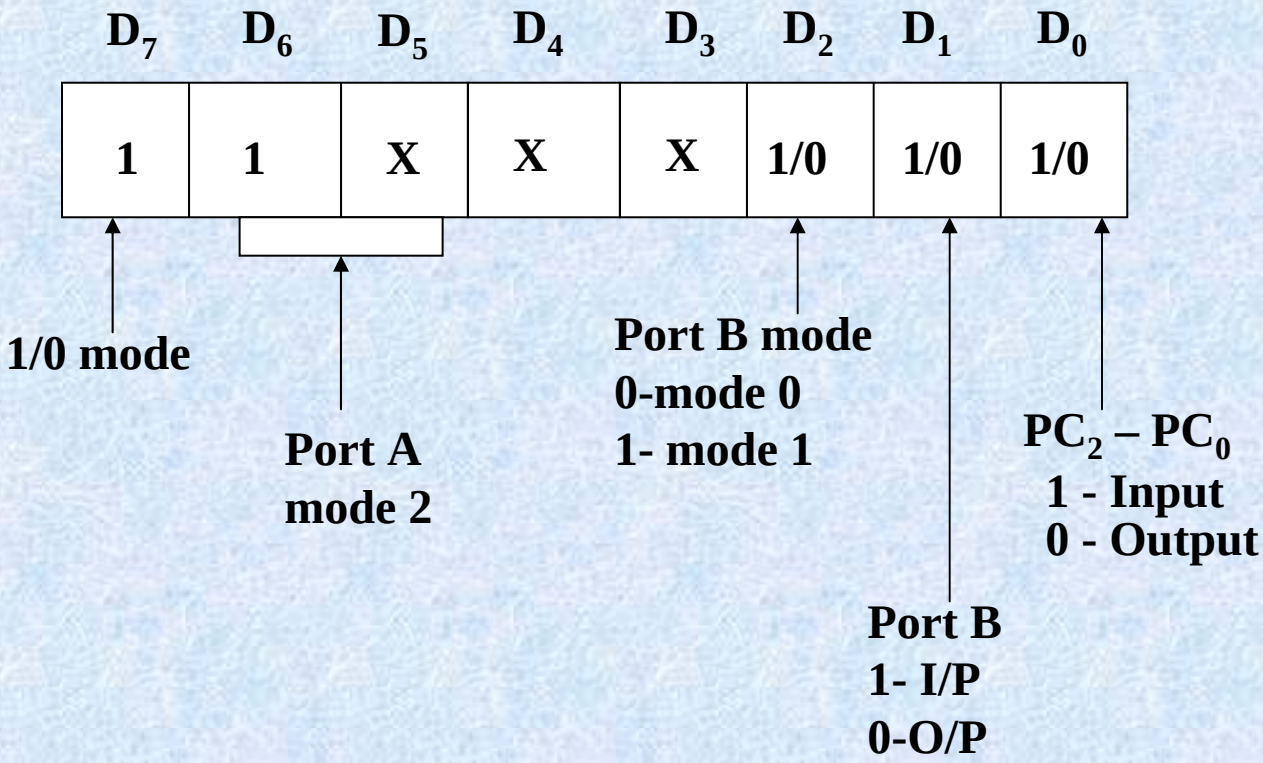
- **IBF** (Input buffer full) When the data is loaded into input buffer, this signal rises to logic ‘1’. This can be used as an acknowledge that the data has been received by the receiver.
- The waveforms in fig show the operation in Mode 2 for output as well as input port.
- Note: $\overline{WR}$ must occur before $\overline{ACK}$ and $\overline{STB}$ must be activated before $\overline{RD}$.



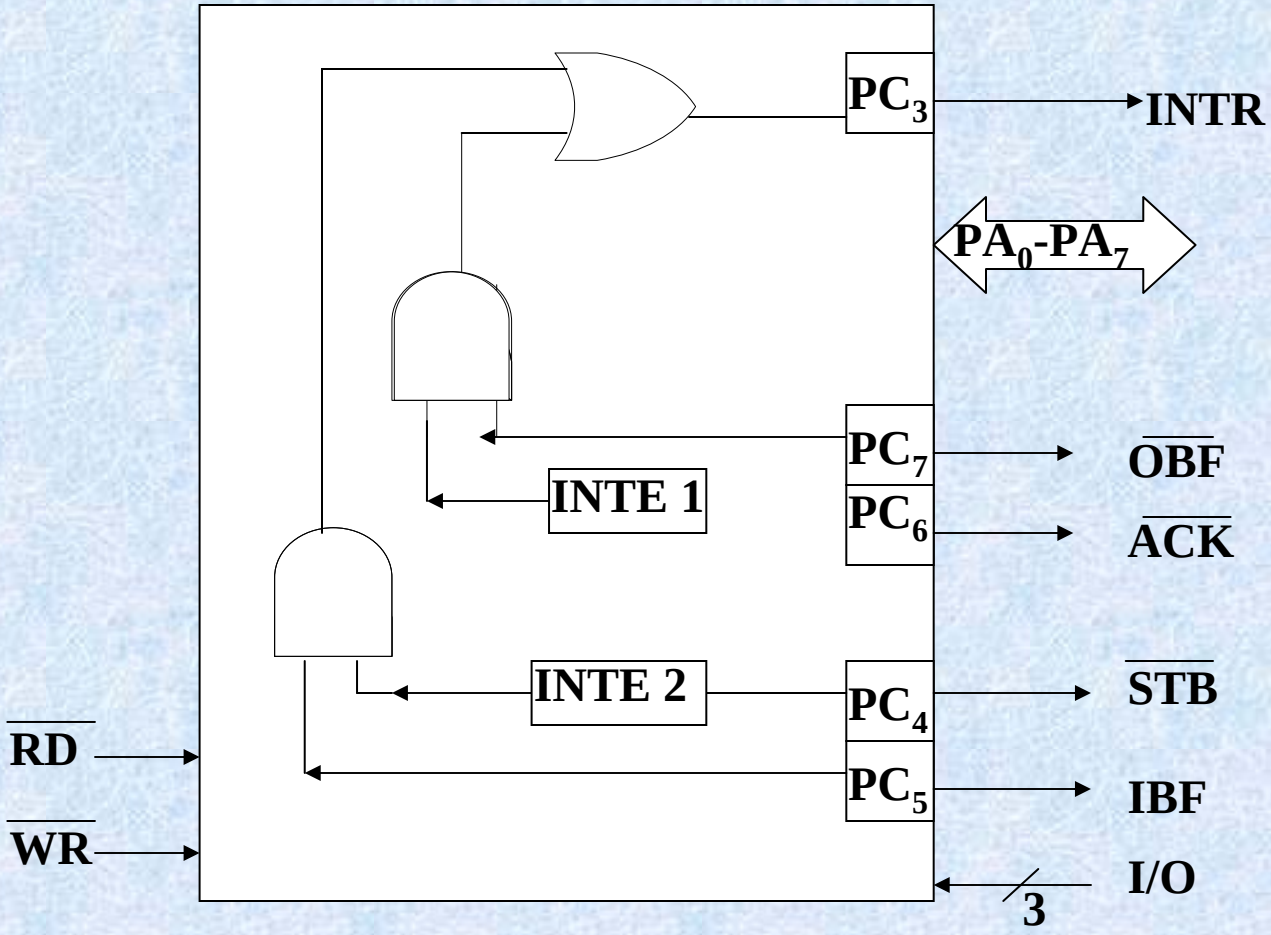
Mode 2 Bidirectional Data Transfer

Modes of Operation of 8255 (cont..)

- The following fig shows a schematic diagram containing an 8-bit bidirectional port, 5-bit control port and the relation of INTR with the control pins. Port B can either be set to Mode 0 or 1 with port A(Group A) is in Mode 2.
- Mode 2 is not available for port B. The following fig shows the control word.
- The INTR goes high only if either IBF, INTE2, STB and RD go high or OBF, INTE1, ACK and WR go high. The port C can be read to know the status of the peripheral device, in terms of the control signals, using the normal I/O instructions.



Mode 2 control word


PC3
INTR
PA0-PA7
PC7
OBF
INTE 1
PC6
ACK
INTE 2
PC4
STB
RD
PC5
IBF
WR
I/O
3

Mode 2 pins

Interfacing Analog to Digital Data Converters

- In most of the cases, the PIO 8255 is used for interfacing the analog to digital converters with microprocessor.
- We have already studied 8255 interfacing with 8086 as an I/O port, in previous section. This section we will only emphasize the interfacing techniques of analog to digital converters with 8255.
- The analog to digital converters is treaded as an input device by the microprocessor, that sends an initialising signal to the ADC to start the analogy to digital data conversation process. The start of conversation signal is a pulse of a specific duration.

Interfacing Analog to Digital Data Converters (cont..)

- The process of analog to digital conversion is a slow process, and the microprocessor has to wait for the digital data till the conversion is over. After the conversion is over, the ADC sends end of conversion EOC signal to inform the microprocessor that the conversion is over and the result is ready at the output buffer of the ADC. These tasks of issuing an SOC pulse to ADC, reading EOC signal from the ADC and reading the digital output of the ADC are carried out by the CPU using 8255 I/O ports.

Interfacing Analog to Digital Data Converters (cont..)

- The time taken by the ADC from the active edge of SOC pulse till the active edge of EOC signal is called as the conversion delay of the ADC.
- It may range any where from a few microseconds in case of fast ADC to even a few hundred milliseconds in case of slow ADCs.
- The available ADC in the market use different conversion techniques for conversion of analog signal to digitals. Successive approximation techniques and dual slope integration techniques are the most popular techniques used in the integrated ADC chip.

Interfacing Analog to Digital Data Converters (cont..)

- General algorithm for ADC interfacing contains the following steps:

1. Ensure the stability of analog input, applied to the ADC.
2. Issue start of conversion pulse to ADC
3. Read end of conversion signal to mark the end of conversion processes.
4. Read digital data output of the ADC as equivalent digital output.

Interfacing Analog to Digital Data Converters (cont..)

- Analog input voltage must be constant at the input of the ADC right from the start of conversion till the end of the conversion to get correct results. This may be ensured by a sample and hold circuit which samples the analog signal and holds it constant for a specific time duration. The microprocessor may issue a hold signal to the sample and hold circuit.
- If the applied input changes before the complete conversion process is over, the digital equivalent of the analog input calculated by the ADC may not be correct.

Interfacing Analog to Digital Data Converters (cont..)

ADC 0808/0809 :

- The analog to digital converter chips 0808 and 0809 are 8-bit CMOS, successive approximation converters. This technique is one of the fast techniques for analog to digital conversion. The conversion delay is 100µs at a clock frequency of 640 KHz, which is quite low as compared to other converters. These converters do not need any external zero or full scale adjustments as they are already taken care of by internal circuits. These converters internally have a 3:8 analog multiplexer so that at a time eight different analog conversion by using address lines -

Interfacing Analog to Digital Data Converters (cont..)

ADD A, ADD B, ADD C. Using these address inputs, multichannel data acquisition system can be designed using a single ADC. The CPU may drive these lines using output port lines in case of multichannel applications. In case of single input applications, these may be hardwired to select the proper input.

- There are unipolar analog to digital converters, i.e. they are able to convert only positive analog input voltage to their digital equivalent. These chips do no contain any internal sample and hold circuit.

Analog I/P selected	Address lines		
	C	B	A
I/P_0	0	0	0
I/P_1	0	0	1
I/P_2	0	1	0
I/P_3	0	1	1
I/P_4	1	0	0
I/P_5	1	0	1
I/P_6	1	1	0
I/P_7	1	1	1

Fig

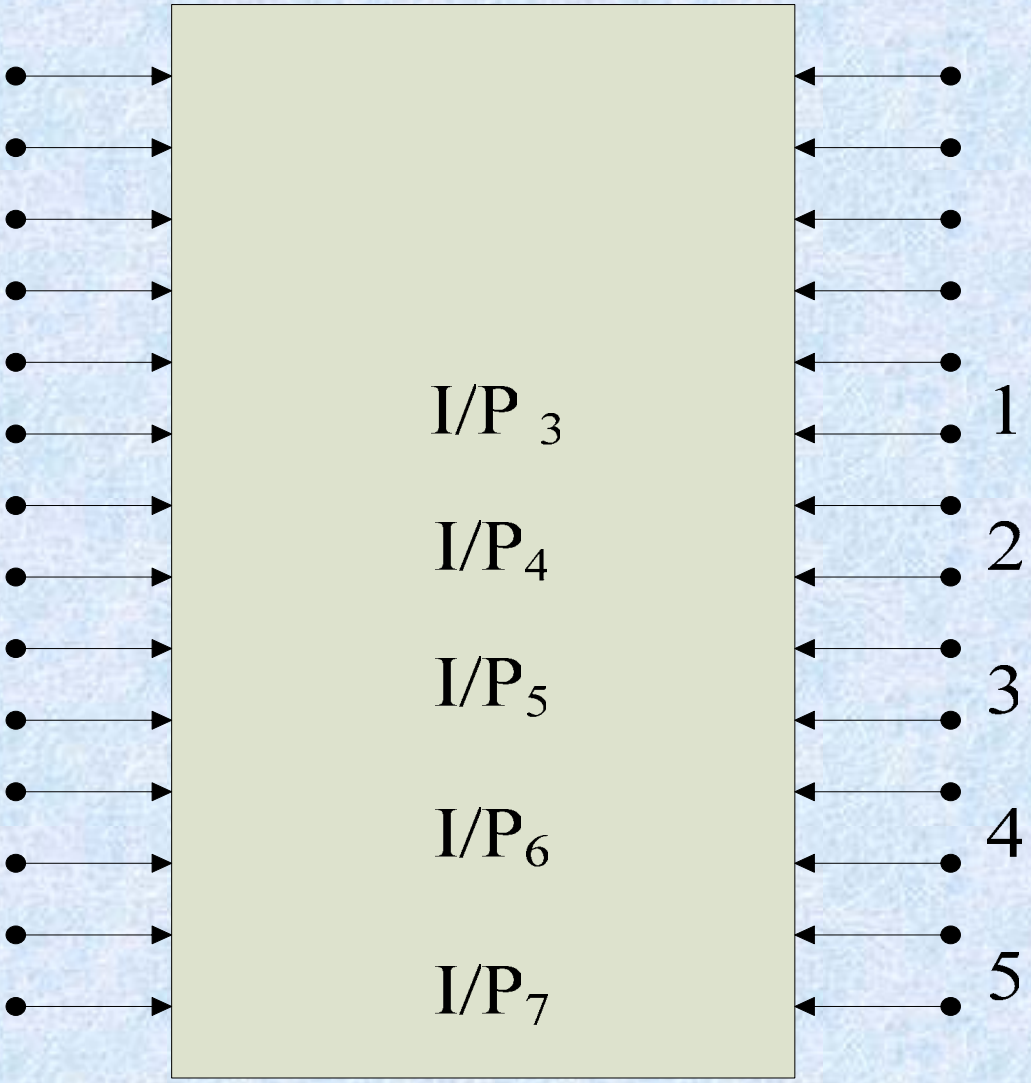
Interfacing Analog to Digital Data Converters (cont..)

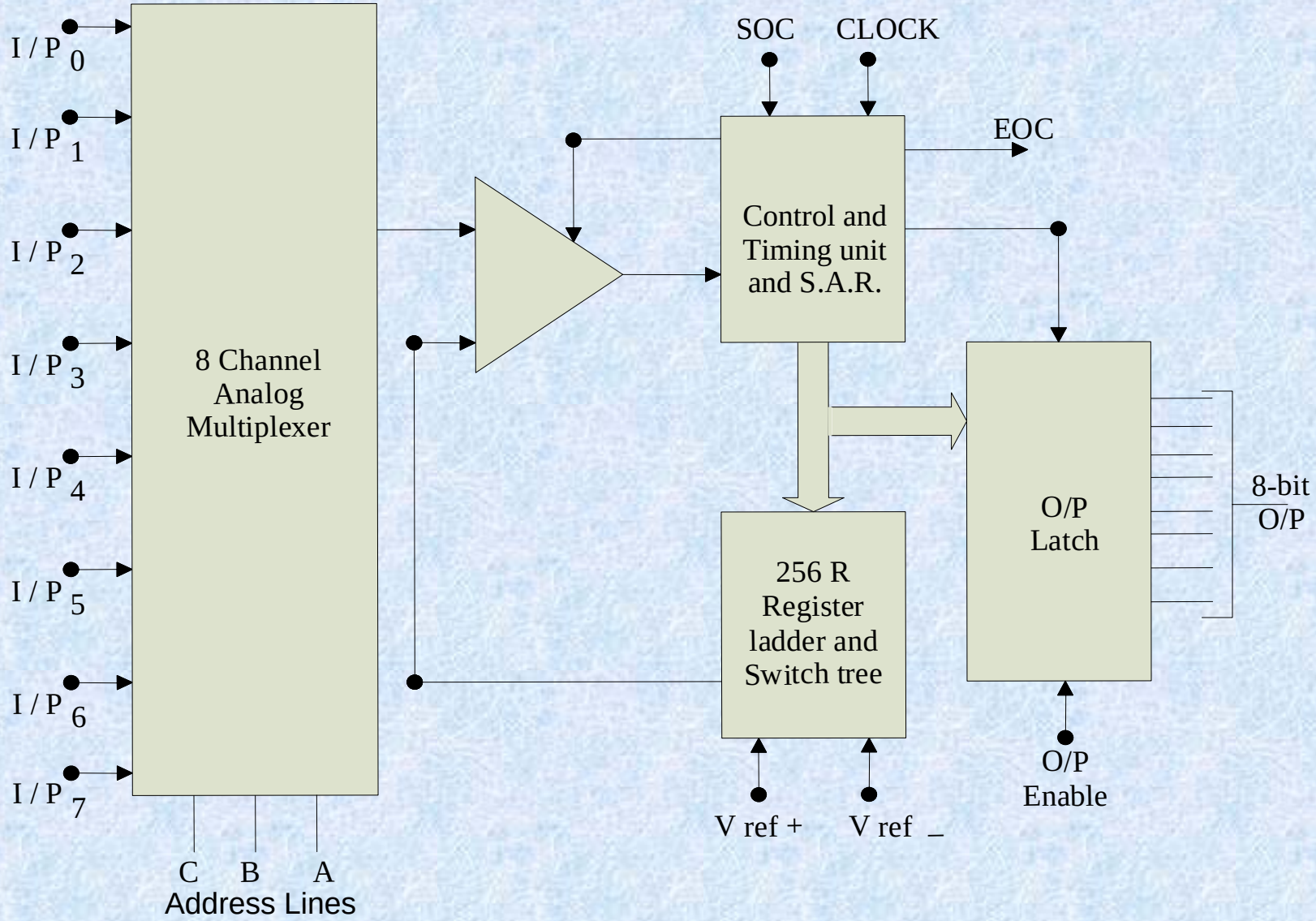
- If one needs a sample and hold circuit for the conversion of fast signal into equivalent digital quantities, it has to be externally connected at each of the analog inputs.

- Vcc — Supply pins +5V
- GND — GND
- Vref + — Reference voltage positive +5 Volts maximum.
- Vref – — Reference voltage negative 0Volts minimum.

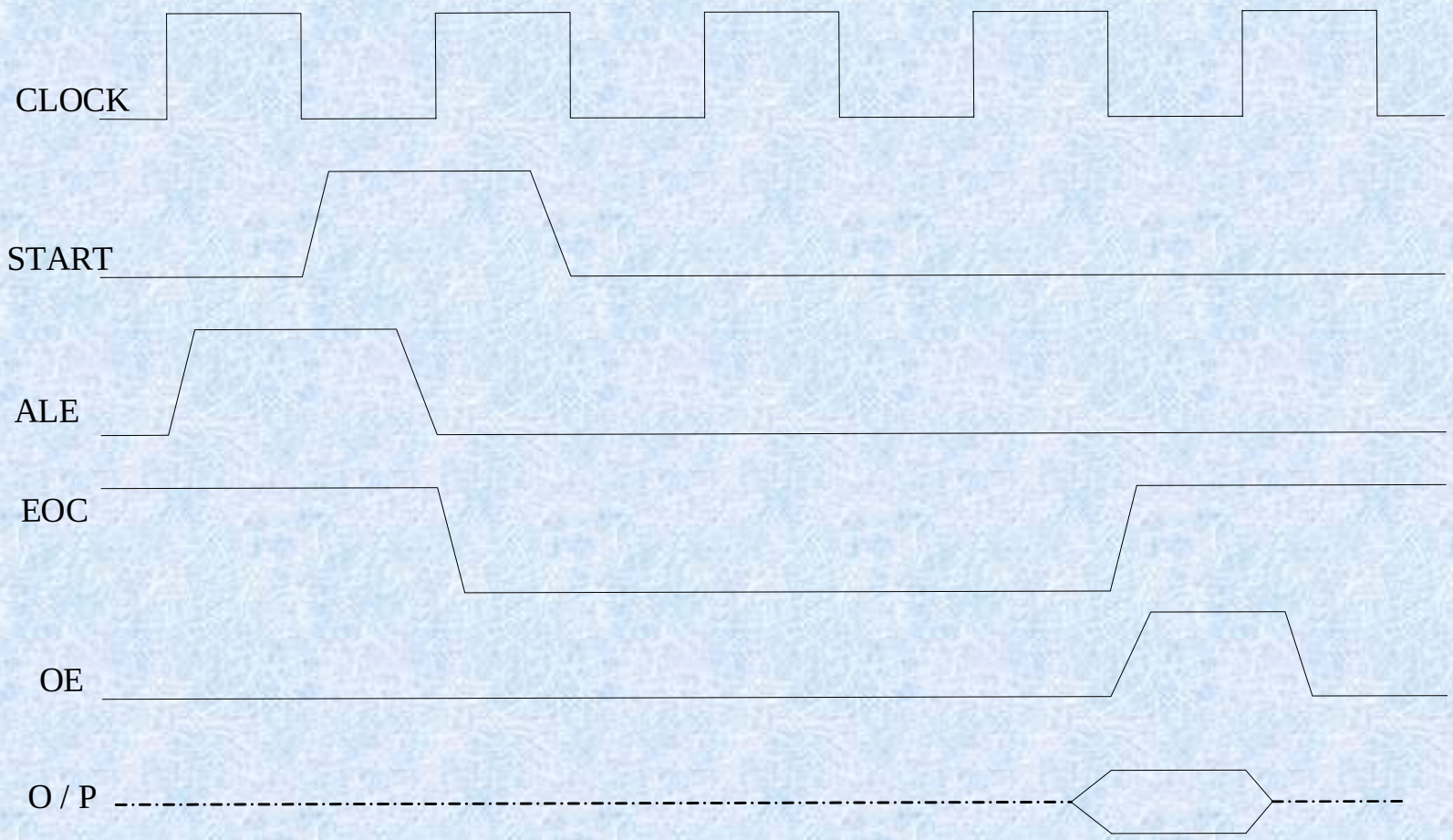
Interfacing Analog to Digital Data Converters (cont..)

- I/P_0 –I/P_7 — Analog inputs
- ADD A,B,C — Address lines for selecting analog inputs.
- $O_7 – O_0$ — Digital 8-bit output with O_7 MSB and O_0 LSB
- SOC — Start of conversion signal pin
- EOC — End of conversion signal pin
- OE — Output latch enable pin, if high enables output
- CLK — Clock input for ADC


I/P 3
I/P4
I/P5
I/P6
I/P7
SOC
EOC
1
2
3
4
5
6
7
ADC
ADC



Block Diagram of ADC 0808 / 0809



Timing Diagram of ADC 0808

Interfacing Analog to Digital Data Converters (cont..)

- ***Example:*** Interfacing ADC 0808 with 8086 using 8255 ports. Use port A of 8255 for transferring digital data output of ADC to the CPU and port C for control signals. Assume that an analog input is present at I/P_2 of the ADC and a clock input of suitable frequency is available for ADC.
- **Solution**: The analog input I/P_2 is used and therefore address pins A,B,C should be 0,1,0 respectively to select I/P_2. The OE and ALE pins are already kept at +5V to select the ADC and enable the outputs. Port C upper acts as the input port to receive the EOC signal while port C lower acts as the output port to send SOC to the ADC.

Interfacing Analog to Digital Data Converters (cont..)

- Port A acts as a 8-bit input data port to receive the digital data output from the ADC. The 8255 control word is written as follows:

  $D_7\ D_6\ D_5\ D_4\ D_3\ D_2\ D_1\ D_0$

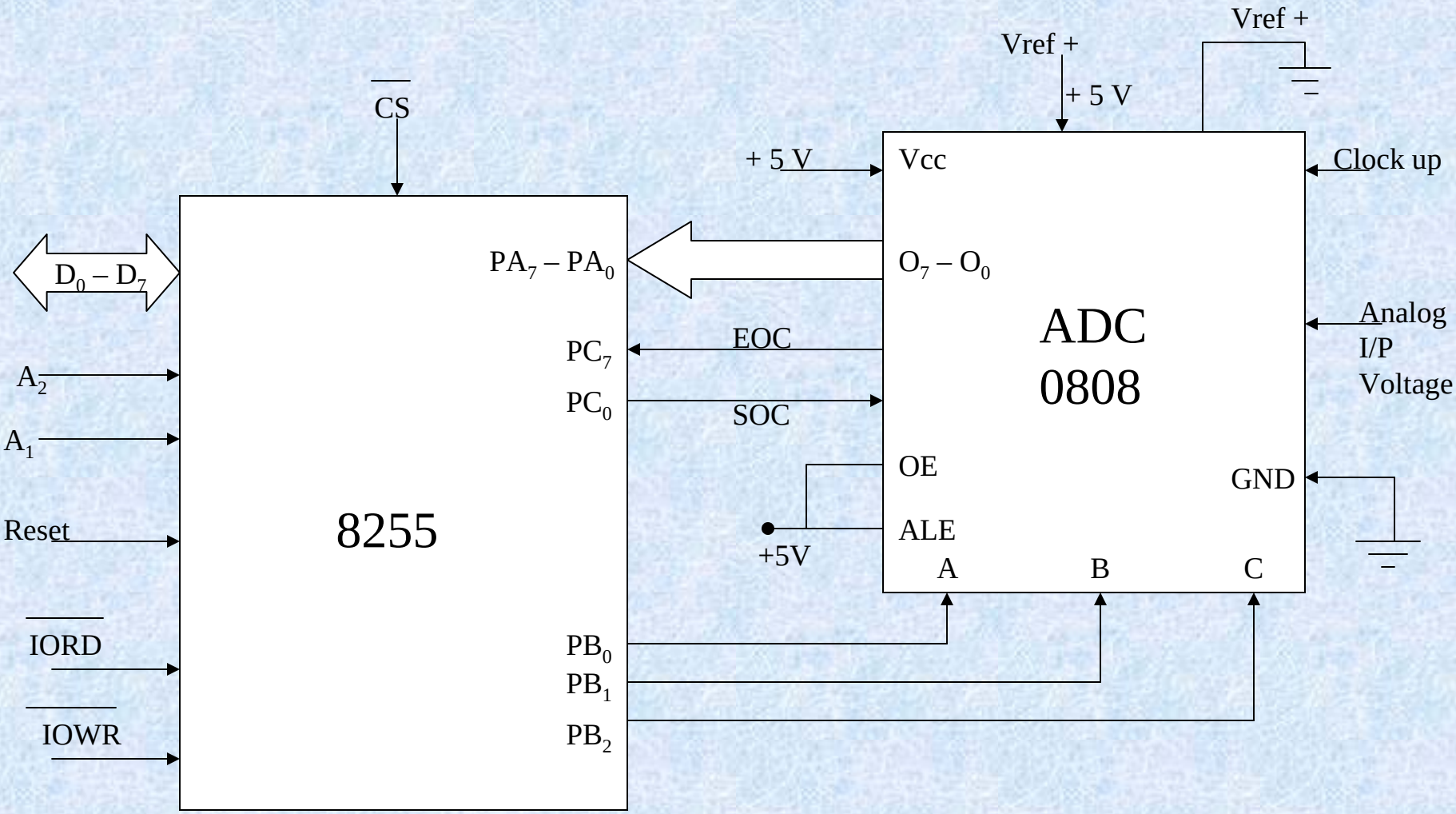
  1 0 0 1 1 0 0 0

- The required ALP is as follows:

```
MOV     AL, 98h     ;initialise 8255 as
OUT     CWR, AL     ;discussed above.
MOV     AL, 02h     ;Select I/P2 as analog
OUT     Port B, AL  ;input.
```

Interfacing Analog to Digital Data Converters (cont..)

```
        MOV   AL, 00h       ;Give start of conversion
        OUT   Port C, AL    ; pulse to the ADC
        MOV   AL, 01h
        OUT   Port C, AL
        MOV   AL, 00h
        OUT   Port C, AL
WAIT:   IN    AL, Port C    ;Check for EOC by
        RCR                 ; reading port C upper and
        JNC   WAIT          ;rotating through carry.
        IN    AL, Port A    ;If EOC, read digital equivalent
                            ;in AL
        HLT                 ;Stop.
```

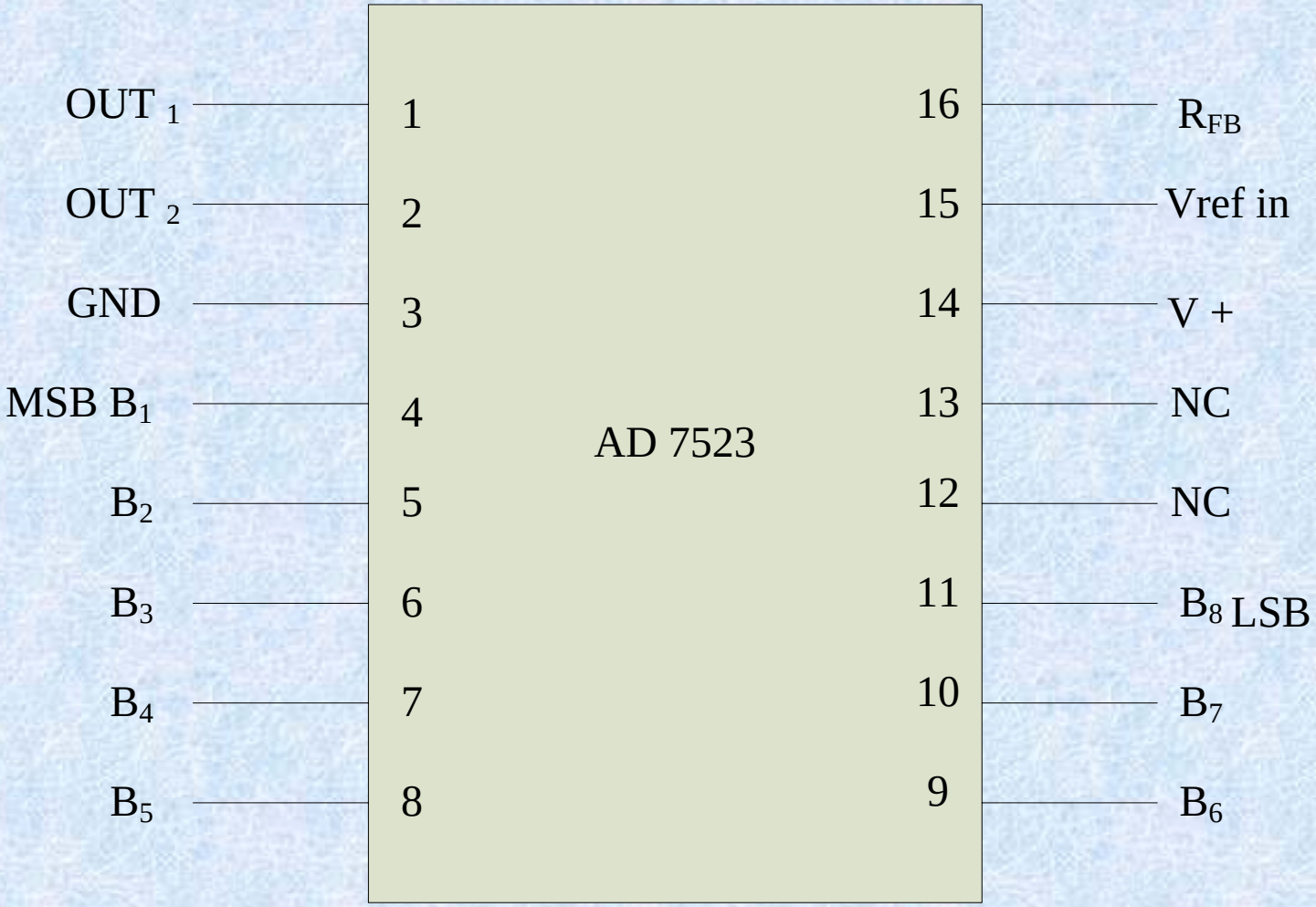


Interfacing 0808 with 8086

Interfacing Digital To Analog Converters (cont..)

INTERFACING DIGITAL TO ANALOG CONVERTERS: The digital to analog converters convert binary number into their equivalent voltages. The DAC find applications in areas like digitally controlled gains, motors speed controls, programmable gain amplifiers etc.

AD 7523 8-bit Multiplying DAC : This is a 16 pin DIP, multiplying digital to analog converter, containing R-2R ladder for D-A conversion along with single pole double thrown NMOS switches to connect the digital inputs to the ladder.



Pin Diagram of AD 7523

Interfacing Analog to Digital Data Converters (cont..)

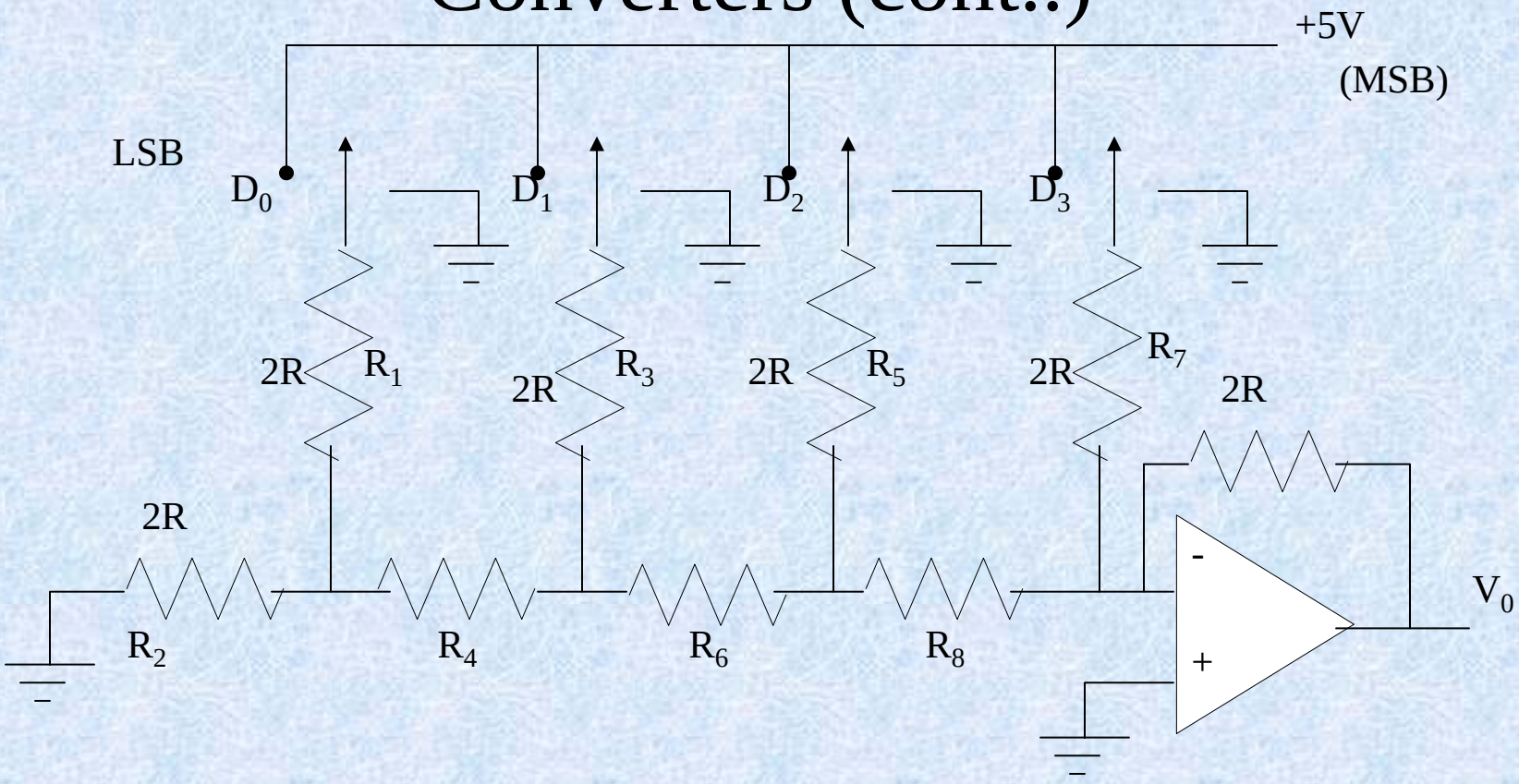


Fig:

Interfacing Digital To Analog Converters (cont..)

- The pin diagram of AD7523 is shown in fig the supply range is from +5V to +15V, while Vref may be any where between -10V to +10V. The maximum analog output voltage will be any where between -10V to +10V, when all the digital inputs are at logic high state.
- Usually a zener is connected between OUT1 and OUT2 to save the DAC from negative transients. An operational amplifier is used as a current to voltage converter at the output of AD to convert the current out put of AD to a proportional output voltage.

Interfacing Digital To Analog Converters (cont..)

- It also offers additional drive capability to the DAC output. An external feedback resistor acts to control the gain. One may not connect any external feedback resistor, if no gain control is required.
- ***EXAMPLE***: Interfacing DAC AD7523 with an 8086 CPU running at $8MH_Z$ and write an assembly language program to generate a sawtooth waveform of period 1ms with Vmax 5V.
- Solution: Fig shows the interfacing circuit of AD 74523 with 8086 using 8255. program gives an ALP to generate a sawtooth waveform using circuit.

Example (cont..)

```
ASSUME    CS:CODE
CODE      SEGMENT
START:    MOV AL,80h        ;make all ports output
          OUT   CW, AL
AGAIN:    MOV   AL,00h      ;start voltage for ramp
BACK :    OUT   PA, AL
          INC   AL
          CMP   AL, 0FFh
          JB    BACK
          JMP   AGAIN
          CODE  ENDS
          END   START
```

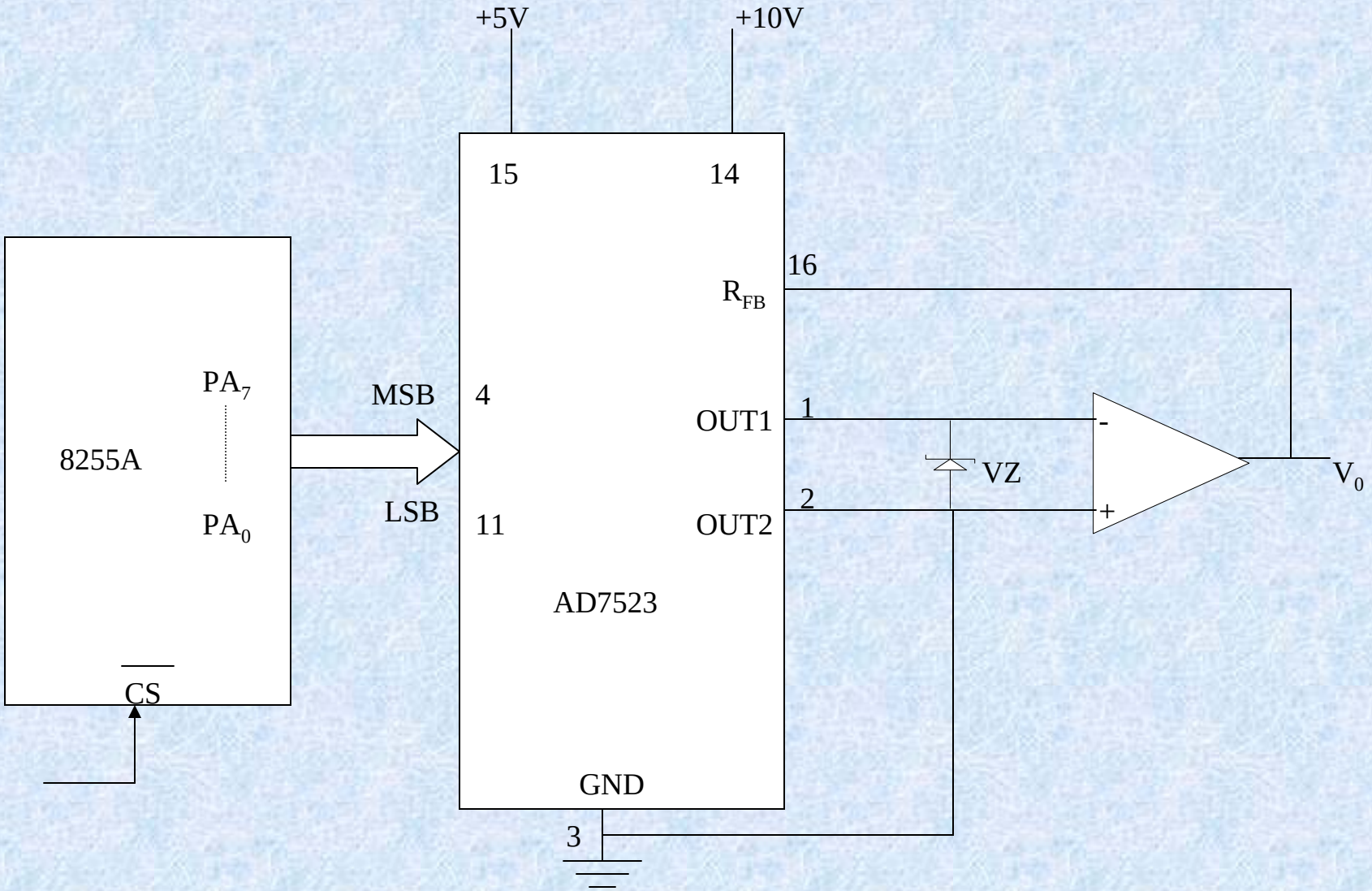


Fig: Interfacing of AD7523

Interfacing Analog to Digital Data Converters (cont..)

- In the above program, port A is initialized as the output port for sending the digital data as input to DAC. The ramp starts from the 0V (analog), hence AL starts with 00H. To increment the ramp, the content of AL is increased during each execution of loop till it reaches F2H.
- After that the saw tooth wave again starts from 00H, i.e. 0V(analog) and the procedure is repeated. The ramp period given by this program is precisely 1.000625 ms. Here the count F2H has been calculated by dividing the required delay of 1ms by the time required for the execution of the loop once. The ramp slope can be controlled by calling a controllable delay after the OUT instruction.

Stepper Motor Interfacing:

- A stepper motor is a device used to obtain an accurate position control of rotating shafts. It employs rotation of its shaft in terms of steps, rather than continuous rotation as in case of AC or DC motors. To rotate the shaft of the stepper motor, a sequence of pulses is needed to be applied to the windings of the stepper motor, in a proper sequence.
- The number of pulses required for one complete rotation of the shaft of the stepper motor is equal to its number of internal teeth on its rotor. The stator teeth and the rotor teeth lock with each other to fix a position of the shaft.
- With a pulse applied to the winding input, the rotor rotates by one teeth position or an angle x. The angle x may be calculated as:

$$X=360^{0}/\text{no. of rotor teeth}$$

- After the rotation of the shaft through angel x, the rotor locks itself with the next tooth in the sequence on the internal surface of stator.
- The internal schematic of a typical stepper motor with four windings is shown in fig.1.

- The stepper motors have been designed to work with digital circuits. Binary level pulses of 0-5V are required at its winding inputs to obtain the rotation of shafts. The sequence of the pulses can be decided, depending upon the required motion of the shaft.

- Fig.1 shows a typical winding arrangement of the stepper motor.
- Fig.2 shows conceptual positioning of the rotor teeth on the surface of rotor, for a six teeth rotor.

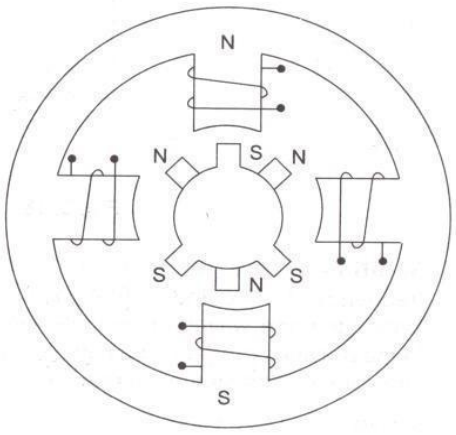


Fig.1 Internal schematic of a four winding stepper motor

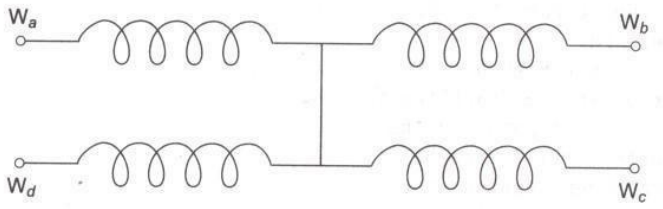


Fig.2 Winding arrangement of a stepper motor.

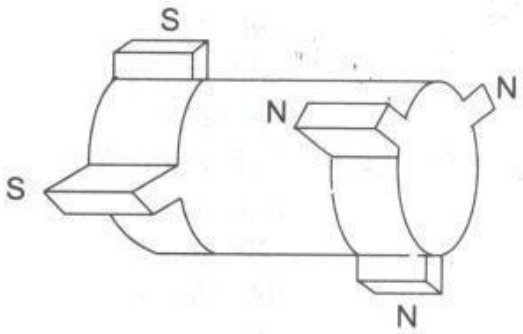


Fig.3 Stepper motor rotor

- The circuit for interfacing a winding Wn with an I/O port is given in fig.4. Each of the windings of a stepper motor needs this circuit for its interfacing with the output port. A typical stepper motor may have parameters like torque 3 Kg-cm, operating voltage 12V, current rating 0.2 A and a step angle 1.8^0 i.e. 200 steps/revolution (number of rotor teeth).
- A simple schematic for rotating the shaft of a stepper motor is called a wave scheme. In this scheme, the windings Wa, Wb, Wc and Wd are applied with the required voltages pulses, in a cyclic fashion. By reversing the sequence of excitation, the direction of rotation of the stepper motor shaft may be reversed.
- Table.1 shows the excitation sequences for clockwise and anticlockwise rotations. Another popular scheme for rotation of a stepper motor shaft applies pulses to two successive windings at a time but these are shifted only by one position at a time. This scheme for rotation of stepper motor shaft is shown in table2.

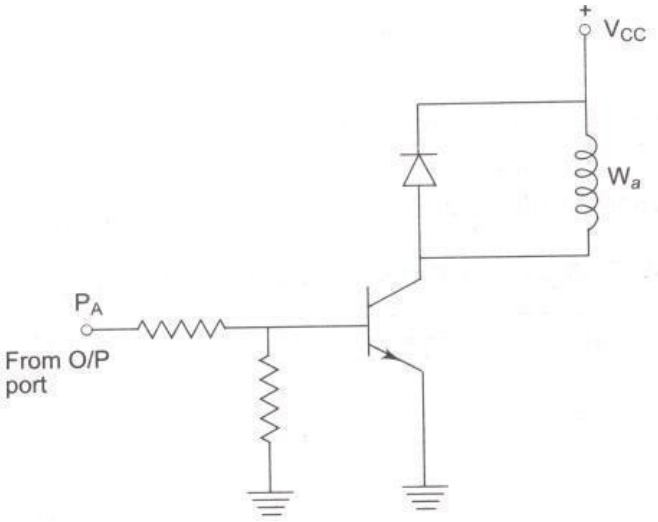


Fig.4 interfacing stepper motor winding.

Table.1 Excitation sequence of a stepper motor using wave switching scheme.

Motion	step	A	B	C	D
Clock Wise Direction	1	1	0	0	0
	2	0	1	0	0
	3	0	0	1	0
	4	0	0	0	1
	5	1	0	0	0
Anti clock wise Direction	1	1	0	0	0
	2	0	0	0	1
	3	0	0	1	0
	4	0	1	0	0
	5	1	0	0	0

Table.2 An alternative scheme for rotating stepper motor shaft

Motion	step	A	B	C	D
Clock wise Direction	1	0	0	1	1
	2	0	1	1	0
	3	1	1	0	0
	4	1	0	0	1
	5	0	0	1	1
Anti clock wise Direction	1	0	0	1	1
	2	1	0	0	1
	3	1	1	0	0
	4	0	1	1	0
	5	0	0	0	0

8527 DMA Controller

The 8527 controller has four independent channels each of which contains an address register and a counter. The counter decrements as each byte transfer occur, and forces termination of the DMA operation after the last transfer. The controller increments the address registers after each operation, so that successive data transfers are made at contiguous ascending addresses. The arbiter resolves conflicts among the channels for access to memory. Two methods have been used in this chip to make the chip useful in a variety of different applications. In one mode the channels have a fixed priority and conflicts are resolved according to the priority, for example, Channel 0 has highest priority and Channel 3 lowest. The second mode is a rotating priority scheme in which priority rankings are the four cycle shifts of 0-1-2-3, when a channel is granted access to the bus the priority ranking shifts cyclically to place the channel in the lowest priority position for the next arbitration cycle.

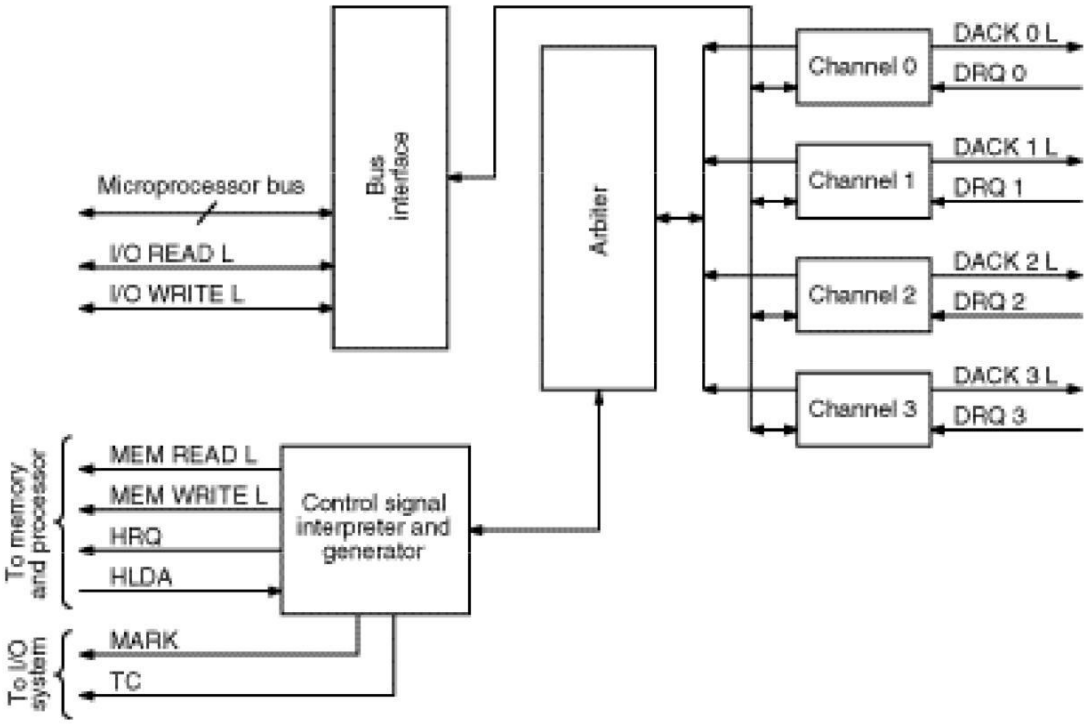


Figure 5-4: Structure of the i8527 DMA controller

The chip has four signals associated with the READ and WRITE operation. MEM READ L and MEM WRITE L are signals produced by DMA controller to exercise memory. The two signals I/O READ L and I/O WRITE L are bidirectional, they are inputs from the microprocessor when the microprocessor sends commands to the 8257 and reads back the 8257 status. During the I/O operation these signals are output from the 8257 and are functionally opposite to the memory signals. The 8257 takes control of the bus by exercising HALT (HRQ) and receives back the "go-ahead" signal on HALT ACKNOWLEDGE (HLDA).

Two signals produced by the DMA controller can be used by the I/O port to assist in controlling the transfer process. One signal TC--terminal count--is asserted during the last cycle of a DMA block. This can be used to describe a DMA mode on an I/O port or to reset the port's internal state to indicate the end of a transfer. The second--MARK--is inserted when the remaining count on a channel became a multiple of 128--providing a convenient timing signal for an external device.

Register Organization of 8257

Table *8257 Register Selection*

Register	*Byte*	*Address Inputs*				*F/L*	*BI-Directional Data Bus*							
		A_3	A_2	A_1	A_0		D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CH-0 DMA Address	LSB	0	0	0	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
	MSB	0	0	0	0	1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
CH-0 Terminal Count	LSB	0	0	0	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
	MSB	0	0	0	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
CH-1 DMA Address	LSB	0	0	1	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
	MSB	0	0	1	0	1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
CH-1 Terminal Count	LSB	0	0	1	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
	MSB	0	0	1	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
CH-2 DMA Address	LSB	0	1	0	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
	MSB	0	1	0	0	1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
CH-2 Terminal Count	LSB	0	1	0	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
	MSB	0	1	0	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
CH-3 DMA Address	LSB	0	1	1	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
	MSB	0	1	1	0	1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
CH-3 Terminal Count	LSB	0	1	1	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
	MSB	0	1	1	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
MODE SET (Programme only)	—	1	0	0	0	0	AL	TCS	EW	RP	EN3	EN2	EN1	EN0
STATUS (Read only)	—	1	0	0	0	0	0	0	0	UP	TC3	TC2	TC1	TC0

The 8257 performs the DMA operation over four independent DMA channels.Each of four channels of 8257 has a pair of two 16-bit registers, viz. DMA address register and terminal count register.There are two common registers for all the channels, namely, mode set register and status register. Thus there are a total of ten registers. The CPU selects one of these ten registers using address lines Ao-A3. Table shows how the Ao-A3 bits may be used for selecting one of these registers.

DMA Address Register

Each DMA channel has one DMA address register. The function of this register is to store the address of the starting memory location, which will be accessed by the DMA channel. Thus the starting address of the memory block which will be accessed by the device is first loaded in the DMA address register of the channel. The device that wants to transfer data over a DMA

channel, will access the block of the memory with the starting address stored in the DMA Address Register.

Terminal Count Register

Each of the four DMA channels of 8257 has one terminal count register (TC). This 16-bit register isused for ascertaining that the data transfer through a DMA channel ceases or stops after the required number of DMA cycles. The low order 14-bits of the terminal count register are initialised with the binary equivalent of the number of required DMA cycles minus one.After each DMA cycle, the terminal count register content will be decremented by one and finally it becomes zero after the required number of DMA cycles are over. The bits 14 and 15 of this register indicate the type of the DMA operation (transfer). If the device wants to write data into the memory, the DMA operation is called DMA write operation. Bit 14 of the register in this case will be set to one and bit 15 will be set to zero. Table gives detail of DMA operation selection and corresponding bit configuration of bits14 and 15 of the TC register.

Table *DMA Operation Selection Using A_{15}/RD and A_{14}/WR*

Bit 15	*Bit 14*	*Type of DMA Operation*
0	0	Verify DMA Cycle
0	1	Write DMA Cycle
1	0	Read DMA Cycle
1	1	(Illegal)

Mode Set Register

The mode set register is used for programming the 8257 as per the requirements of the system. The function of the mode set register is to enable the DMA channels individually and also to set the various modes of operation.The DMA channel should not be enabled till the DMA address register and the terminal count register contain valid information, otherwise, an unwanted DMA request may initiate a DMA cycle, probably destroying the valid memory data. The bits Do-D3 enable one of the four DMA channels of 8257. for example, if Do is '1', channel 0 is enabled. If bit 4 is set, rotating priority is enabled, otherwise, the normal, i.e. fixed priority is enabled.

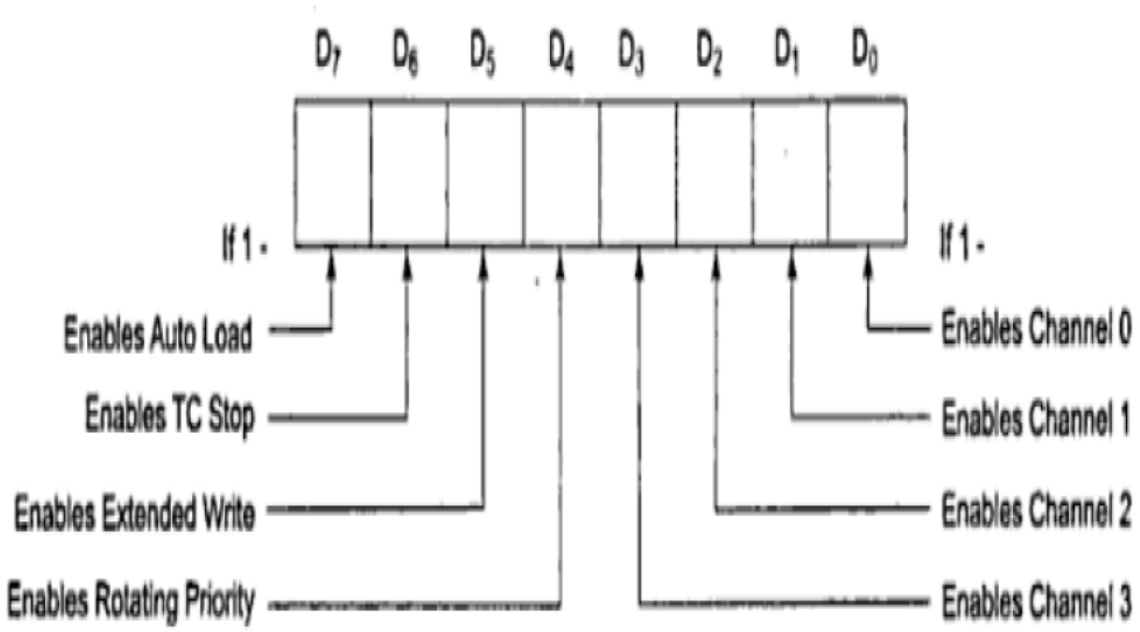
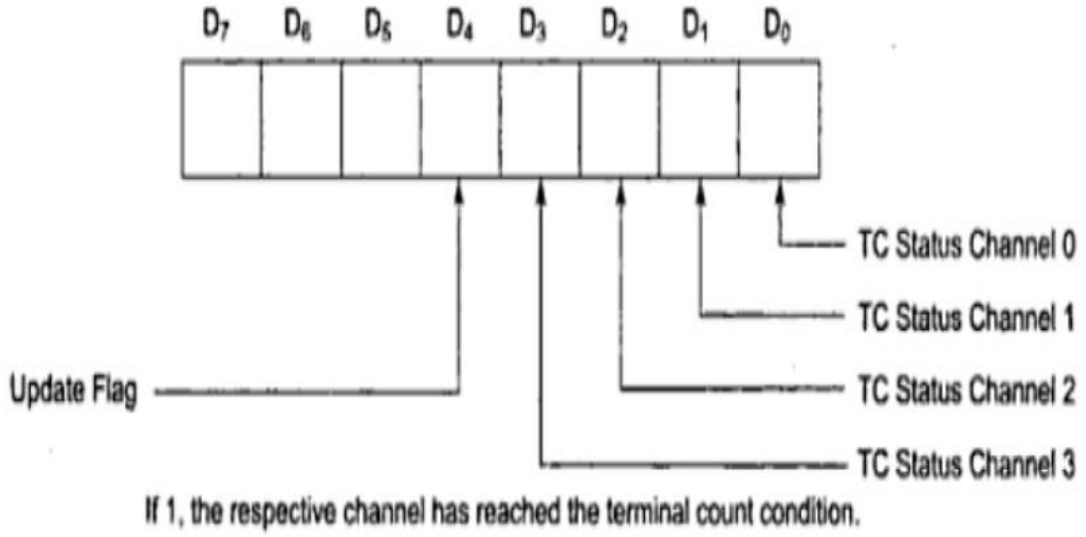


Fig. *Bit Definitions of the Mode Set Register*

If the TC STOP bit is set, the selected channel is disabled after the terminal count condition is reached, and it further prevents any DMA cycle on the channel. To enable the channel again, this bit must be reprogrammed. If the TC STOP bit is programmed to be zero, the channel is not disabled, even after the count reaches zero and further request are allowed on the same channel. The auto load bit, if set, enables channel 2 for the repeat block chaining operations, without immediate software intervention between the two successive blocks. The channel 2 registers are used as usual, while the channel 3 registers are used to store the block reinitialisation parameters, i.e. the DMA starting address and terminal count. After the first block is transferred using DMA, the channel 2 registers are reloaded with the corresponding channel 3 registers for the next block transfer, if the update flag is set. The extended write bit, if set to '1', extends the duration of MEMW and IOW signals by activating them earlier, this is useful in interfacing the peripherals with different access times. If the peripheral is not accessed within the stipulated time, it is expected to give the 'NOT
READY' indication to 8257, to request it to add one or more wait states in the DMA CYCLE. The mode set register can only be written into.

Status Register

The status register of 8257 is shown in figure. The lower order 4-bits of this register contain the terminal count status for the four individual channels. If any of these bits is set, it indicates that the specific channel has reached the terminal count condition.

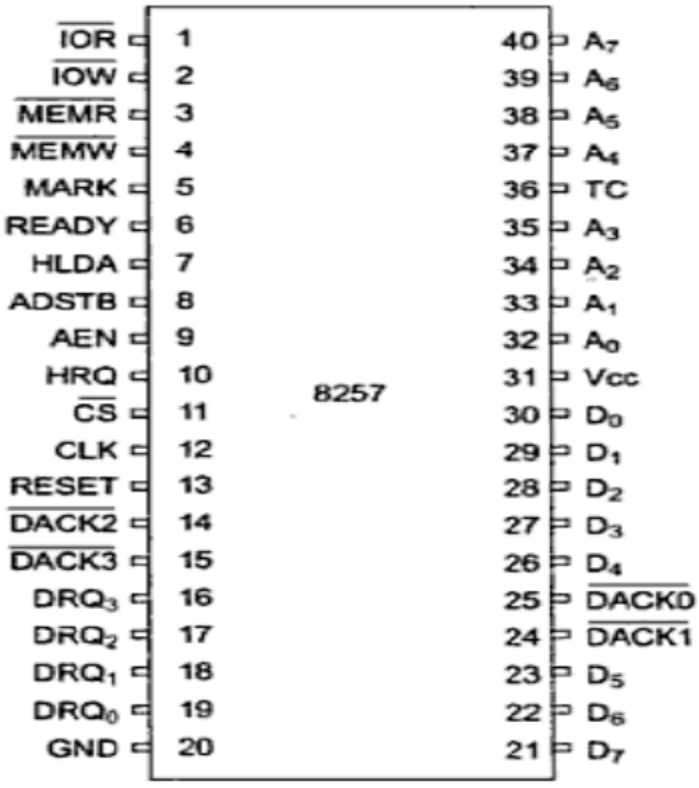


These bits remain set till either the status is read by the CPU or the 8257 is reset. The update flag is not affected by the read operation. This flag can only be cleared by resetting 8257 or by resetting the auto load bit of the mode set register. If the update flag is set, the contents of the channel 3 registers are reloaded to the corresponding registers of channel 2 whenever the channel 2 reaches a terminal count condition, after transferring one block and the next block is to be transferred using the autoload feature of 8257. The update flag is set every time, the channel 2 registers are loaded with contents of the channel 3 registers. It is cleared by the completion of the first DMA cycle of the new block. This register can only read.

Data Bus Buffer, Read/Write Logic, Control Unit and Priority Resolver

The 8-bit. Tristate, bidirectional buffer interfaces the internal bus of 8257 with the external system bus under the control of various control signals. In the slave mode, the read/write logic accepts the I/O Read or I/O Write signals, decodes the Ao-A3 lines and either writes the contents of the data bus to the addressed internal register or reads the contents of the selected register depending upon whether IOW or IOR signal is activated. In master mode, the read/write logic generates the IOR and IOW signals to control the data flow to or from the selected peripheral. The control logic controls the sequences of operations and generates the required control signals like AEN, ADSTB, MEMR,MEMW, TC and MARK along with the address lines A4-A7, in master mode. The priority resolver resolves the priority of the four DMA channels depending upon whether normal priority or rotating priority is programmed.

Signal Description of 8257



Pin Diagram of 8257

DRQo-DRQ3 : These are the four individual channel DMA request inputs, used by the peripheral devices for requesting the DMA services. The DRQo has the highest priority while DRQ3 has the lowest one, if the fixed priority mode is selected.

DACKo-DACK3 : These are the active-low DMA acknowledge output lines which inform the requesting peripheral that the request has been honoured and the bus is relinquished by the CPU. These ines may act as strobe lines for the requesting devices.

Do-D7: These are bidirectional, data lines used to interface the system bus with the internal data bus of 8257. These lines carry command words to 8257 and status word from 8257, in slave mode, i.e. under the control of CPU. The data over these lines may be transferred in both the directions. When the 8257 is the bus master (master mode, i.e. not under CPU control), it uses Do-D7 lines to send higher byte of the generated address to the latch. This address is further latched using ADSTB signal. the address is transferred over Do-D7 during the first clock cycle of the DMA
cycle. During the rest of the period, data is available on the data bus.

IOR: This is an active-low bidirectional tristate input line that acts as an input in the slave mode. In slave mode, this input signal is used by the CPU to read internal registers of 8257.this line acts output in master mode. In master mode, this signal is used to read data from a peripheral during a memory write cycle.

IOW : This is an active low bidirection tristate line that acts as input in slave mode to load the contents of the data bus to the 8-bit mode register or upper/lower byte of a 16-bit DMA address register or terminal count register. In the master mode, it is a control output that loads the data to a peripheral during DMA memory read cycle (write to peripheral).

CLK: This is a clock frequency input required to derive basic system timings for the internal operation of 8257.

RESET : This active-high asynchronous input disables all the DMA channels by clearing the mode register and tristates all the control lines.

Ao-A3: These are the four least significant address lines. In slave mode, they act as input which selects one of the registers to be read or written. In the master mode, they are the four least significant memory address output lines generated by 8257.

CS: This is an active-low chip select line that enables the read/write operations from/to 8257, in slave mode. In the master mode, it is automatically disabled to prevent the chip from getting selected (by CPU) while performing the DMA operation.

A4-A7 : This is the higher nibble of the lower byte address generated by 8257 during the master mode of DMA operation.

READY: This is an active-high asynchronous input used to stretch memory read and write cycles of 8257 by inserting wait states. This is used while interfacing slower peripherals..

HRQ: The hold request output requests the access of the system bus. In the noncascaded 8257 systems, this is connected with HOLD pin of CPU. In the cascade mode, this pin of a slave is connected with a DRQ input line of the master 8257, while that of the master is connected with HOLD input of the CPU.

HLDA : The CPU drives this input to the DMA controller high, while granting the bus to the device. This pin is connected to the HLDA output of the CPU. This input, if high, indicates to the DMA controller that the bus has been granted to the requesting peripheral by the CPU.

MEMR: This active –low memory read output is used to read data from the addressed memory locations during DMA read cycles.

MEMW : This active-low three state output is used to write data to the addressed memory location during DMA write operation.

ADST : This output from 8257 strobes the higher byte of the memory address generated by the DMA controller into the latches.

AEN: This output is used to disable the system data bus and the control the bus driven by the CPU, this may be used to disable the system address and data bus by using the enable input of the bus drivers to inhibit the non-DMA devices from responding during DMA operations. If the 8257 is I/O mapped, this should be used to disable the other I/O
devices, when the DMA controller addresses is on the address bus.

TC: Terminal count output indicates to the currently selected peripherals that the present DMA cycle is the last for the previously programmed data block. If the TC STOP bit in the mode set register is set, the selected channel will be disabled at the end of the DMA cycle. The TC pin is activated when the 14-bit content of the terminal count register of the selected channel becomes equal to zero. The lower order 14 bits of the terminal count register are to be programmed with a 14-bit equivalent of (n-1), if n is the desired number of DMA cycles.

MARK : The modulo 128 mark output indicates to the selected peripheral that the current DMA cycle is the 128th cycle since the previous MARK output. The mark will be activated after each 128 cycles or integral multiples of it from the beginning if the data block (the first DMA cycle), if the total number of the required DMA cycles (n) is
completely divisible by 128.

Vcc :
This is a +5v supply pin required for operation of the circuit.

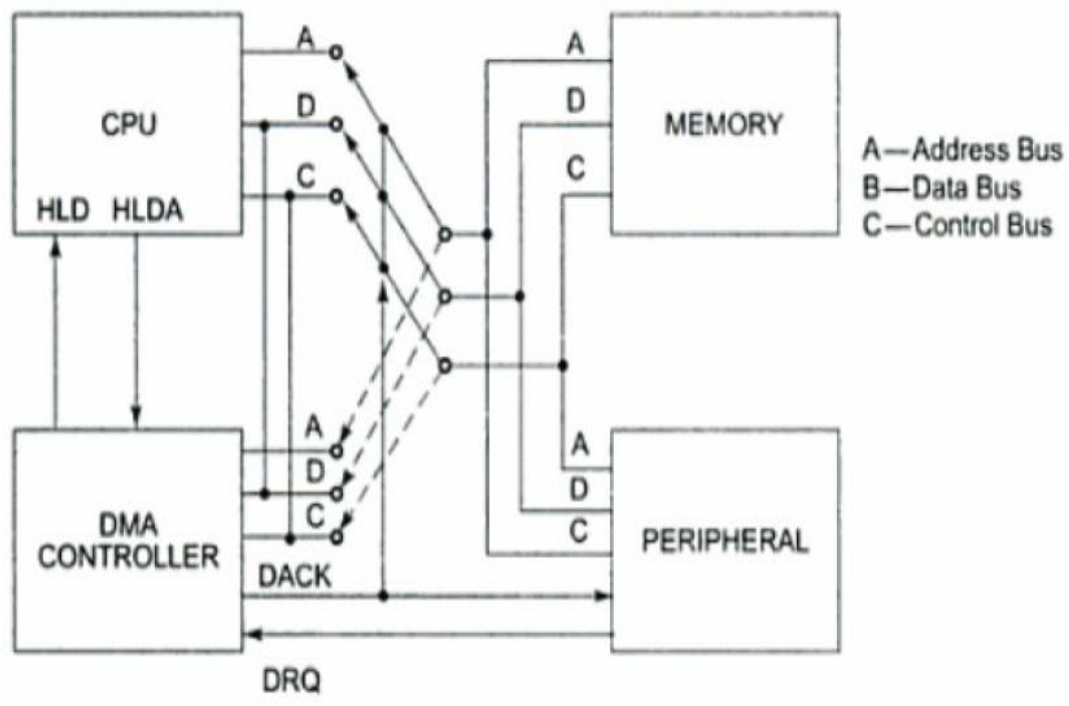
GND :

This is a return line for the supply (ground pin of the IC).

Interfacing 8257 with 8086

Once a DMA controller is initialised by a CPU property, it is ready to take control of the system bus on a DMA request, either from a peripheral or itself (in case of memory-tomemory transfer). The DMA controller sends a HOLD request to the CPU and waits for the CPU to assert the HLDA signal. The CPU relinquishes the control of the bus before asserting the HLDA signal.

A conceptual implementation of the system is shown in Figure



Once the HLDA signal goes high, the DMA controller activates the DACK signal to the requesting peripheral and gains the control of the system bus. The DMA controller is the sole master of the bus, till the DMA operation is over. The CPU remains in the HOLD status (all of its signals are tristate except HOLD and HLDA), till the DMA controller is the master of the bus.

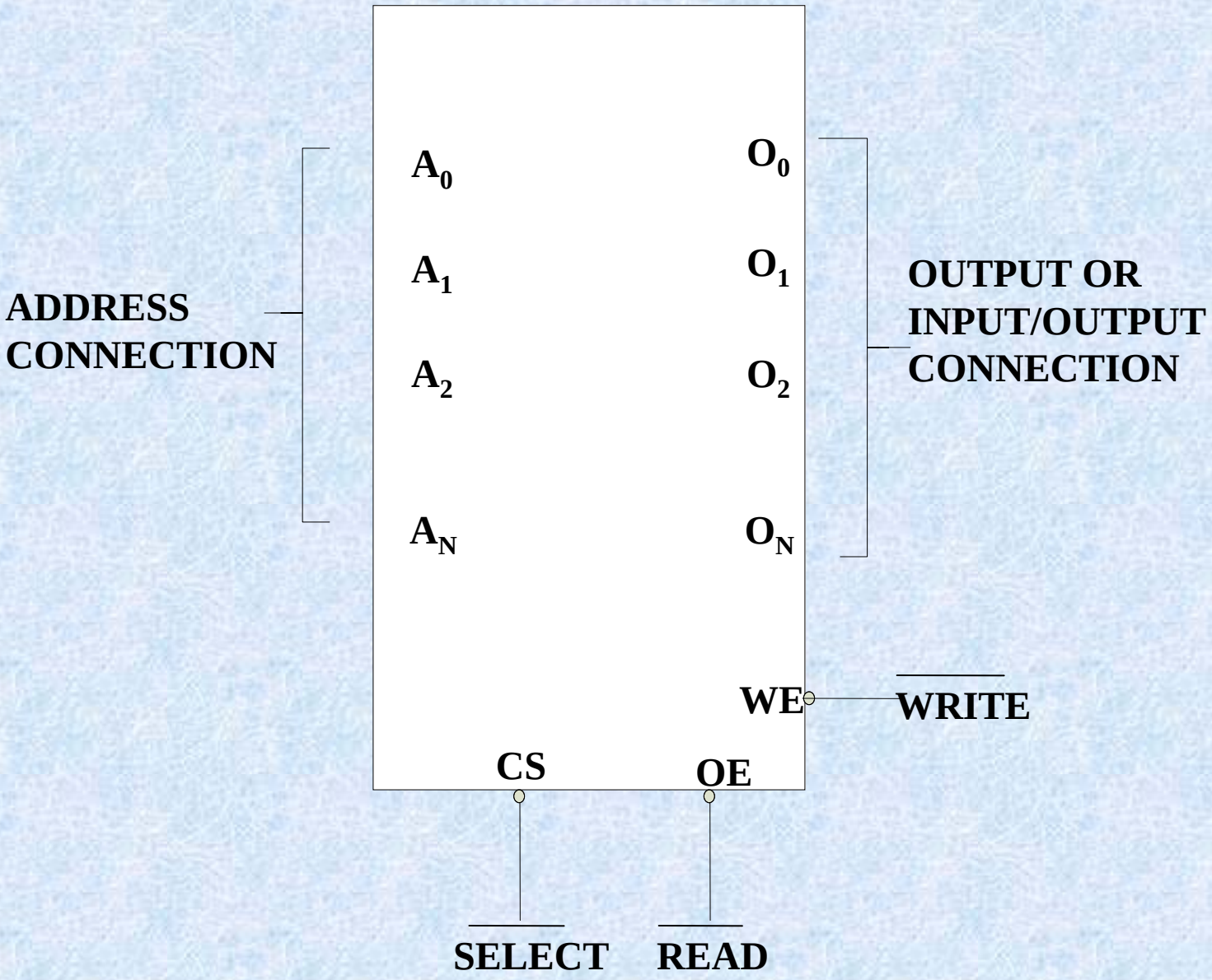
In other words, the DMA controller interfacing circuit implements a switching arrangement for the address, data and control busses of the memory and peripheral subsystem from/to the CPU to/from the DMA controller.

Interface (cont..)

- We have four common types of memory:
- Read only memory (ROM)
- Flash memory (EEPROM)
- Static Random access memory (SARAM)
- Dynamic Random access memory (DRAM).
- Pin connections common to all memory devices are: The address input, data output or input/outputs, selection input and control input used to select a read or write operation.

Interface (cont..)

- ***Address connections***: All memory devices have address inputs that select a memory location within the memory device. Address inputs are labeled from A_0 to A_n.
- ***Data connections***: All memory devices have a set of data outputs or input/outputs. Today many of them have bi-directional common I/O pins.
- ***Selection connections***: Each memory device has an input, that selects or enables the memory device. This kind of input is most often called a chip select ($\overline{CS}$), chip enable ($\overline{CE}$) or simply select ($\overline{S}$) input.



MEMORY COMPONENT ILLUSTRATING THE ADDRESS, DATA AND CONTROL CONNECTIONS

Interface (cont..)

- RAM memory generally has at least one $\overline{CS}$ or $\overline{S}$ input and ROM at least one $\overline{CE}$.
- If the $\overline{CE}$, $\overline{CS}$, $\overline{S}$ input is active the memory device perform the read or write.
- If it is inactive the memory device cannot perform read or write operation.
- If more than one $\overline{CS}$ connection is present, all most be active to perform read or write data.
- ***Control connections***: A ROM usually has only one control input, while a RAM often has one or two control inputs.

Interface (cont..)

- The control input most often found on the ROM is the output enable ($\overline{OE}$) or gate ($\overline{G}$), this allows data to flow out of the output data pins of the ROM.
- If $\overline{OE}$ and the selected input are both active, then the output is enable, if $\overline{OE}$ is inactive, the output is disabled at its high-impedance state.
- The $\overline{OE}$ connection enables and disables a set of three-state buffer located within the memory device and must be active to read data.

Interface (cont..)

- A RAM memory device has either one or two control inputs. If there is one control input it is often called R/$\overline{W}$.
- This pin selects a read operation or a write operation only if the device is selected by the selection input ($\overline{CS}$).
- If the RAM has two control inputs, they are usually labeled $\overline{WE}$ or $\overline{W}$ and $\overline{OE}$ or $\overline{G}$.
- ($\overline{WE}$) write enable must be active to perform a memory write operation and OE must be active to perform a memory read operation.
- When these two controls $\overline{WE}$ and $\overline{OE}$ are present, they must never be active at the same time.

Interface (cont..)

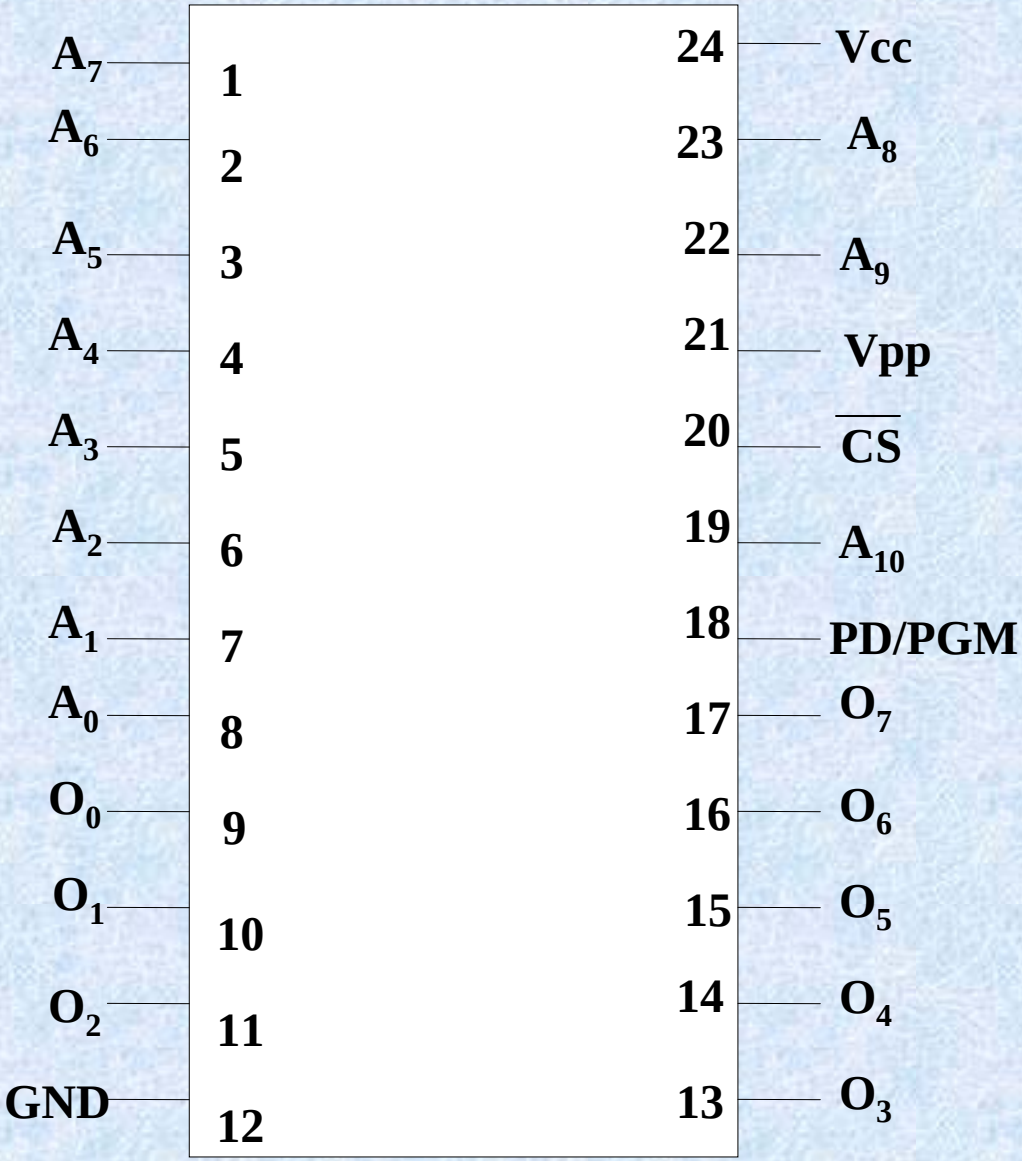
- The ROM read only memory permanently stores programs and data and data was always present, even when power is disconnected.
- It is also called as nonvolatile memory.
- EPROM (erasable programmable read only memory) is also erasable if exposed to high intensity ultraviolet light for about 20 minutes or less, depending upon the type of EPROM.
- We have PROM (programmable read only memory)
- RMM (read mostly memory) is also called the flash memory.

Interface (cont..)

- The flash memory is also called as an EEPROM (electrically erasable programmable ROM), EAROM (electrically alterable ROM), or a NOVROM (nonvolatile ROM).
- These memory devices are electrically erasable in the system, but require more time to erase than a normal RAM.
- EPROM contains the series of 27XXX contains the following part numbers : 2704(512 * 8), 2708(1K * 8), 2716(2K * 8), 2732(4K * 8), 2764(8K * 8), 27128(16K * 8) etc..

Interface (cont..)

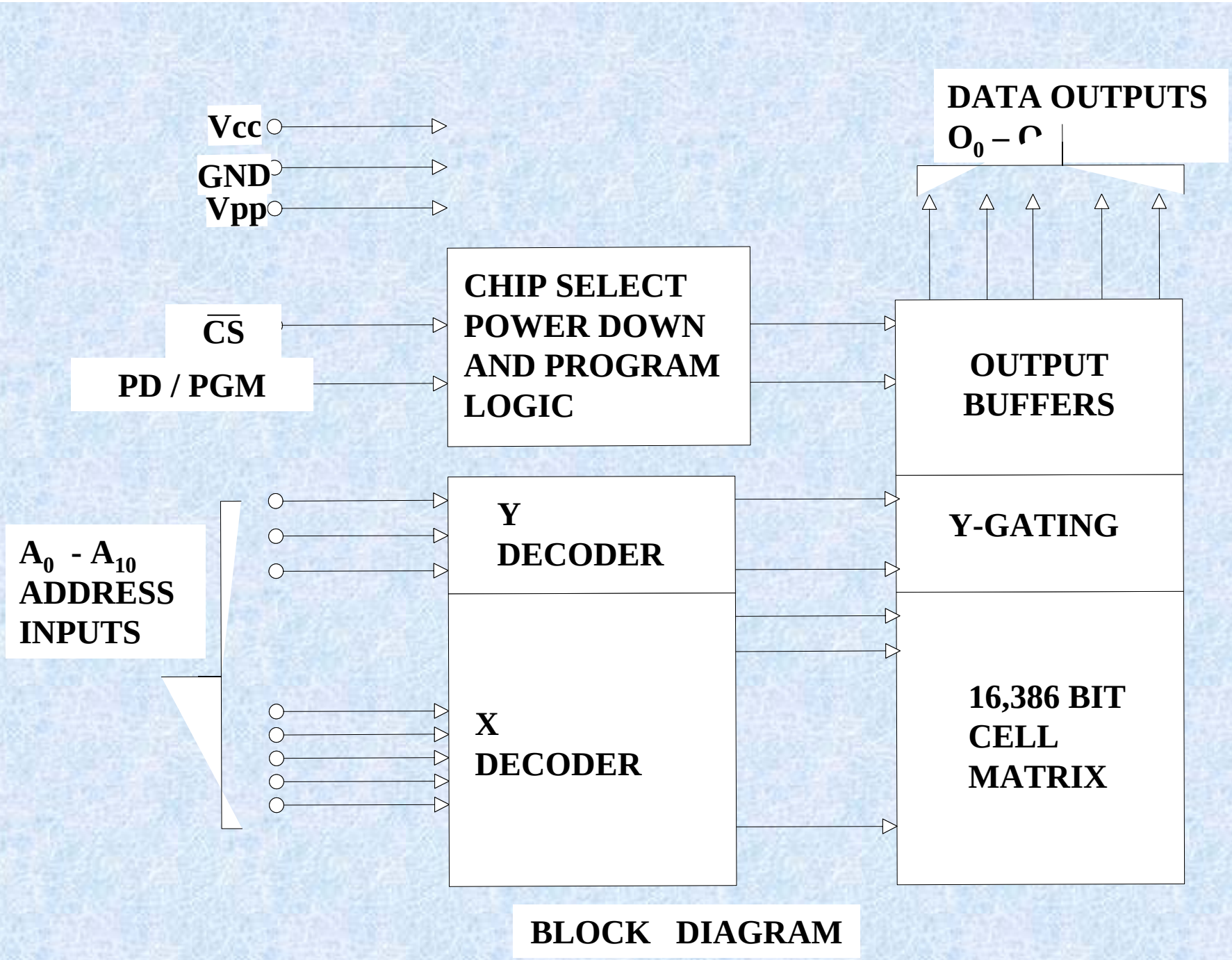
- Each of these parts contains address pins, eight data connections, one or more chip selection inputs ($\overline{CE}$) and an output enable pin ($\overline{OE}$).
- This device contains **11** address inputs and **8** data outputs.
- If both the pin connection $\overline{CE}$ and $\overline{OE}$ are at logic 0, data will appear on the output connection . If both the pins are not at logic 0, the data output connections remains at their high impedance or off state.
- To read data from the EPROM Vpp pin must be placed at a logic 1.



PIN CONFIGURATION OF 2716 EPROM

Pin Names

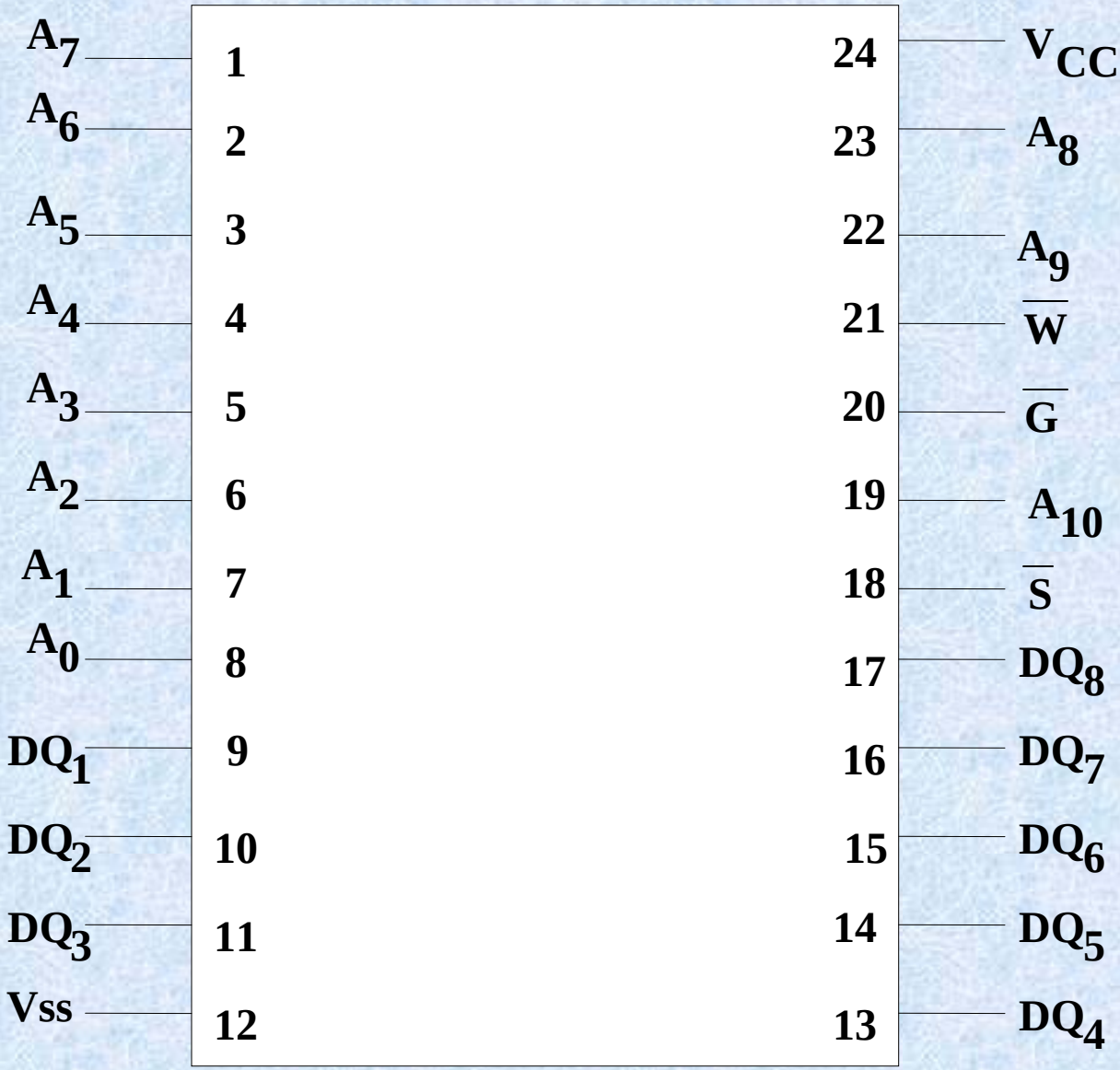
$A_0 - A_{10}$	**ADDRESSES**
PD/PGM	**POWER DOWN/PROGRAM**
$\overline{CS}$	**CHIP SELECT**
O_0-O_7	**OUT PUTS**



BLOCK DIAGRAM

Interface (cont..)

- Static RAM memory device retain data for as long as DC power is applied. Because no special action is required to retain stored data, these devices are called as static memory. They are also called volatile memory because they will not retain data without power.
- The main difference between a ROM and RAM is that a RAM is written under normal operation, while ROM is programmed outside the computer and is only normally read.
- The SRAM stores temporary data and is used when the size of read/write memory is relatively small.



PIN CONFIGURATION OF TMS 4016 SRAM

$A_0 - A_{10}$	ADDRESSES
$\overline{W}$	WRITE ENABLE
$\overline{S}$	CHIP SELECT
$DQ_0 - DQ_8$	DATA IN / DATA OUT
$\overline{G}$	OUT PUT ENABLE
Vss	GROUND
Vcc	+ 5 V SUPPLY

PIN NAMES

Interface.

- The control inputs of this RAM are slightly different from those presented earlier. The $\overline{OE}$ pin is labeled $\overline{G}$, the $\overline{CS}$ pin $\bar{S}$ and the $\overline{WE}$ pin $\overline{W}$.
- This 4016 SRAM device has 11 address inputs and 8 data input/output connections.

Static RAM Interfacing (cont..)

- The semiconductor RAM are broadly two types – static RAM and dynamic RAM.
- The semiconductor memories are organised as two dimensional arrays of memory locations.
- For example 4K * 8 or 4K byte memory contains 4096 locations, where each locations contains 8-bit data and only one of the 4096 locations can be selected at a time. Once a location is selected all the bits in it are accessible using a group of conductors called Data bus.
- For addressing the 4K bytes of memory, 12 address lines are required.

Static RAM Interfacing (cont..)

- In general to address a memory location out of N memory locations, we will require at least n bits of address, i.e. n address lines where $n = \text{Log}_2 N$.
- Thus if the microprocessor has n address lines, then it is able to address at the most N locations of memory, where $2^n=N$. If out of N locations only P memory locations are to be interfaced, then the least significant p address lines out of the available n lines can be directly connected from the microprocessor to the memory chip while the remaining (n-p) higher order address lines may be used for address decoding as inputs to the chip selection logic.

Static RAM Interfacing (cont..)

- The memory address depends upon the hardware circuit used for decoding the chip select ($\overline{CS}$). The output of the decoding circuit is connected with the $\overline{CS}$ pin of the memory chip.
- The general procedure of static memory interfacing with 8086 is briefly described as follows:

1. Arrange the available memory chip so as to obtain 16-bit data bus width. The upper 8-bit bank is called as odd address memory bank and the lower 8-bit bank is called as even address memory bank.

Static RAM Interfacing (cont..)

2. Connect available memory address lines of memory chip with those of the microprocessor and also connect the memory $\overline{RD}$ and $\overline{WR}$ inputs to the corresponding processor control signals. Connect the 16-bit data bus of the memory bank with that of the microprocessor 8086.
3. The remaining address lines of the microprocessor, $\overline{BHE}$ and A_0 are used for decoding the required chip select signals for the odd and even memory banks. The $\overline{CS}$ of memory is derived from the o/p of the decoding circuit.

Static RAM Interfacing.

- As a good and efficient interfacing practice, the address map of the system should be continuous as far as possible, i.e. there should not be no windows in the map and no fold back space should be allowed.
- A memory location should have a single address corresponding to it, i.e. absolute decoding should be preferred and minimum hardware should be used for decoding.

Dynamic RAM (cont..)

- Whenever a large capacity memory is required in a microcomputer system, the memory subsystem is generally designed using dynamic RAM because there are various advantages of dynamic RAM.
- E.g. higher packing density, lower cost and less power consumption. A typical static RAM cell may require six transistors while the dynamic RAM cell requires only a transistors along with a capacitor. Hence it is possible to obtain higher packaging density and hence low cost units are available.

Dynamic RAM (cont..)

- The basic dynamic RAM cell uses a capacitor to store the charge as a representation of data. This capacitor is manufactured as a diode that is reverse-biased so that the storage capacitance comes into the picture. This storage capacitance is utilized for storing the charge representation of data but the reverse-biased diode has leakage current that tends to discharge the capacitor giving rise to the possibility of data loss. To avoid this possible data loss, the data stored in a dynamic RAM cell must be refreshed after a fixed time interval regularly. The process of refreshing the data in RAM is called as ***Refresh cycle***.

Dynamic RAM (cont..)

- The refresh activity is similar to reading the data from each and every cell of memory, independent of the requirement of microprocessor. During this refresh period all other operations related to the memory subsystem are suspended. Hence the refresh activity causes loss of time, resulting in reduce system performance.
- However keeping in view the advantages of dynamic RAM, like low power consumption, high packaging density and low cost, most of the advanced computing system are designed using dynamic RAM, at the cost of operating speed.

Dynamic RAM (cont..)

- A dedicated hardware chip called as dynamic RAM controller is the most important part of the interfacing circuit.
- The ***Refresh cycle*** is different from the memory read cycle in the following aspects.

1. The memory address is not provided by the CPU address bus, rather it is generated by a refresh mechanism counter called as refresh counter.
2. Unlike memory read cycle, more than one memory chip may be enabled at a time so as to reduce the number of total memory refresh cycles.

Dynamic RAM (cont..)

3. The data enable control of the selected memory chip is deactivated, and data is not allowed to appear on the system data bus during refresh, as more than one memory units are refreshed simultaneously. This is to avoid the data from the different chips to appear on the bus simultaneously.
4. Memory read is either a processor initiated or an external bus master initiated and carried out by the refresh mechanism.

Dynamic RAM (cont..)

- Dynamic RAM is available in units of several kilobits to megabits of memory. This memory is arranged internally in a two dimensional matrix array so that it will have n rows and m columns. The row address n and column address m are important for the refreshing operation.
- For example, a typical 4K bit dynamic RAM chip has an internally arranged bit array of dimension 64 * 64 , i.e. 64 rows and 64 columns. The row address and column address will require 6 bits each. These 6 bits for each row address and column address will be generated by the refresh counter, during the refresh cycles.

Dynamic RAM (cont..)

- A complete row of 64 cells is refreshed at a time to minimizes the refreshing time. Thus the refresh counter needs to generate only row addresses. The row address are multiplexed, over lower order address lines.
- The refresh signals act to control the multiplexer, i.e. when refresh cycle is in process the refresh counter puts the row address over the address bus for refreshing. Otherwise, the address bus of the processor is connected to the address bus of DRAM, during normal processor initiated activities.
- A timer, called refresh timer, derives a pulse for refreshing action after each refresh interval.

Dynamic RAM (cont..)

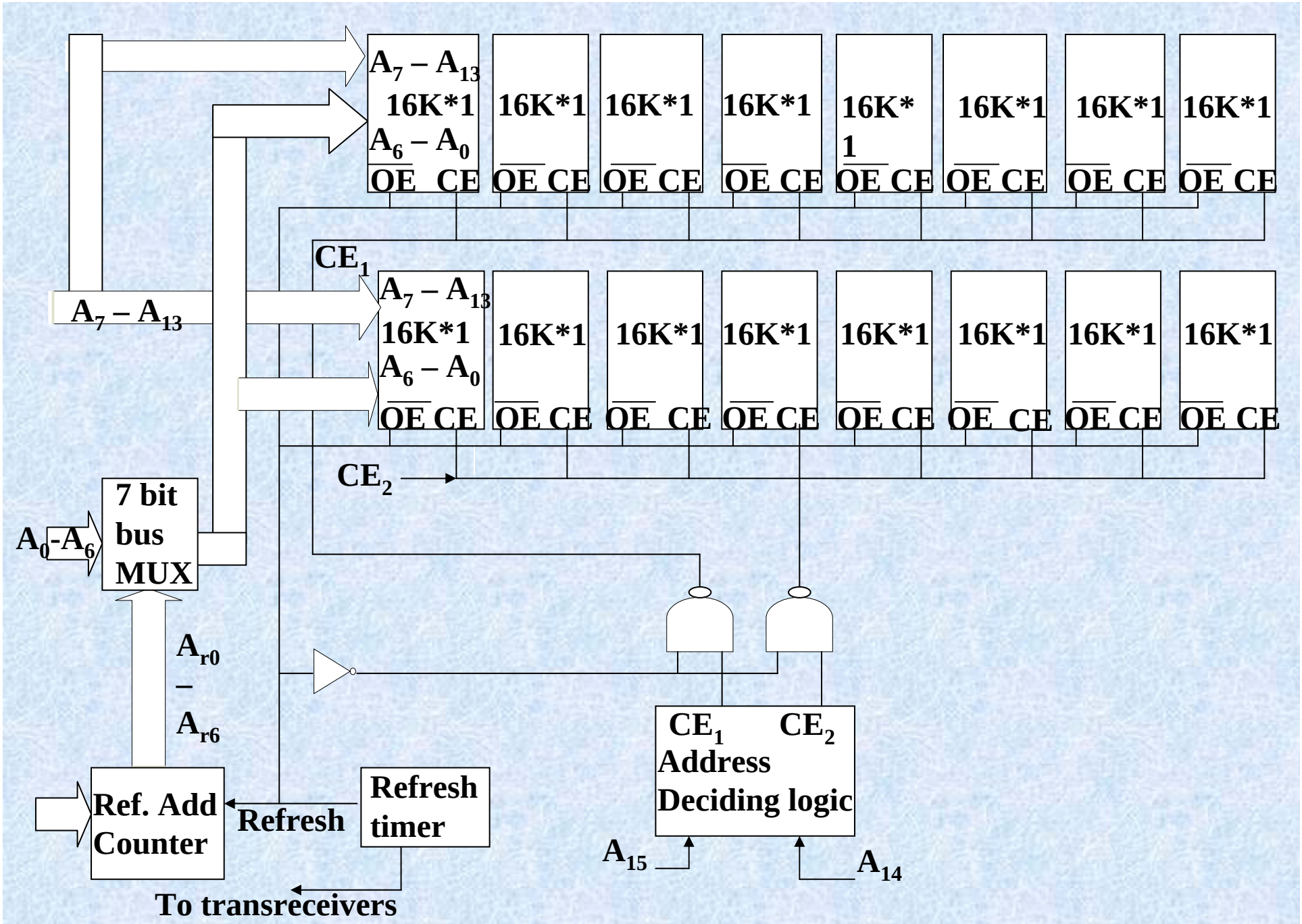
- Refresh interval can be qualitatively defined as the time for which a dynamic RAM cell can hold data charge level practically constant, i.e. no data loss takes place.
- Suppose the typical dynamic RAM chip has 64 rows, then each row should be refreshed after each refresh interval or in other words, all the 64 rows are to refreshed in a single refresh interval.
- This refresh interval depends upon the manufacturing technology of the dynamic RAM cell. It may range anywhere from 1ms to 3ms.

Dynamic RAM (cont..)

- Let us consider 2ms as a typical refresh time interval. Hence, the frequency of the refresh pulses will be calculated as follows:
- Refresh Time (per row) $t_r = (2 * 10^{-3}) / 64$.
- Refresh Frequency $f_r = 64 / (2 * 10^{-3}) = 32 * 10^3$ Hz.
- The following block diagram explains the refreshing logic and 8086 interfacing with dynamic RAM.
- Each chip is of 16K * 1-bit dynamic RAM cell array. The system contains two 16K byte dynamic RAM units. All the address and data lines are assumed to be available from an 8086 microprocessor system.

Dynamic RAM (cont..)

- The $\overline{OE}$ pin controls output data buffer of the memory chips. The CE pins are active high chip selects of memory chips. The refresh cycle starts, if the refresh output of the refresh timer goes high, $\overline{OE}$ and $\overline{CE}$ also tend to go high.
- The high CE enables the memory chip for refreshing, while high OE prevents the data from appearing on the data bus, as discussed in memory refresh cycle. The 16K * 1-bit dynamic RAM has an internal array of 128*128 cells, requiring 7 bits for row address. The lower order seven lines A_0-A_6 are multiplexed with the refresh counter output A_{10}-A_{16}.



Dynamic RAM Refreshing Logic

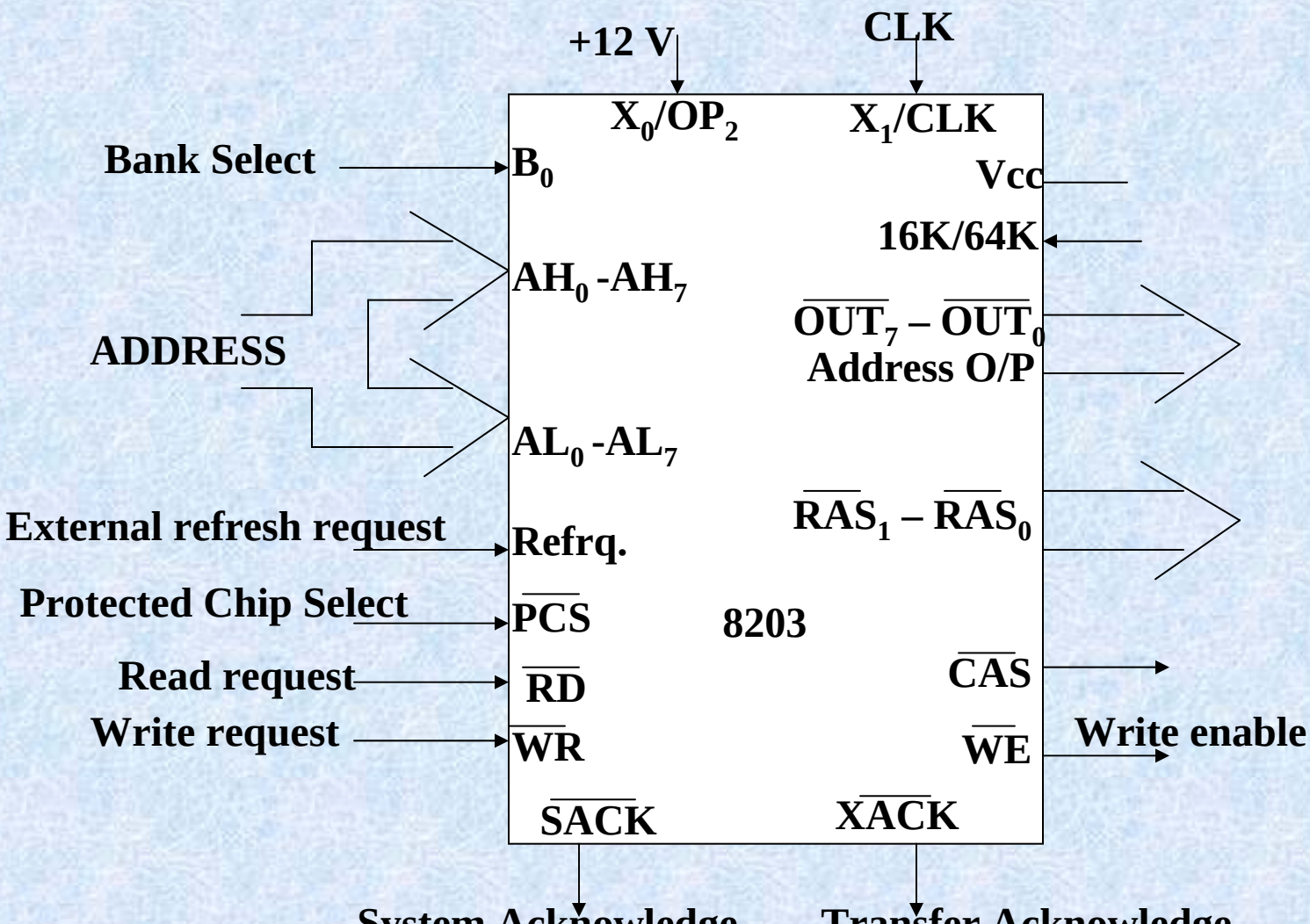


Fig : Dynamic RAM controller

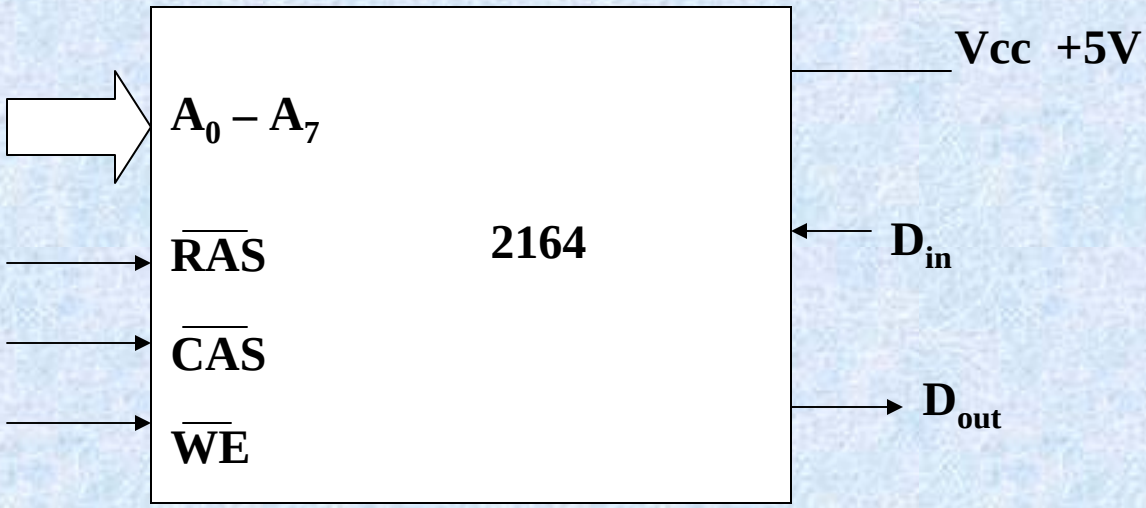


Fig : 1- bit Dynamic RAM

Dynamic RAM (cont..)

- The pin assignment for 2164 dynamic RAM is as in above fig.
- The $\overline{RAS}$ and $\overline{CAS}$ are row and column address strobes and are driven by the dynamic RAM controller outputs. A_0 –A_7 lines are the row or column address lines, driven by the OUT_0 – OUT_7 outputs of the controller. The $\overline{WE}$ pin indicates memory write cycles. The D_{IN} and D_{OUT} pins are data pins for write and read operations respectively.
- In practical circuits, the refreshing logic is integrated inside dynamic RAM controller chips like 8203, 8202, 8207 etc.

Dynamic RAM (cont..)

- Intel's 8203 is a dynamic RAM controller that support 16K or 64K dynamic RAM chip. This selection is done using pin 16K/64K. If it is high, the 8203 is configured to control 16K dynamic RAM, else it controls 64K dynamic RAM. The address inputs of 8203 controller accepts address lines A_1 to A_{16} on lines AL_0-AL_7 and AH_0-AH_7.
- The A_0 lines is used to select the even or odd bank. The $\overline{RD}$ and $\overline{WR}$ signals decode whether the cycle is a memory read or memory write cycle and are accepted as inputs to 8203 from the microprocessor.

Dynamic RAM (cont..)

- The $\overline{WE}$ signal specifies the memory write cycle and is not output from 8203 that drives the WE input of dynamic RAM memory chip. The $\overline{OUT}_0$ – $\overline{OUT}_7$ set of eight pins is an 8-bit output bus that carries multiplexed row and column addresses are derived from the address lines A_1-A_{16} accepted by the controller on its inputs AL_0-AL_7 and AH_0-AH_7.
- An external crystal may be applied between X_0 and X_1 pins, otherwise with the OP_2 pin at +12V, a clock signal may be applied at pin CLK.

Dynamic RAM (cont..)

- The $\overline{\text{PCS}}$ pin accepts the chip select signal derived by an address decoder. The REFREQ pin is used whenever the memory refresh cycle is to be initiated by an external signal.
- The $\overline{\text{XACK}}$ signal indicates that data is available during a read cycle or it has been written if it is a write cycle. It can be used as a strobe for data latches or as a ready signal to the processor.
- The $\overline{\text{SACK}}$ output signal marks the beginning of a memory access cycle.

Dynamic RAM (cont..)

- If a memory request is made during a memory refresh cycle, the $\overline{\text{SACK}}$ signal is delayed till the starring of memory read or write cycle.
- Following fig shows the 8203 can be used to control a 256K bytes memory subsystem for a maximum mode 8086 microprocessor system.
- This design assumes that data and address busses are inverted and latched, hence the inverting buffers and inverting latches are used (8283-inverting buffer and 8287- inverting latch).

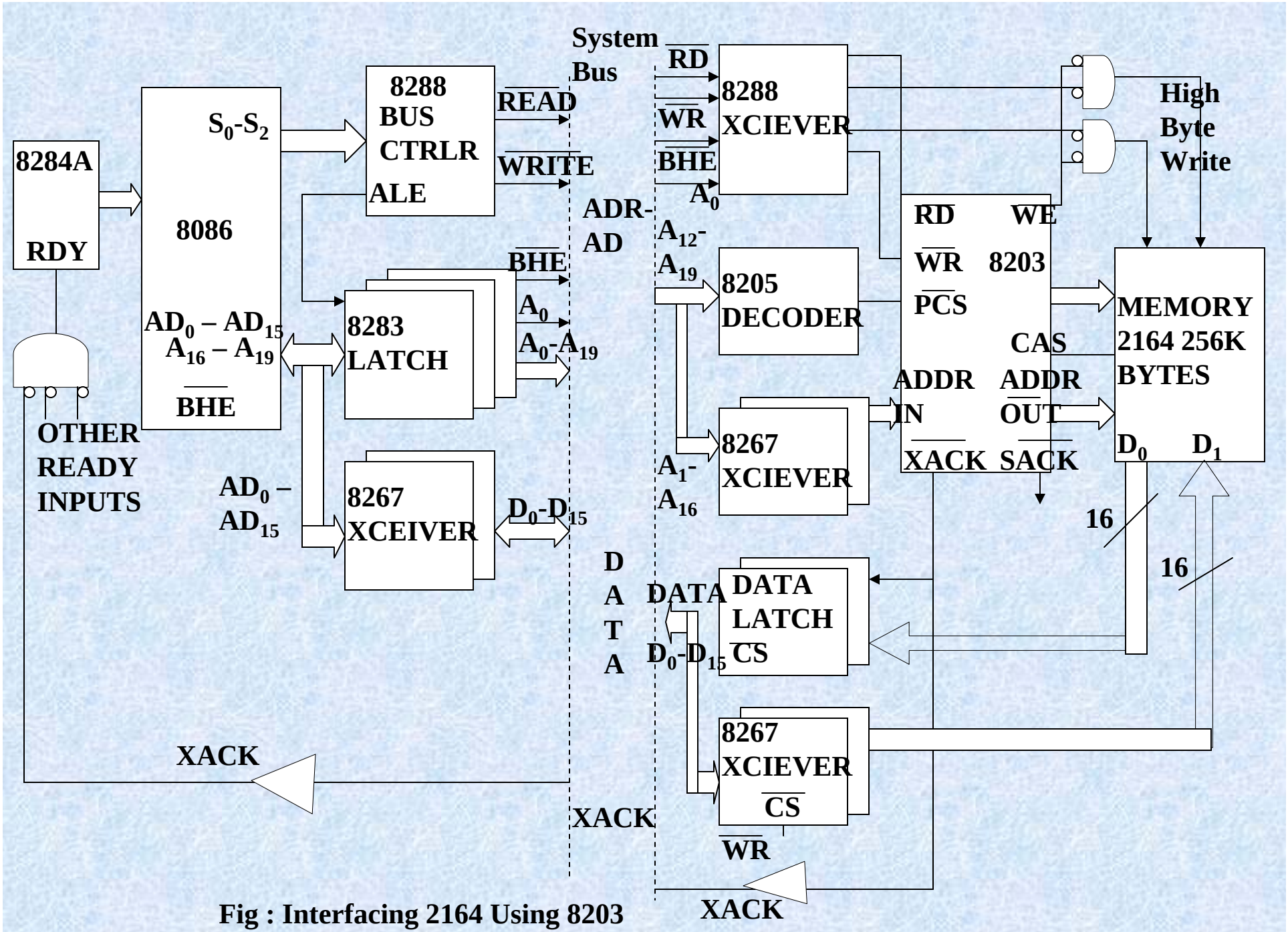


Fig : Interfacing 2164 Using 8203

Dynamic RAM.

- Most of the functions of 8208 and 8203 are similar but 8208 can be used to refresh the dynamic RAM using DMA approach. The memory system is divided into even and odd banks of 256K bytes each, as required for an 8086 system.
- The inverted $\overline{\text{AACK}}$ output of 8208 latches the A_0 and $\overline{\text{BHE}}$ signals required for selecting the banks. If the latched bank select signal and the $\overline{\text{WE}}$/PCLK output of 8208 both become low. It indicates a write operation to the respective bank.

Interrupts of 8086

- Dictionary meaning of “Interrupt”
- While the CPU is executing a program, an Interrupt breaks the normal sequence of execution of instructions, diverts its execution to some other program called Interrupt Service Routine (ISR).
- After executing ISR, the control is transferred back again to the main program which was being executed at the time of interruption
- Interrupt with in an interrupt is called nested interrupt.

Interrupts of 8086 cntd..

- Whenever a number of devices interrupt a CPU at a time and if the processor is able to handle them properly, it is said to have multiple interrupt processing capability
  Eg: 8085 – 5 interrupts can be handled simultaneously.
- In case of 8086, two types of interrupts are there: NMI – Non maskable Interrupt

  INTR – maskable by using the IF flag.

Interrupts of 8086 cntd..

- INTR is of 256 types varying from 00 to FFH (00 – 255).
- If more than one type of INTR interrupt occurs at a time, then an external chip called programmable Interrupt Controller is required to handle them.
- Interrupt → external [Hardware]

  internal [Software]

Interrupts of 8086 cntd..

- External interrupt is generated outside the processor. Eg: Key board
- Internal interrupt is generated internally by the processor circuit.
  Eg: divide by zero, overflow, INT instruction.
- Suppose an external device interrupts the CPU at the interrupt pin, either NMI or INTR of 8086, while the CPU is executing an instruction of a program, the sequence is as follows:

Interrupts of 8086 cntd..

Sequence

- Completes the execution of the current instruction
- IP is incremented to point to the next instruction.
- CPU acknowledges using INTA immediately if interrupt is NMI or TRAP etc.
- Otherwise (INT req.), checks for IF flag.
  IF =1 → interrupt is serviced
  =0 → ignored

- Note that responses to NMI, TRAP etc are independent of IF flag.
- Once interrupt is serviced, IF flag will be set to zero

Interrupts of 8086 cntd..

- The third source of an interrupt is some error condition produced in the 8086 by the execution of an instruction.
  Eg: divide by zero, overflow etc.
- At the end of each instruction cycle, the 8086 checks to see if any interrupt has been requested.
- If an interrupt has been requested, the 8086 responds to the interrupt by stepping through the following series of major actions.

Interrupts of 8086 cntd..

1. it decrements the stack pointer by 2 and pushes the flag register on to the stack
2. It disables the 8086 INTR interrupt input by clearing the interrupt flag (IF) in the flag register.
3. It resets the TRAP flag (TF) in the flag register.
4. It decrements the stack pointer by 2 and pushes the current code segment register contents on to the stack.
5. It decrements the stack pointer again by 2 and pushes the current instruction pointer contents on to the stack.
6. It does an indirect far jump to the start of the procedure you wrote to respond to the interrupt.

Interrupts of 8086 cntd..

- Fig 8.1 summarizes these steps in diagram form
- An IRET instruction at the end of the interrupt-service procedure returns execution to the main program.

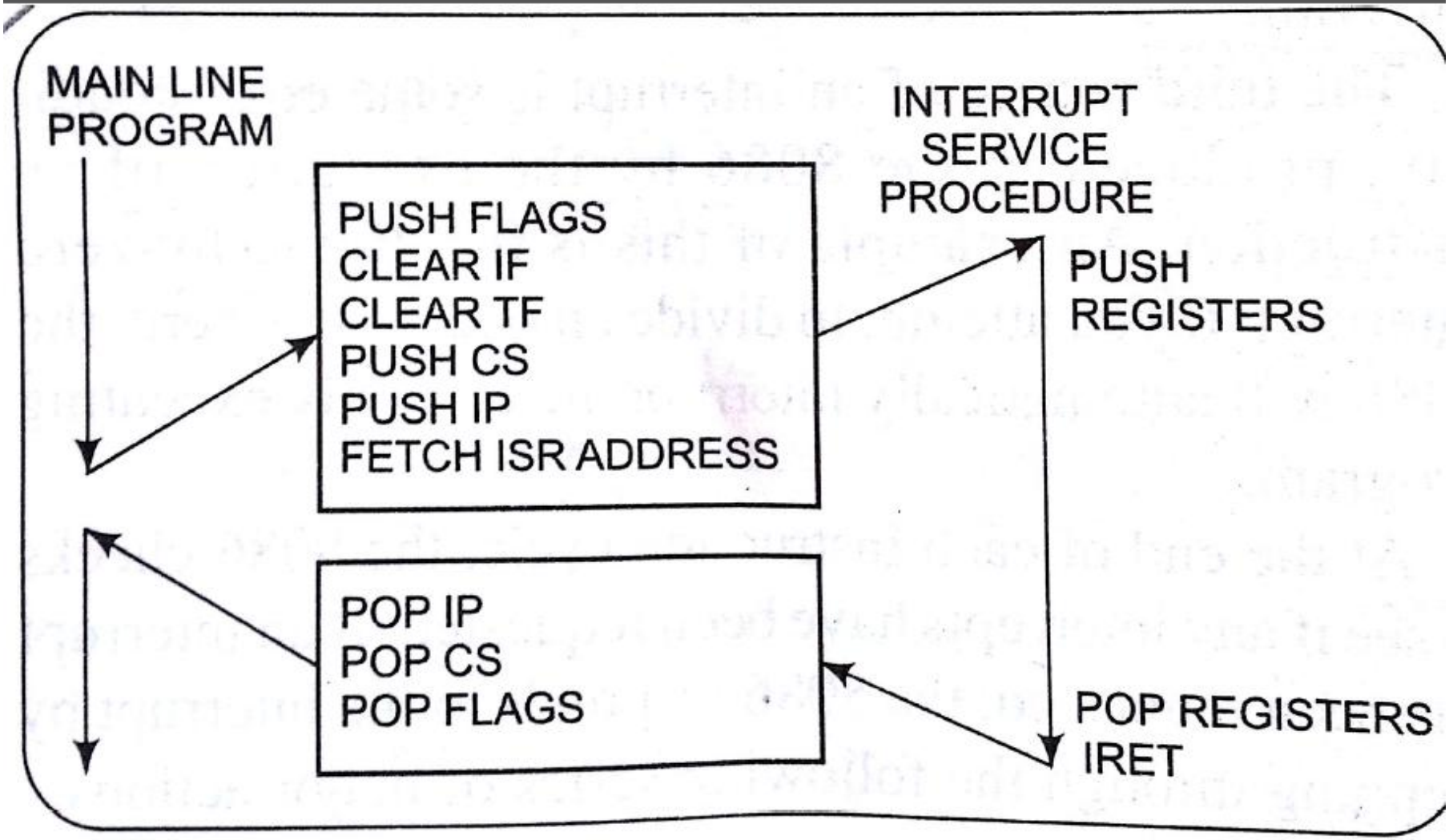


Fig. 8.1 *8086 interrupt response.*

Interrupts of 8086 cntd..

How to get to the interrupt procedure

- The starting address of the Interrupt service procedure is to be stored in CS & IP.
- To get this starting address, it requires 4 memory locations
- In an 8086 system the first 1 Kbytes of memory from 00000H to 003FFH, is set aside as a table for storing the starting addresses of interrupt service procedures
- 1 Kbytes → 4 memory locations X 256 interrupt types

Interrupts of 8086 cntd..

- The starting address of an Interrupt Service Procedure is often called the interrupt vector or interrupt pointer, so the table is referred to as the interrupt-vector table or interrupt pointer table
- Fig 8.2 shows the interrupt-vector table in memory
- IP value → Lower word CS value → higher word
- Each double word interrupt vector is identified by a number from 0 to 255 which is called as the TYPE of the interrupt.
- When the 8086 responds to a particular type interrupt, it automatically multiplies the type by 4 to produce the desired address in vector table.

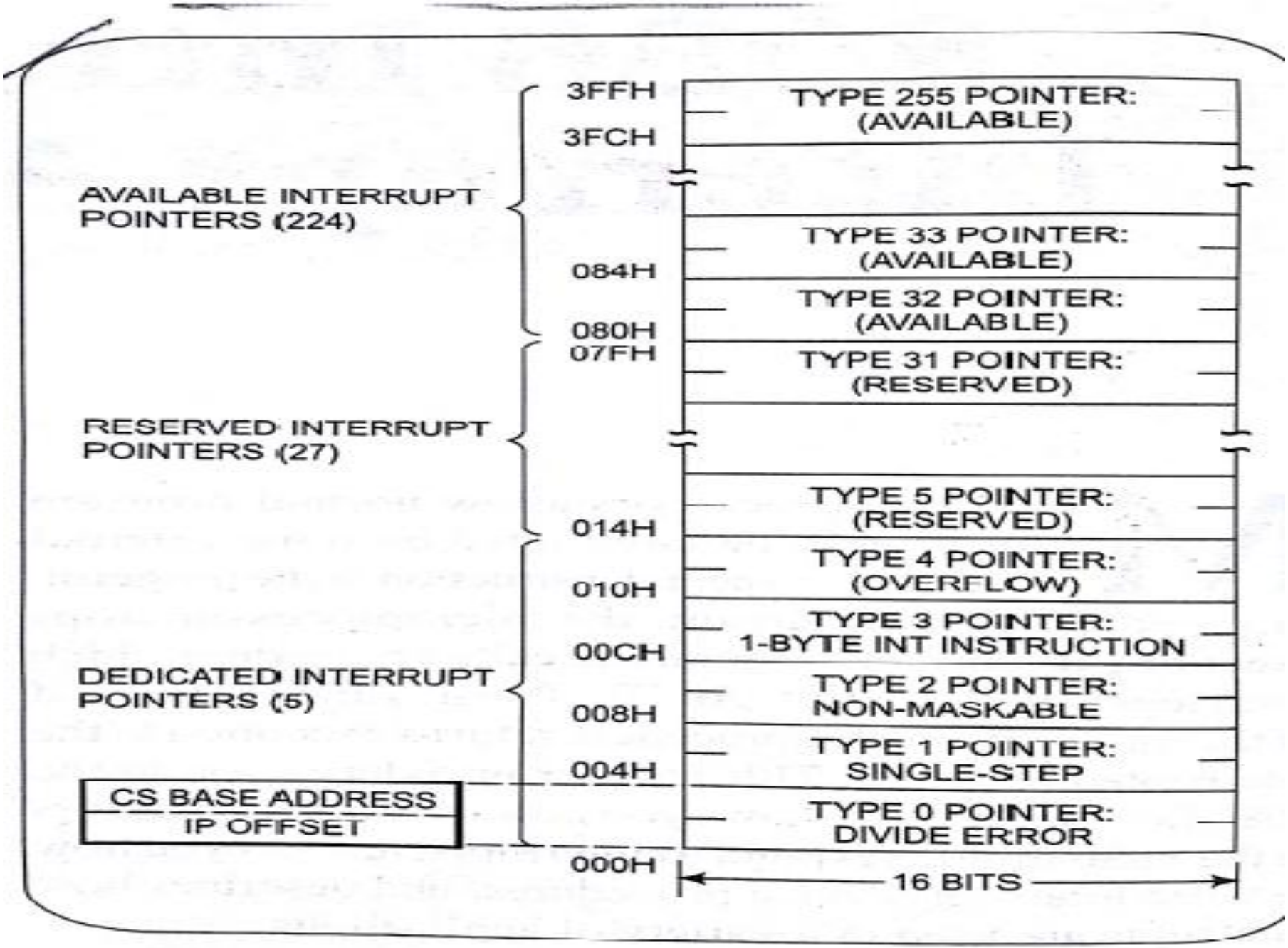


Fig. 8.2 *8086 interrupt-pointer table.*

Types of Interrupts

TYPE 0 – Divide by Zero

- It's a divide by zero error interrupt.
- DIV, IDIV
  16 bit unsigned → quotient = AL
  8 bit unsigned remainder = AH

  32 bit unsigned → quotient = AX
  16 bit unsigned remainder = DX

Types of Interrupts cntd..

- Divide by zero gives result which can not be fit in the destination register, hence it causes interrupt type 0.
- Since type 0 can not be disabled in any way, we have to take care of this in two ways.
  - Check not to divide by zero
  - Through interrupt service procedure (better method)

Types of Interrupts cntd..

TYPE 1, Single Step Interrupt

- Used for debugging
- Normal procedure
- The trap flag is reset when the 8086 does a TYPE 1 interrupt, so the single step mode will be disabled during the interrupt service procedure.

Types of Interrupts cntd..

TYPE 2 – Non Maskable

- Equivalent to NMI
- Normal procedure
- Eg: Pressure sensor on a large steam boiler connected to the NMI input.
  - System Power failure

TYPE 3 – Break Point

- It is break point interrupt

Types of Interrupts cntd..

TYPE 4 – Overflow

- It is overflow error interrupt
- Normal procedure
- Solutions are

  Jump Overflow – J0 Routine with in the main program

  Interrupt Overflow – Separate service routine
- OF = 0 → INTO = NOP

  OF = 1 → INTO = TYPE 4 Interrupt

Software Interrupts – Types 0 thru 255

- Eg: INT 38

Types of Interrupts cntd..

INTR Interrupts

- It is maskable
- CLI & STI instructions
- Disabling INTR input at start of INTR Interrupt Service Procedure avoids interrupting itself continuously.

Priority of 8086 Interrupts

Interrupt	Priority
DIVIDE ERROR, INT n, INTO	HIGHEST
NMI	
INTR	LOWEST

Programmable Interrupt Controller 8259A

- If we are working with an 8086, we have a problem here because the 8086 has only two interrupt inputs, NMI and INTR.
- If we save NMI for a power failure interrupt, this leaves only one interrupt for all the other applications. For applications where we have interrupts from multiple source, we use an external device called a ***priority interrupt controller*** (PIC) to the interrupt signals into a single interrupt input on the processor.

Architecture and Signal Descriptions of 8259A

- The architectural block diagram of 8259A is shown in fig1. The functional explication of each block is given in the following text in brief.
- **Interrupt Request Register (RR)**: The interrupts at IRQ input lines are handled by Interrupt Request internally. IRR stores all the interrupt request in it in order to serve them one by one on the priority basis.
- **In-Service Register (ISR)**: This stores all the interrupt requests those are being served, i.e. ISR keeps a track of the requests being served.

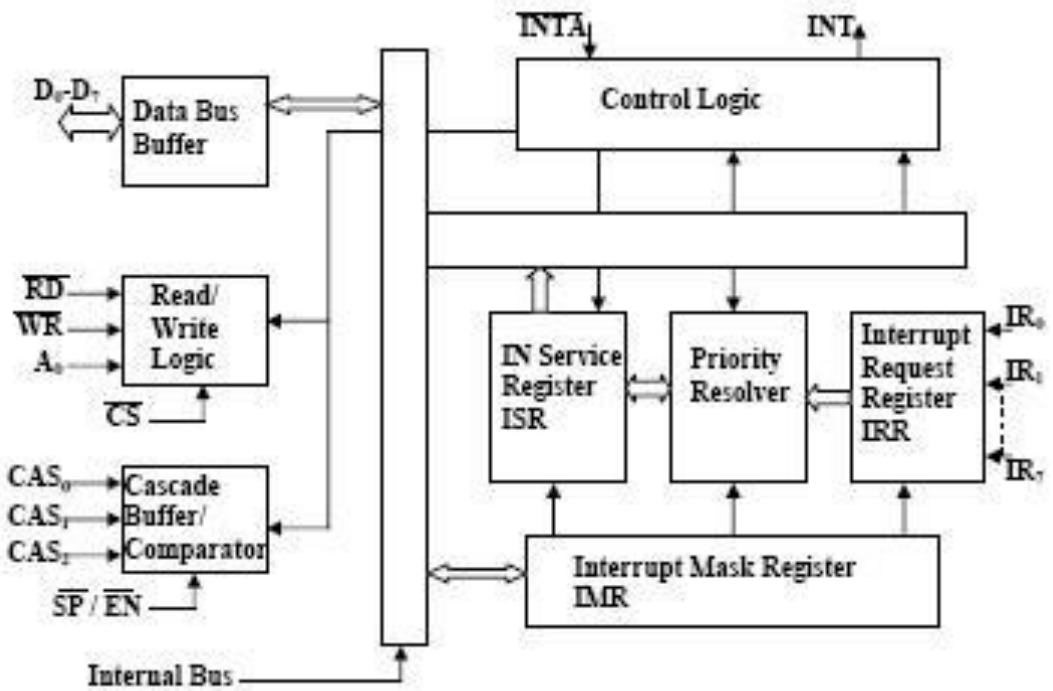


Fig:1 8259A Block Diagram

- **Priority Resolver :** This unit determines the priorities of the interrupt requests appearing simultaneously. The highest priority is selected and stored into the corresponding bit of

ISR during INTA pulse. The IR0 has the highest priority while the IR7 has the lowest one, normally in fixed priority mode. The priorities however may be altered by programming the 8259A in rotating priority mode.

- **Interrupt Mask Register (IMR)** : This register stores the bits required to mask the interrupt inputs. IMR operates on IRR at the direction of the Priority Resolver.
- **Interrupt Control Logic**: This block manages the interrupt and interrupt acknowledge signals to be sent to the CPU for serving one of the eight interrupt requests. This also accepts the interrupt acknowledge (INTA) signal from CPU that causes the 8259A to release vector address on to the data bus.
- **Data Bus Buffer** : This tristate bidirectional buffer interfaces internal 8259A bus to the microprocessor system data bus. Control words, status and vector information pass through data buffer during read or write operations.
- **Read/Write Control Logic**: This circuit accepts and decodes commands from the CPU. This block also allows the status of the 8259A to be transferred on to the data bus.
- **Cascade Buffer/Comparator**: This block stores and compares the ID's all the 8259A used in system. The three I/O pins CASO-2 are outputs when the 8259A is used as a master. The same pins act as inputs when the 8259A is in slave mode. The 8259A in master mode sends the ID of the interrupting slave device on these lines. The slave thus selected, will send its preprogrammed vector address on the data bus during the next INTA pulse.
- **CS**: This is an active-low chip select signal for enabling RD and WR operations of 8259A. INTA function is independent of CS.
- **WR** : This pin is an active-low write enable input to 8259A. This enables it to accept command words from CPU.
- **RD** : This is an active-low read enable input to 8259A. A low on this line enables 8259A to release status onto the data bus of CPU.
- **D0-D7** : These pins from a bidirectional data bus that carries 8-bit data either to control word or from status word registers. This also carries interrupt vector information.
- **CAS0 – CAS2 Cascade Lines** : A signal 8259A provides eight vectored interrupts. If more interrupts are required, the 8259A is used in cascade mode. In cascade mode, a master 8259A along with eight slaves 8259A can provide upto 64 vectored interrupt lines. These three lines act as select lines for addressing the slave 8259A.
- **PS/EN** : This pin is a dual purpose pin. When the chip is used in buffered mode, it can be used as buffered enable to control buffer transreceivers. If this is not used in buffered mode then the pin is used as input to designate whether the chip is used as a master (SP =1) or slave (EN = 0).
- **INT** : This pin goes high whenever a valid interrupt request is asserted. This is used to interrupt the CPU and is connected to the interrupt input of CPU.
- **IR0 – IR7 (Interrupt requests)** :These pins act as inputs to accept interrupt request to the CPU. In edge triggered mode, an interrupt service is requested by raising an IR pin

from a low to a high state and holding it high until it is acknowledged, and just by latching it to high level, if used in level triggered mode.

Pin Diagram

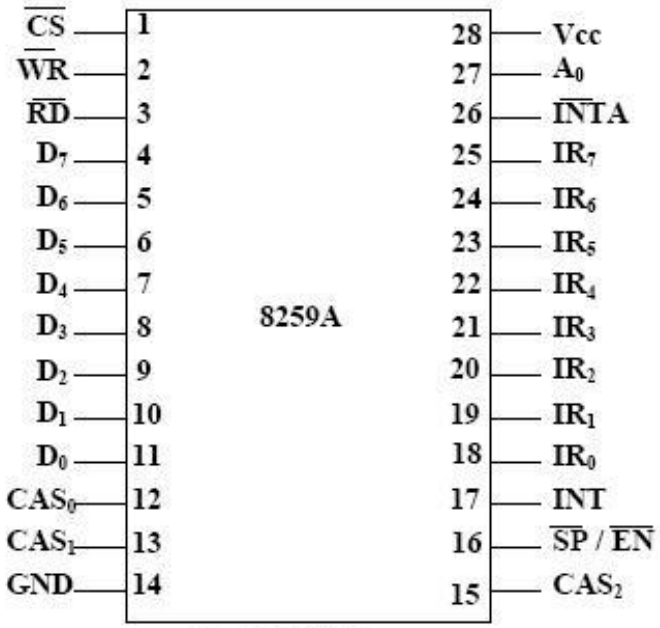


Fig : 8259 Pin Diagram

INTA (Interrupt acknowledge): This pin is an input used to strobe-in 8259A interrupt vector data on to the data bus. In conjunction with CS, WR and RD pins, this selects the different operations like, writing command words, reading status word, etc.

- The device 8259A can be interfaced with any CPU using either polling or interrupt. In polling, the CPU keeps on checking each peripheral device in sequence to ascertain if it requires any service from the CPU. If any such service request is noticed, the CPU serves the request and then goes on to the next device in sequence.
- After the entire peripheral device are scanned as above the CPU again starts from first device.
- This type of system operation results in the reduction of processing speed because most of the CPU time is consumed in polling the peripheral devices.
- In the interrupt driven method, the CPU performs the main processing task till it is interrupted by a service requesting peripheral device.
- The net processing speed of these type of systems is high because the CPU serves the peripheral only if it receives the interrupt request
- If more than one interrupt requests are received at a time, all the requesting peripherals are served one by one on priority basis.
- This method of interfacing may require additional hardware if number of peripherals to be interfaced is more than the interrupt pins available with the CPU.

SERIAL COMMUNICATION

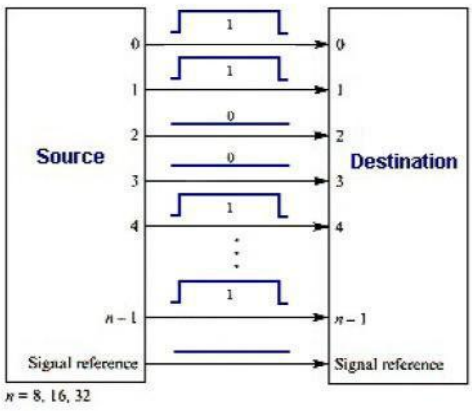
INTRODUCTION

Serial communication is common method of transmitting data between a computer and a peripheral device such as a programmable instrument or even another computer. Serial communication transmits data one bit at a time, sequentially, over a single communication line to a receiver. Serial is also a most popular communication protocol that is used by many devices for instrumentation. This method is used when data transfer rates are very low or the data must be transferred over long distances and also where the cost of cable and synchronization difficulties makes parallel communication impractical. Serial communication is popular because most computers have one or more serial ports, so no extra hardware is needed other than a cable to connect the instrument to the computer or two computers together.

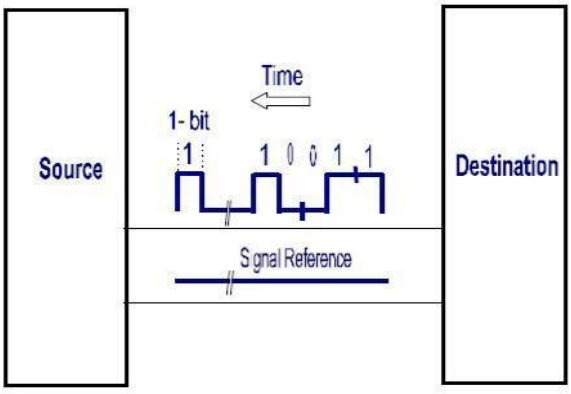
SERIAL AND PARALLEL TRANSMISSION

Let us now try to have a comparative study on parallel and serial communications to understand the differences and advantages & disadvantages of both in detail.

We know that parallel ports are typically used to connect a PC to a printer and are rarely used for other connections. A parallel port sends and receives data eight bits at a time over eight separate wires or lines. This allows data to be transferred very quickly. However, the setup looks more bulky because of the number of individual wires it must contain. But, in the case of a serial communication, as stated earlier, a serial port sends and receives data, one bit at a time over one wire. While it takes eight times as long to transfer each byte of data this way, only a few wires are required. Although this is slower than parallel communication, which allows the transmission of an entire byte at once, it is simpler and can be used over longer distances. So, at first sight it would seem that a serial link must be inferior to a parallel one, because it can transmit less data on each clock tick. However, it is often the case that, in modern technology, serial links can be clocked considerably faster than parallel links, and achieves a higher data rate.



Parallel Transmission



Serial Transmission

SERIAL DATA TRANSMISSION MODES

When data is transmitted between two pieces of equipment, three communication modes of operation can be used.

Simplex: In a simple connection, data is transmitted in one direction only. For example, from a computer to printer that cannot send status signals back to the computer.

Half-duplex: In a half-duplex connection, two-way transfer of data is possible, but only in one direction at a time.

Full duplex: In a full-duplex configuration, both ends can send and receive data simultaneously, which technique is common in our PCs.

SERIAL DATA TRANSFER SCHEMS

Like any data transfer methods, Serial Communication also requires coordination between the sender and receiver. For example, when to start the transmission and when to end it, when one particular bit or byte ends and another begins, when the receiver's capacity has been exceeded, and so on. Here comes the need for synchronization between the sender and the receiver. A protocol defines the specific methods of coordinating transmission between a sender and receiver. For example a serial data signal between two PCs must have individual bits and bytes that the receiving PC can distinguish. If it doesn't, then the receiving PC can't tell where one byte ends and the next one begin or where one bit ends and begins. So the signal must be synchronized in such a way that the receiver can distinguish the bits and bytes as the transmitter intends them to be distinguished.

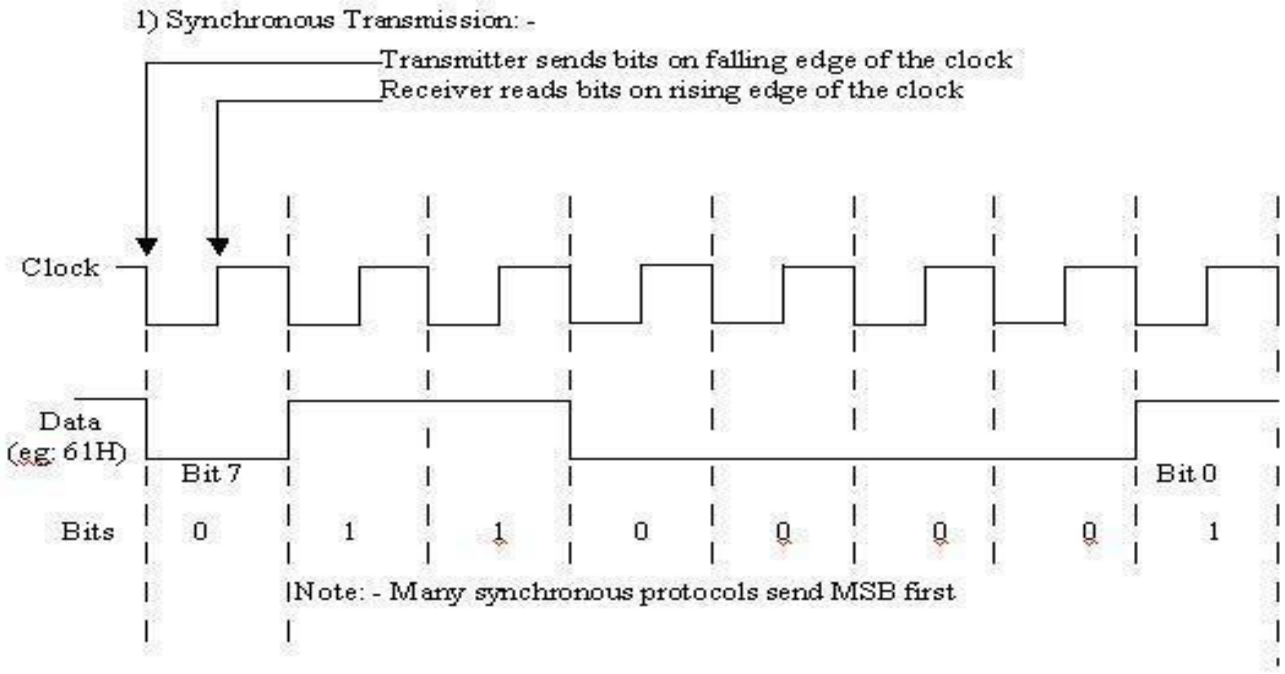
There are two ways to synchronize the two ends of the communication.

1. Synchronous data transmission
2. Asynchronous data transmission

Synchronous Data Transmission

The synchronous signaling methods use two different signals. A pulse on one signal line indicates when another bit of information is ready on the other signal line.

In synchronous transmission, the stream of data to be transferred is encoded and sent on one line, and a periodic pulse of voltage which is often called the "clock" is put on another line, that tells the receiver about the beginning and the ending of each bit.



Advantages: The only advantage of synchronous data transfer is the Lower overhead and thus, greater throughput, compared to asynchronous one.

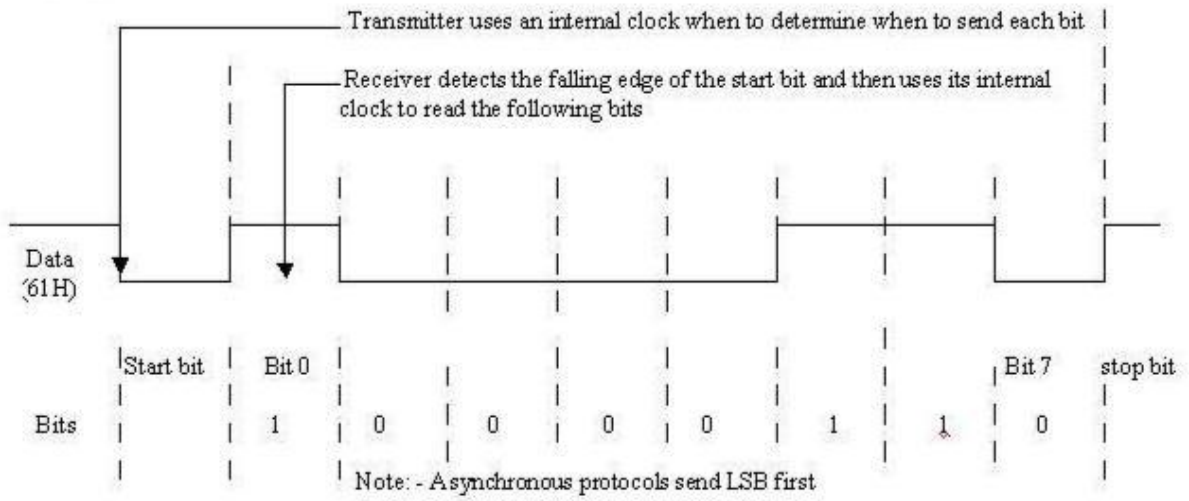
Disadvantages:

- Slightly more complex
- Hardware is more expensive

Asynchronous data transmission

The asynchronous signaling methods use only one signal. The receiver uses transitions on that signal to figure out the transmitter bit rate (known as auto baud) and timing. A pulse from the local clock indicates when another bit is ready. That means synchronous transmissions use an external clock, while asynchronous transmissions are use special signals along the transmission medium. Asynchronous communication is the commonly prevailing communication method in the personal computer industry, due to the reason that it is easier to implement and has the unique advantage that bytes can be sent whenever they are ready, a no need to wait for blocks of data to a c c u m u l a t e .

2) Asynchronous Transmission: -



Advantages:

- Simple and doesn't require much synchronization on both communication sides.
- The timing is not as critical as for synchronous transmission; therefore hardware can be made cheaper.
- Set-up is very fast, so well suited for applications where messages are generated at irregular intervals, for example data entry from the keyboard.

Disadvantages:

One of the main disadvantages of asynchronous technique is the large relative overhead, where a high proportion of the transmitted bits are uniquely for control purposes and thus carry no useful information.

8251-PROGRAMMABLE COMMUNICATION INTERFACE (USART-Universal Synchronous/Asynchronous Receiver/Transmitter)

INTRODUCTION

A USART is also called a programmable communications interface (PCI). When information is to be sent by 8086 over long distances, it is economical to send it on a single line. The 8086 has to convert parallel data to serial data and then output it. Thus lot of microprocessor time is required for such a conversion.

Similarly, if 8086 receives serial data over long distances, the 8086 has to internally convert this into parallel data before processing it. Again, lot of time is required for such a

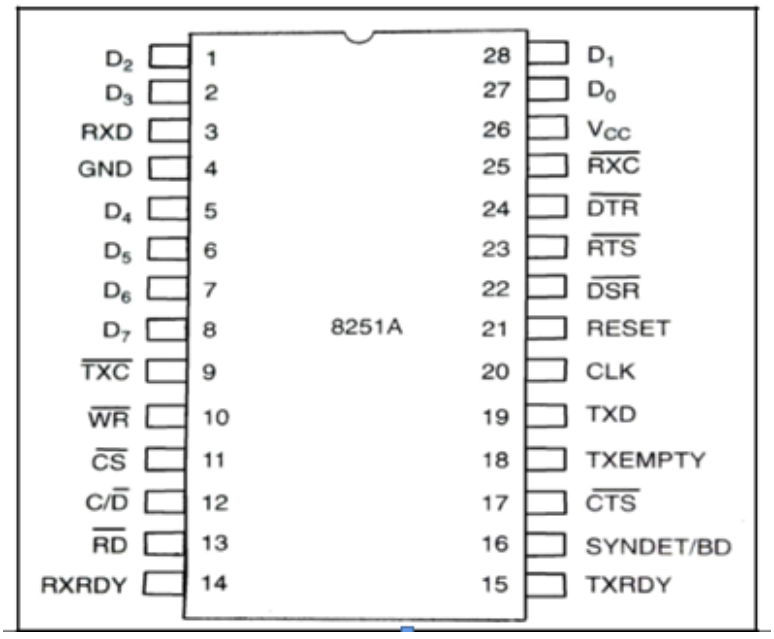
conversion. The 8086 can delegate the job of conversion from serial to parallel and vice versa to the 8251A USART used in the system.

The Intel8251A is the industry standard Universal Synchronous/Asynchronous Receiver/Transmitter (USART), designed for data communications with Intel microprocessor families such as 8080, 85, 86 and
88. The 8251A converts the parallel data received from the processor on the D7-0 data pins into serial data, and transmits it on TxD (transmit data) output pin of 8251A. Similarly, it converts the serial data received on RxD (receive data) input into parallel data, and the processor reads it using the data pins D7-0.

FEATURES

- Compatible with extended range of Intel microprocessors.
- It provides both synchronous and asynchronous data transmission.
- Synchronous 5-8 bit characters.
- Asynchronous 5-8 bit characters.
- It has full duplex, double buffered transmitter and receiver.
- Detects the errors-parity, overrun and framing errors.
- All inputs and outputs are TTL compatible.
- Available in 28-pin DIP package.

PIN DIAGRAM



ARCHITECTURE

The 8251A is a USART (Universal Synchronous Asynchronous Receiver Transmitter) for serial data communication. As a peripheral device of a microcomputer system, the 8251 receives parallel data from the CPU and transmits serial data after conversion. This device

also receives serial data from the outside and transmits parallel data to the CPU after conversion. The internal block diagram of 8251A is shown in fig below.

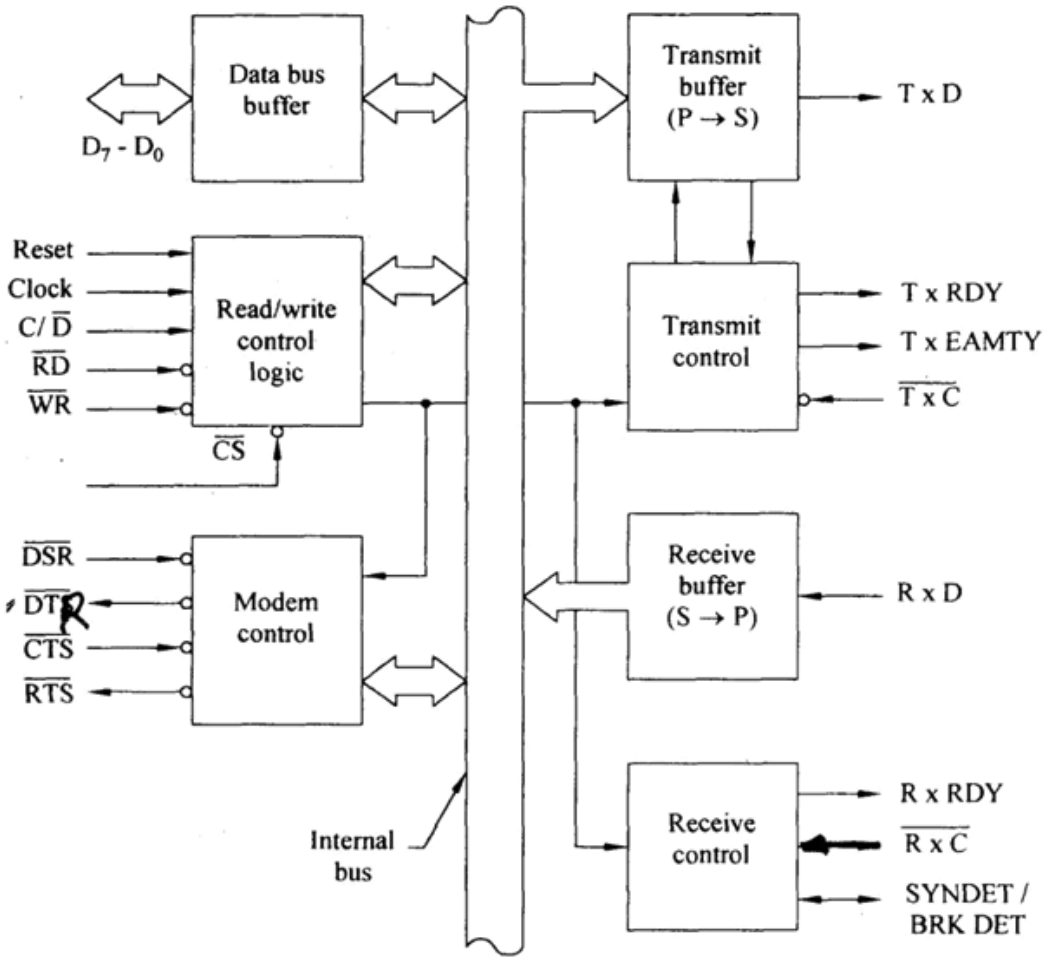


Fig. 5.7 Block diagram of 8251

Fig. 5.7 shows the block diagram of 8251 A. The block diagram shows all the elements of a programmable chip; it includes the interfacing signals, the control register and the status register. The functions of various blocks are described below:

(A) Data bus buffer: This 3-state, bidirectional buffer is used to interface the 8251A to the system data bus. Data is transmitted or received by the buffer upon execution of input and output instruction of the CPU Command words and status information are also transferred through the data bus buffer. The command, status and data in and data out are separate 8-bit registers to provide double buffering.

The functional block accepts inputs form the control bus and generates control signals for overall device operation. It contains the control word register and command word register that store the various control formats for the device functional definition.

(B) Read/Write logic and Registers:

This section includes R/W control logic, six input signals, control logic, and three buffer registers; data register, control register and status register. The input signals to control logic are as follows:

RESET: A high on this input forces the 8251A into an idle mode. The device will remain at idle until a new set of control words is written into the 8251A to program its functional definition.

A command reset operation also puts the device into the idle state.

CLK (Clock): The CLK input is used to generate internal device timing and is normally connected to the phase 2 (TTL) output of the Clock Generator. No external inputs or outputs are referenced to CLK but the frequency of CLK must be greater than 30 times the Receiver or Transmitter data bit rates.

$\overline{WR}$ (Write): A "low" on this input informs the 8251A that the CPU is writing data or control words to the 8251A.

$\overline{RD}$ (Read): A "low" on this input informs the 8251A that the CPU is reading data or status information from the 8251A.

C/$\overline{D}$	RD	WR	CS	
0	0	1	0	8251 data-data bus
0	1	0	0	Data bus-8251A data
1	0	1	0	Status-data bus
1	1	0	0	Data bus-control
×	1	1	0	Data bus-3 state
×	×	×	1	Data bus 3 state

C/$\overline{D}$ (Control/Data): This input, in conjuction with the $\overline{WR}$ and $\overline{RD}$ inputs informs the 8251A that the word on the Data bus is either a data character, control word or status information.

1 = CONTROL/STATUS; 0 = DATA

$\overline{CS}$ (Chip Select) : A "low" on this input selects the 8251A. No reading or writing will occur unless the device is selected. When $\overline{CS}$ is high, the Data Bus is in the float state and $\overline{RD}$ and $\overline{WR}$ have no effect on the chip.

(C) Modem Control:

The 8251A has a set of control inputs and outputs that can be used to simplify the interface to almost any modem. The modem control signals are general purpose in nature and can be used for functions other than modem control, if necessary.

$\overline{DSR}$ (Data Set Ready) : The $\overline{DSR}$ input signal is a general-purpose, 1-bit inverting input port. Its condition can be tested by the CPU using a Status Read operation. The $\overline{DSR}$ input is normally used to test modem condition such as Data Set Ready.

$\overline{\text{DTR}}$ (Data Terminal Ready): The $\overline{\text{DTR}}$ output signal is a general-purpose, 1-bit inverting output port. It can be set "low" by programming the appropriate bit in the Command instruction word. The $\overline{\text{DTR}}$ output signal is normally used for modem control such as Data Terminal Ready.

$\overline{\text{RTS}}$ (Request to Send): The $\overline{\text{RTS}}$ output signal is a general-purpose, 1-bit inverting output port. It can be set "low" by programming the appropriate bit in the Command instruction word. The $\overline{\text{RTS}}$ output signal is normally used for modem control such as Request to send.

$\overline{\text{CTS}}$ (Clear to Send): A "low" on this input enables the 8251A to transmit serial data if the Tx Enable bit in the Command byte is set to a "one", if either a T× Enable off or $\overline{\text{CTS}}$ off condition occurs while the T× is in operation, the T× will transmit all the data in the USART, written prior to T× Disable command before shutting down.

(D) Transmitter Buffer:

The transmitter Buffer accepts parallel data from the Data Bus Buffer, converts it to a serial bit stream, inserts the appropriate characters or bits (based on the communication technique) and outputs a composite serial stream of data on the TxD output pin on the falling edge of $\overline{\text{T x C}}$. The transmitter will begin transmission upon being enabled, if $\overline{\text{CTS}}$ = 0. The $\overline{\text{T x C}}$ line will be held in the marking state immediately upon a master Reset or when Tx Enable or $\overline{\text{CTS}}$ is off or the transmitter is empty.

(E) Transmitter Control:

The Transmitter Control manages all activities associated with the transmission of serial data. It accepts and issues signals both externally and internally to accomplish this function.

T x RDY (Transmitter Ready): This output signals the CPU that the transmitter is ready to accept a data character. The T x RDY output pin can be used as an interrupt to the system, since it is masked by T x Enable; or, for Polled operation, the CPU can check T x RDY using a Status Read operation. T x RDY is automatically reset by the leading edge of $\overline{\text{WR}}$ when a data character is loaded from the CPU.

T x E (Transmitter Empty): When the 8251A has no character to send, the T x empty will go high. It resets upon receiving a character from CPU if the transmitter is enabled. Tx empty remains high when the transmitter is disabled. T x Empty can be used to indicate the end of transmission node, so that the CPU knows when to turn around in the half-duplex operation mode.

In the synchronous mode, a high on this output indicates that a character has not been loaded and the SYNC character or characters are about to be are being transmitted as filters. T x Empty does not go low when the Sync characters are being shifted out.

T x C (Transmitter Clock): The Transmitter Clock control the rate at which the character is to be transmitted. In the Synchronous transmission mode, the Baud Rate (1x) is equal to the T x C frequency. In Asynchronous transmission mode, the baud rate is a fraction of the actual T x C frequency. A portion of the mode instruction selects this factor it can be 1,1/16 or 1/64 the T x C.

For example
If Baud rate equals 220 Baud

$\overline{TXC}$ equals 220 Hz in the 1x mode.

$\overline{TXC}$ equals 3.52 KHz in the 16x mode.

$\overline{TXC}$ equals 14.08 KHz in the 64x mode.

The falling edge of $\overline{TXC}$ shifts the serial data out of the 8251A.

(F) Receiver Buffer:

The Receiver accepts serial data, converts this serial input to parallel format checks for bits or characters that are unique to the communication technique and sends an "assembled" character to the CPU. Serial data is input to R × D pin, and is clocked in on this rising edge of $\overline{R \times C}$.

(G) Receiver Control:

This functional block shown in Fig. manages all receiver - related activities which consists of the following features.

The R × D initialization circuits prevents the 8251A from mistaking and unused input line for an active low data line in the break condition. Before starting to receive serial characters on the R × D line, a valid '1' must first be detected after a chip master reset. Once this has been determined, a search for a valid low bit (start bit) is enabled. This feature is only active in the asynchronous mode, and is only done once for each master reset.

The false start bit detection circuit prevents false starts due to a transient noise spike by first detecting the falling edge and then strobing the nominal center of the start (R × D = low).

Parity error detection sets the corresponding status bit.

The framing error status bit is set if the stop bit is absent at the end of the data byte (asynchronous mode).

R × RDY (Receiver Ready) : This output indicates the the 8251A contains a character that is ready to be input to the CPU. R × RDY can be connected to the interrupt structure of the CPU or, for polled operation, the CPU can check the condition of R × RDY using a status read operation.

R × Enable, when off, holds R × RDY in the reset condition. For asynchronous mode, to set R × RDY, the receiver must be enabled to sense a start bit and a complete character must be assembled and transferred to the data output register.

Failure to read the received character from the R × Data output register prior to the assembly of the next R × data character will set overrun condition error and the previous character will be written over and lost. If the R × data is being read by the CPU when the internal transfer is occurring, overrun error will be set and the old character will be lost.

$\overline{R \times C}$ (Receiver Clock) : The receiver clock controls the rate at which the character is to be received. In synchronous mode, the baud rate (1×) is equal to the actual frequency of $\overline{R \times C}$. In asynchronous mode, the baud rate is a fraction of the actual $\overline{R \times C}$ frequency. A portion of the mode instruction selects this factor: 1, 1/16 of 1/64 the $\overline{R \times C}$.

For Example:

Baud rate equals 200 baud, if

$\overline{R \times C}$ equals 200 Hz in the 1 × mode.

$\overline{R \times C}$ equals 3200 Hz in the 16 × mode.

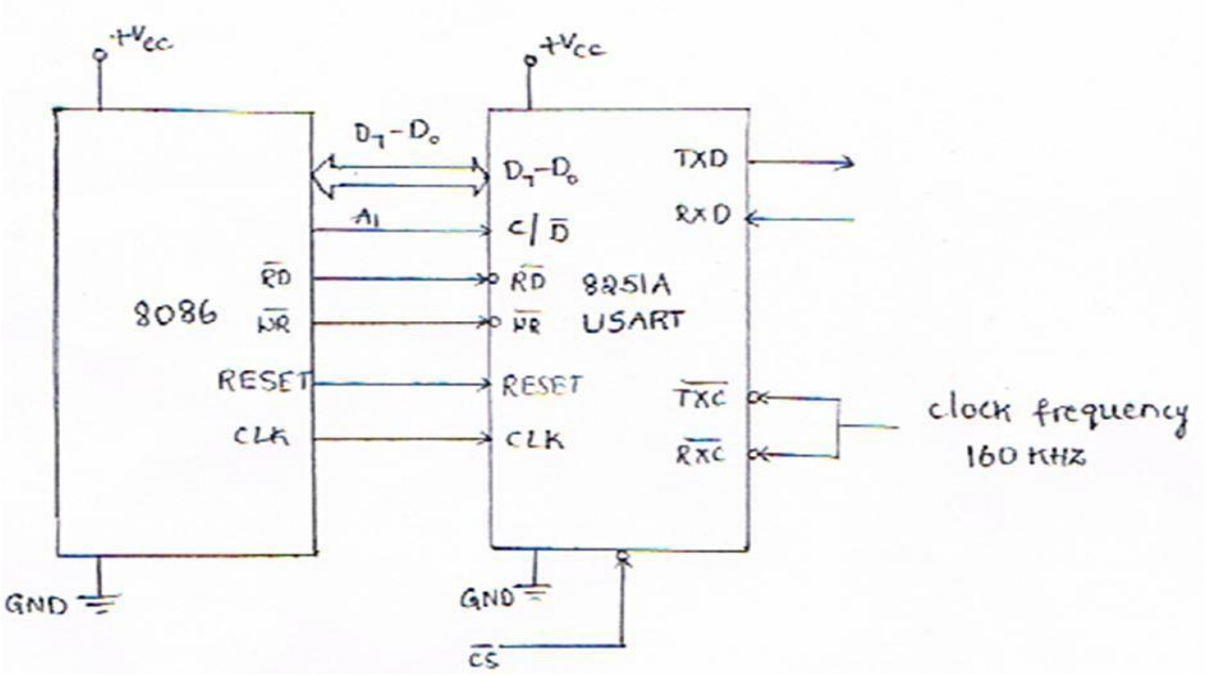
$\overline{R \times C}$ equals 128 KHz in the 64 × mode.

SYNDET (SYNC Detect/BRKDET Break Detect): This is used in Synchronous Mode for SYNDET and may be used as either input or output, programmable through the Control Word. It is reset to output mode low upon RESET. When used as an output (internal Sync mode), the SYNDET pin will go "high" to indicate that the 8251A has located the SYNC character in the Receive mode. If the 8251A is programmed to use double Sync characters (bi-sync), then SYNDET will go "high" in the middle of the last bit of the second Sync character SYNDET is automatically reset upon a status Read operation.

When used as an input (external SYNC detect mode), a positive going signal will cause the 8251A to start assembling data characters on the rising edge of the next $\overline{R \times C}$. Once in SYNC, the "high" input signal can be removed. When External SYNC Detect is programmed, internal SYNC Detect is disabled.

BREAK (Async Mode Only): This output will go high whenever the receiver remains low through two consecutive stop bit sequences (including the start bits, data bits, and parity bits); Break Detect may also be read as a Status bit. It is reset only upon a master chip Reset of R × Data returning to a "one" state.

8251A USART INTERFACING WITH 8086



PROGRAMMING THE 8251A

Prior to starting a data transmission or reception, the 8251A must be loaded with a set of control words generated by the microprocessor. These control signals define the complete functional definition of the 8251A and must immediately follow a reset operation (internal or external). The control words are split into two formats.

1. Mode instruction
2. Command instruction

Mode instruction: Mode instruction is used for setting the function of the 8251A. Mode instruction will be in "wait for write" at either internal reset or external reset. That is, the writing of a control word after resetting will be recognized as a "mode instruction."

Items set by mode instruction are as follows:

- Synchronous/asynchronous mode
- Stop bit length (asynchronous mode)
- Character length
- Parity bit
- Baud rate factor (asynchronous mode)
- Internal/external synchronization (synchronous mode)
- Number of synchronous characters (Synchronous mode)

The bit configuration of mode instruction format is shown in Figures below. In the case of synchronous mode, it is necessary to write one-or two byte sync characters. If sync characters

were written, a function will be set because the writing of sync characters constitutes part of mode instruction.

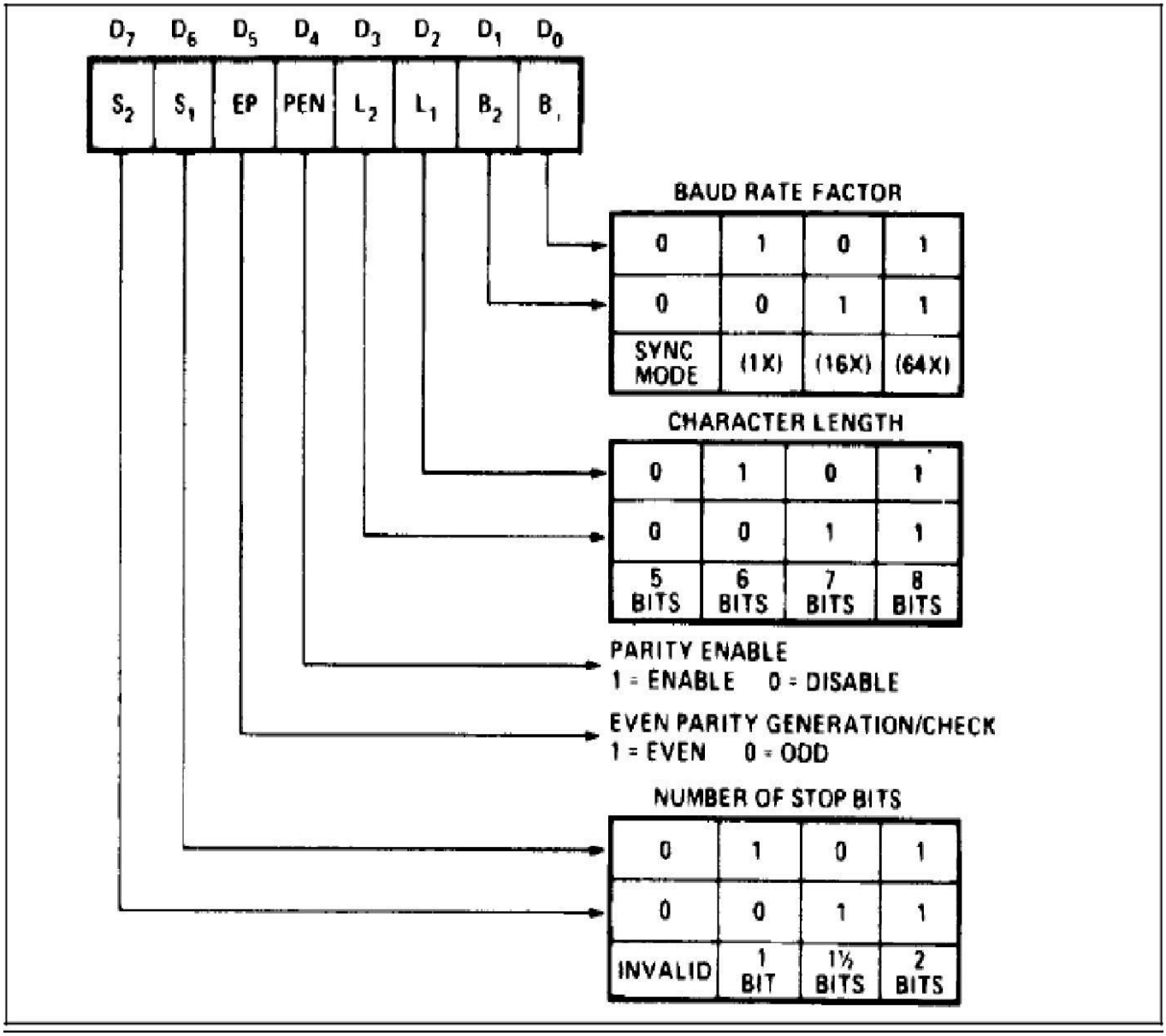
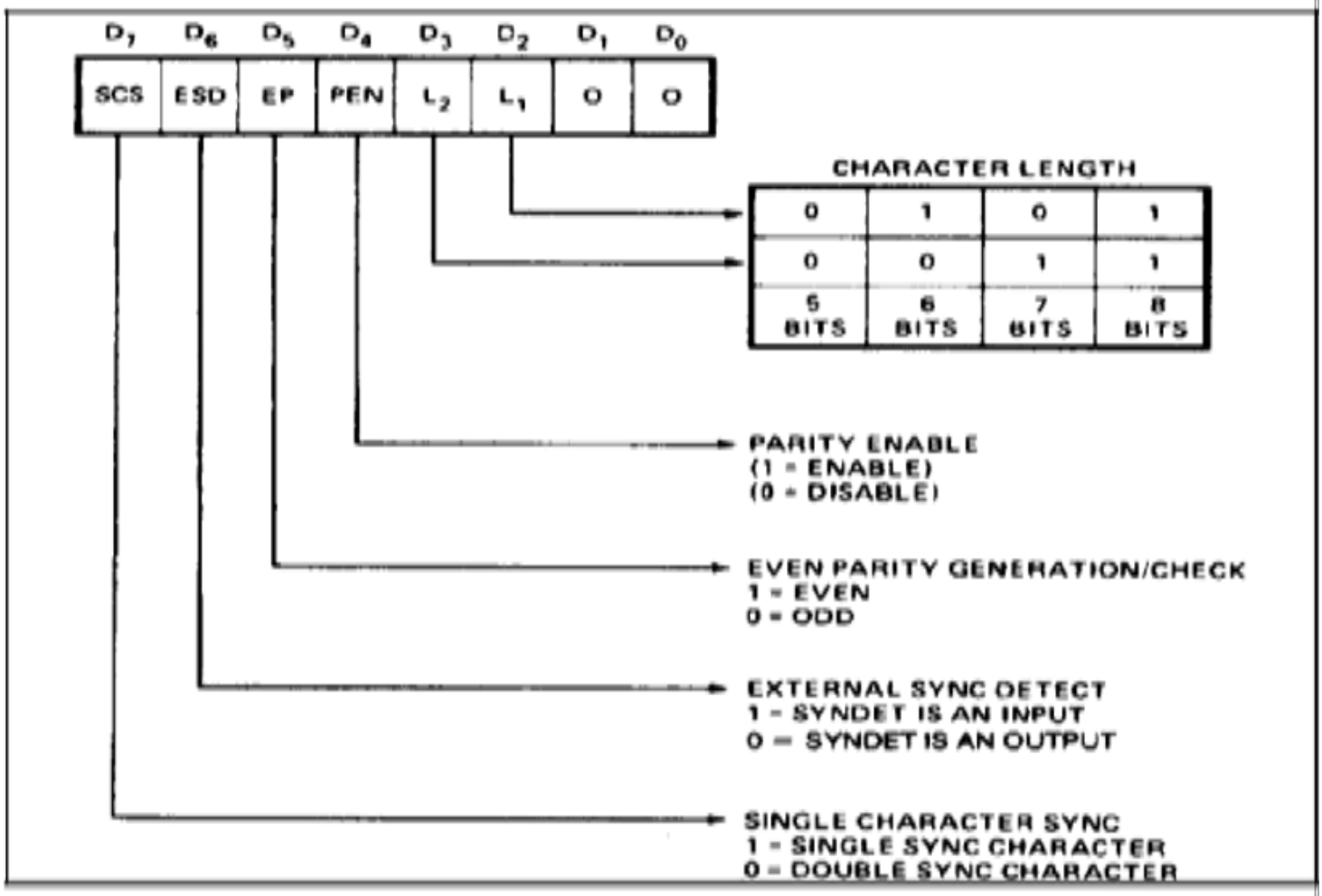


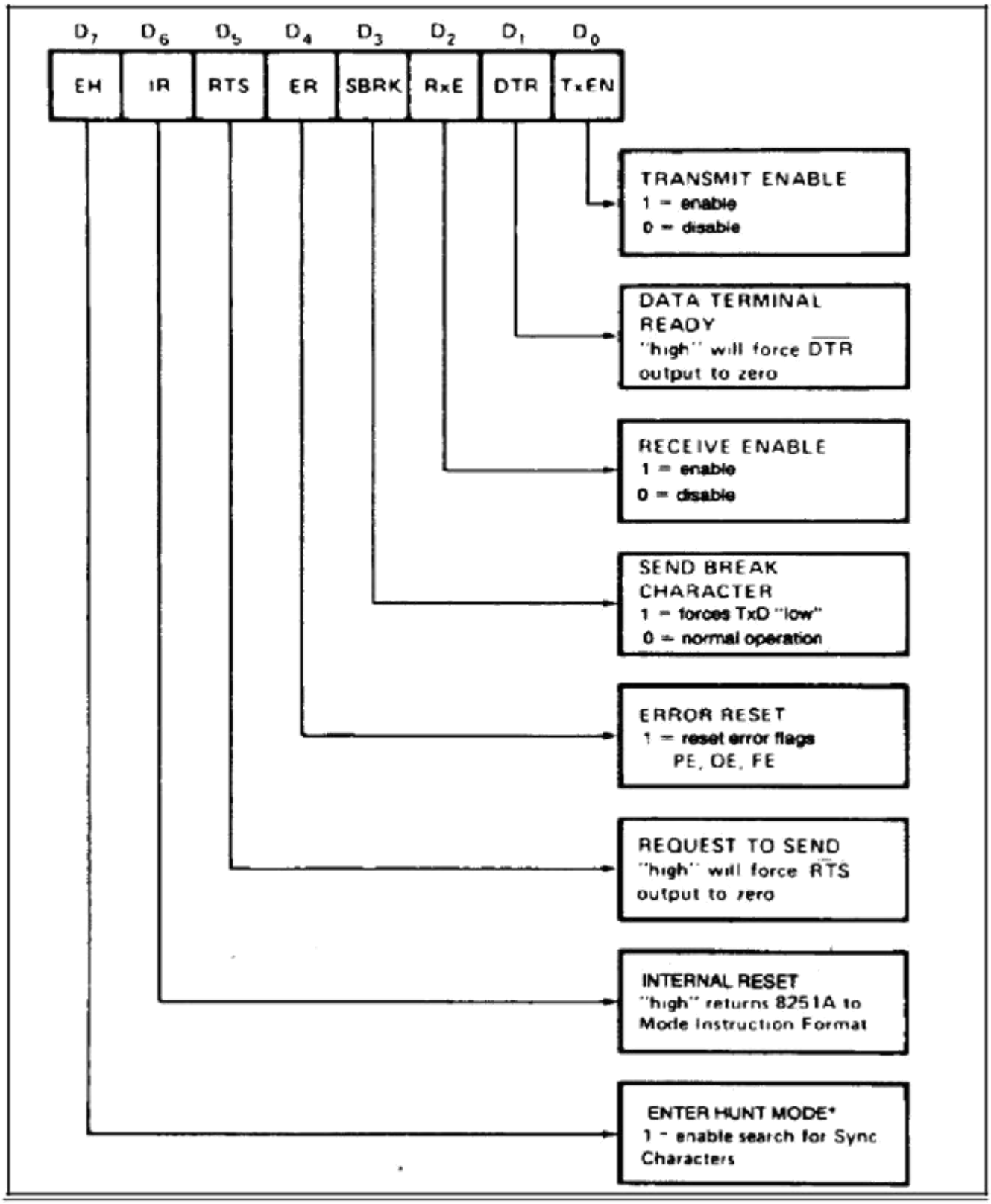
Fig. Mode instruction format, Asynchronous mode



Command Instruction: Command is used for setting the operation of the 8251. It is possible to write a command whenever necessary after writing a mode instruction and sync characters.

Items to be set by command are as follows:

- Transmit Enable/Disable
- Receive Enable/Disable
- DTR, RTS Output of data.
- Resetting of error flag.
- Sending to break characters
- Internal resetting
- Hunt mode (synchronous mode)


D7 D6 D5 D4 D3 D2 D1 D0
EH IR RTS ER SBRK RxE DTR TxEN
TRANSMIT ENABLE
1 = enable
0 = disable
DATA TERMINAL READY
"high" will force DTR output to zero
RECEIVE ENABLE
1 = enable
0 = disable
SEND BREAK CHARACTER
1 = forces TxD "low"
0 = normal operation
ERROR RESET
1 = reset error flags PE, OE, FE
REQUEST TO SEND
"high" will force RTS output to zero
INTERNAL RESET
"high" returns 8251A to Mode Instruction Format
ENTER HUNT MODE*
1 = enable search for Sync Characters

Status Word: It is possible to see the internal status of the 8251 by reading a status word. The format of status word is shown below.

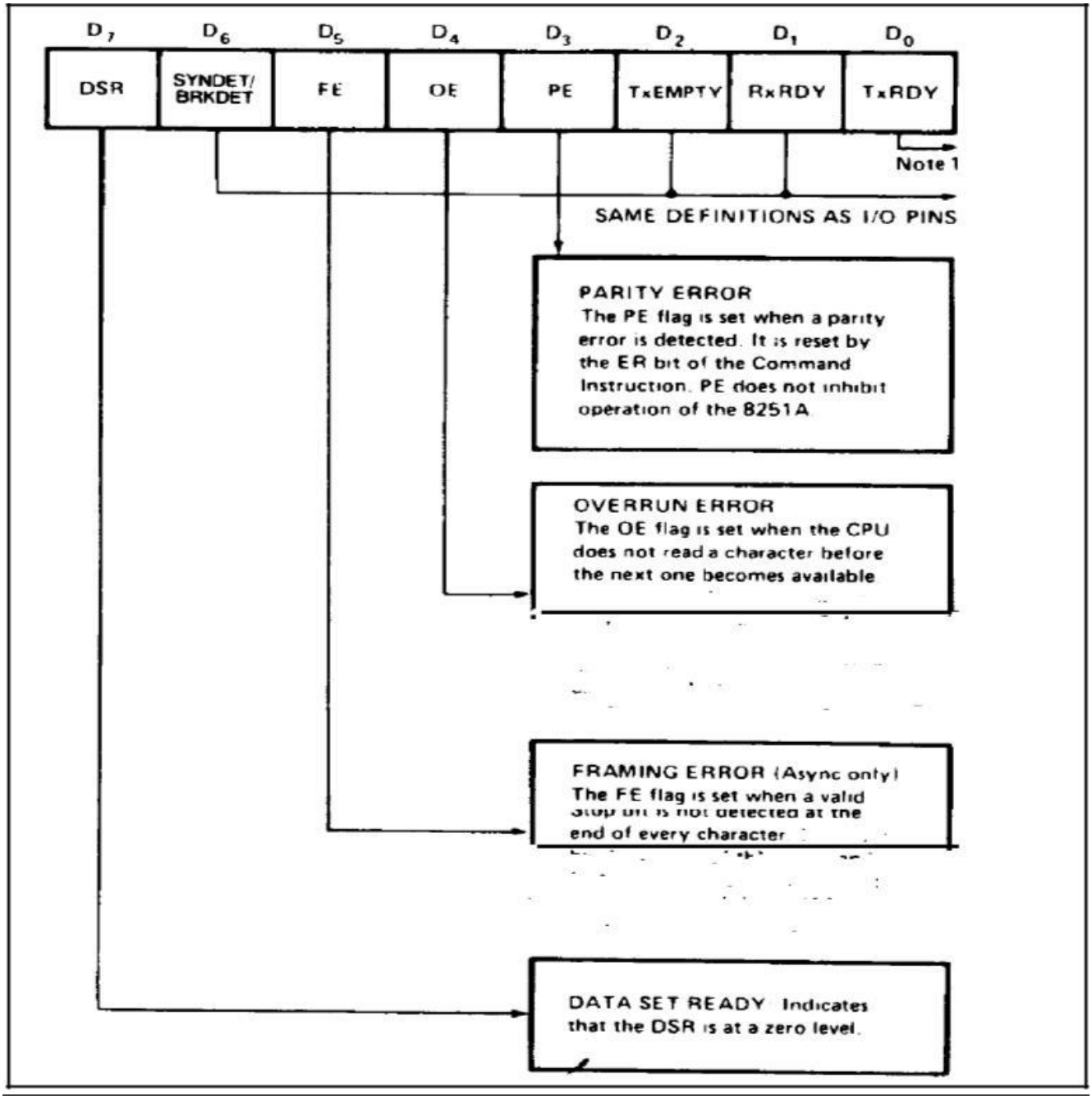


Fig. Status word

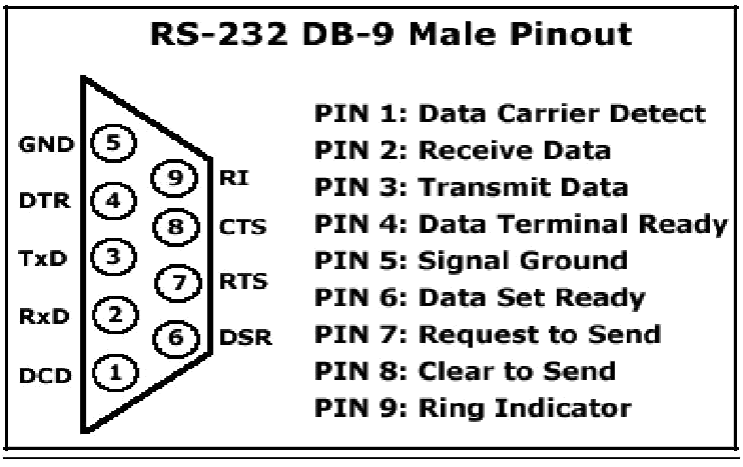
RECOMMENDED STANDARD -232C (RS-232C)

RS-232 (Recommended standard-232) is a standard interface approved by the Electronic Industries Association (EIA) for connecting serial devices. In other words, RS-232 is a long-established standard that describes the physical interface and protocol for relatively low-speed serial data communication between computers and related devices. RS-232 is the interface that your computer uses to talk to and exchange data with your modem and other serial devices. The serial ports on most computers use a subset of the RS-232C standard.

RS-232C is defined as the "Interface between data terminal equipment and data communications equipment using serial binary data exchange." This definition defines data terminal equipment (DTE) as the computer, while data communications equipment (DCE) is the modem. A modem cable has pin-to-pin connections, and is designed to connect a DTE device to a DCE device. In addition to communications between computer equipment over telephone lines, RS-232C is now widely used for direct connections between data acquisition devices and computer systems. RS-232C cables are commonly available with 4, 9 or 25-pin wiring. The 25-pin cable connects every pin; the 9-pin cables do not include many of the uncommonly used connections; 4-pin cables provide the bare minimum connections, and have jumpers to provide "handshaking" for those devices that require it.

In RS-232, user data is sent as a time-series of bits. Both synchronous and asynchronous transmissions are supported by the standard. In addition to the data circuits, the standard defines a number of control circuits used to manage the connection between the DTE and DCE. Each data or control circuit only operates in one direction, which is, signaling from a DTE to the attached DCE or the reverse. Since transmit data and receive data are separate circuits, the interface can operate in a full duplex manner, supporting concurrent data flow in both directions.

The RS-232 standard defines the voltage levels that correspond to logical one and logical zero levels for the data transmission and the control signal lines. Valid signals are either in the range of +3 to +15 volts for logic 0 or the range -3 to -15 volts for logic 1, the range between -3 to +3 volts is not a valid RS-232 level. For data transmission lines (TxD, RxD and their secondary channel equivalents) logic one is defined as a negative voltage, the signal condition is called "mark." Logic zero is positive and the signal condition is termed "space." The 9-pin RS-232C standard is shown in figure below.



End of Unit III