Microprocessors and Microcontrollers

Module 1: Architecture of Microprocessors (6)

General definitions of mini computers, microprocessors, micro controllers and digital signal processors. Overview of 8085 microprocessor. Overview of 8086 microprocessor. Signals and pins of 8086 microprocessor

Module 2: Assembly language of 8086 (6)

Description of Instructions. Assembly directives. Assembly software programs with algorithms

Module 3: Interfacing with 8086 (8)

Interfacing with RAMs, ROMs along with the explanation of timing diagrams. Interfacing with peripheral ICs like 8255, 8254, 8279, 8259, 8259 etc. Interfacing with key boards, LEDs, LCDs, ADCs, and DACs etc.

Module 4: Coprocessor 8087 (4)

Architecture of 8087, interfacing with 8086. Data types, instructions and programming

Module 5: Architecture of Micro controllers (4)

Overview of the architecture of 8051 microcontroller. Overview of the architecture of 8096 16 bit microcontroller

Module 6: Assembly language of 8051 (4)

Description of Instructions. Assembly directives. Assembly software programs with algorithms

Module 7: Interfacing with 8051 (5)

Interfacing with keyboards, LEDs, 7 segment LEDs, LCDs, Interfacing with ADCs.

Interfacing with DACs, etc.

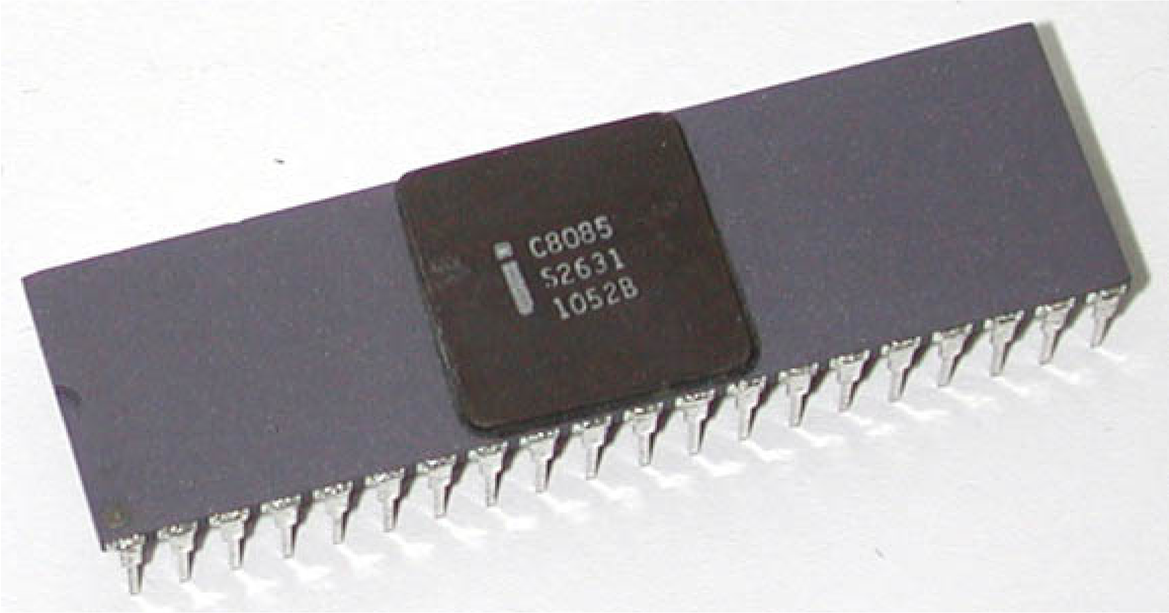
Module 8: High end processors (2)

Introduction to 80386 and 80486

Lecture Plan:

Module	Learning Units	Hours	Total
1. Architecture of Microprocessors	1. General definitions of mini computers, microprocessors, micro controllers and digital signal processors	1	6
	2. Overview of 8085 microprocessor	1	
	3. Overview of 8086 microprocessor	2.5	
	4. Signals and pins of 8086 microprocessor	1.5	
2.Assembly language of 8086	5. Description of Instructions	2.5	6
	6. Assembly directives	0.5	
	7. Algorithms with assembly software programs	3	
3. Interfacing with 8086	8. Interfacing with RAMs, ROMs along with the explanation of timing diagrams	2	8
	9. Interfacing with peripheral ICs like 8255,8254, 8279, 8259, 8259, key boards, LEDs, LCDs, ADCs, DACs etc.	6	
4. Coprocessor 8087	10. Architecture of 8087, interfacing with 8086	2	4
	11. Data types, instructions and programming	2	
5. Architecture of Micro controllers	12. Overview of the architecture of 8051 microcontroller.	2	4
	13. Overview of the architecture of 8096 16 bit microcontroller	2	
6. Assembly language of 8051	14.Description of Instructions 15.Assembly directives	2 1	5
	16. Algorithms with assembly software programs	2	
7. Interfacing with 8051	17. Interfacing with keyboards, LEDs, 7 segment LEDs, LCDs, ADCs, DACs	4	4
8. High end processors	18. Introduction to 80386 and 80486	2	2

Intel C8085



40-pin ceramic DIP
Purple ceramic/black top/tin pins

8085 Microprocessor

- The salient features of 8085 μp are :
- It is a 8 bit microprocessor.
- It is manufactured with N-MOS technology.
- It has 16 bit address bus and hence can address upto 2^{16} = 65536 bytes (64KB) memory locations through A_0-A_{15}.
- The first 8 lines of address bus and 8 lines of databus are multiplexed AD_0 – AD_7.

- Data bus is a group of 8 lines $D_0 – D_7$.
- It supports external interrupt request.
- A 16 bit program counter (PC)
- A 16 bit stack pointer (SP)
- Six 8-bit general purpose register arranged in pairs: BC, DE, HL.
- It requires a signal +5V power supply and operates at 3.2 MHZ single phase clock.
- It is enclosed with 40 pins DIP (Dual in line package).

Memory:

- Program, data and stack memories occupy the same memory space. The total addressable memory size is 64 KB.
- **Program memory** - program can be located anywhere in memory. Jump, branch and call instructions use 16-bit addresses, i.e. they can be used to jump/branch anywhere within 64 KB. All jump/branch instructions use absolute addressing.

- **Data memory** - the processor always uses 16-bit addresses so that data can be placed anywhere.
- **Stack memory** is limited only by the size of memory. Stack grows downward.
- First 64 bytes in a zero memory page should be reserved for vectors used by RST instructions.

Interrupts

- The processor has 5 interrupts. They are presented below in the order of their priority (from lowest to highest):
- **INTR** is maskable 8080A compatible interrupt. When the interrupt occurs the processor fetches from the bus one instruction, usually one of these instructions:
- One of the 8 RST instructions (RST_0 - RST_7). The processor saves current program counter into stack and branches to memory location N * 8 (where N is a 3-bit number from 0 to 7 supplied with the RST instruction).

- **CALL** instruction (3 byte instruction). The processor calls the subroutine, address of which is specified in the second and third bytes of the instruction.
- **RST5.5** is a maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 2CH (hexadecimal) address.
- **RST6.5** is a maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 34H (hexadecimal) address.

- **RST7.5** is a maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 3CH (hexadecimal) address.
- **TRAP** is a non-maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 24H (hexadecimal) address.
- All maskable interrupts can be enabled or disabled using EI and DI instructions. RST 5.5, RST6.5 and RST7.5 interrupts can be enabled or disabled individually using SIM instruction.

Reset Signals

- $\overline{\textbf{RESET IN}}$: When this signal goes low, the program counter (PC) is set to Zero, µp is reset and resets the interrupt enable and HLDA flip-flops.
- The data and address buses and the control lines are 3-stated during RESET and because of asynchronous nature of RESET, the processor internal registers and flags may be altered by RESET with unpredictable results.
- $\overline{\text{RESET IN}}$ is a Schmitt-triggered input, allowing connection to an R-C network for power-on RESET delay.
- Upon power-up, $\overline{\text{RESET IN}}$ must remain low for at least 10 ms after minimum Vcc has been reached.

- For proper reset operation after the power – up duration, RESET IN should be kept low a minimum of three clock periods.
- The CPU is held in the reset condition as long as RESET IN is applied. Typical Power-on RESET RC values $R_1 = 75K\Omega$, $C_1 = 1\mu F$.
- **RESET OUT**: This signal indicates that µp is being reset. This signal can be used to reset other devices. The signal is synchronized to the processor clock and lasts an integral number of clock periods.

Serial communication Signal

- **SID - Serial Input Data Line**: The data on this line is loaded into accumulator bit 7 when ever a RIM instruction is executed.
- **SOD – Serial Output Data Line**: The SIM instruction loads the value of bit 7 of the accumulator into SOD latch if bit 6 (SOE) of the accumulator is 1.

DMA Signals

- **HOLD**: Indicates that another master is requesting the use of the address and data buses. The CPU, upon receiving the hold request, will relinquish the use of the bus as soon as the completion of the current bus transfer.
- Internal processing can continue. The processor can regain the bus only after the HOLD is removed.
- When the HOLD is acknowledged, the Address, Data $\overline{RD}$, $\overline{WR}$ and IO/$\overline{M}$ lines are 3-stated.

- **HLDA: Hold Acknowledge** : Indicates that the CPU has received the HOLD request and that it will relinquish the bus in the next clock cycle.
- HLDA goes low after the Hold request is removed. The CPU takes the bus one half clock cycle after HLDA goes low.

- **READY :** This signal Synchronizes the fast CPU and the slow memory, peripherals.
- If READY is high during a read or write cycle, it indicates that the memory or peripheral is ready to send or receive data.
- If READY is low, the CPU will wait an integral number of clock cycle for READY to go high before completing the read or write cycle.
- READY must conform to specified setup and hold times.

Registers

- **Accumulator** or A register is an 8-bit register used for arithmetic, logic, I/O and load/store operations.
- **Flag Register** has five 1-bit flags.
- **Sign** - set if the most significant bit of the result is set.
- **Zero** - set if the result is zero.
- **Auxiliary carry** - set if there was a carry out from bit 3 to bit 4 of the result.
- **Parity** - set if the parity (the number of set bits in the result) is even.

- **Carry** - set if there was a carry during addition, or borrow during subtraction/comparison/rotation.

General Registers*:*

- 8-bit B and 8-bit C registers can be used as one 16-bit BC register pair. When used as a pair the C register contains low-order byte. Some instructions may use BC register as a data pointer.
- 8-bit D and 8-bit E registers can be used as one 16-bit DE register pair. When used as a pair the E register contains low-order byte. Some instructions may use DE register as a data pointer.

- 8-bit H and 8-bit L registers can be used as one 16-bit HL register pair. When used as a pair the L register contains low-order byte. HL register usually contains a data pointer used to reference memory addresses.
- **Stack pointer** is a 16 bit register. This register is always decremented/incremented by 2 during push and pop.
- **Program counter** is a 16-bit register.

Instruction Set

- 8085 instruction set consists of the following instructions:
- Data moving instructions.
- Arithmetic - add, subtract, increment and decrement.
- Logic - AND, OR, XOR and rotate.
- Control transfer - conditional, unconditional, call subroutine, return from subroutine and restarts.
- Input/Output instructions.
- Other - setting/clearing flag bits, enabling/disabling interrupts, stack operations, etc.

Addressing modes:

- **Register** - references the data in a register or in a register pair.

  Register indirect - instruction specifies register pair containing address, where the data is located.

  Direct, Immediate - 8 or 16-bit data.

8086 Microprocessor

- It is a 16 bit µp.
- 8086 has a 20 bit address bus can access upto 2^{20} memory locations (1 MB) .
- It can support upto 64K I/O ports.
- It provides 14, 16-bit registers.
- It has multiplexed address and data bus AD_0- AD_{15} and A_{16} – A_{19}.

Next Page

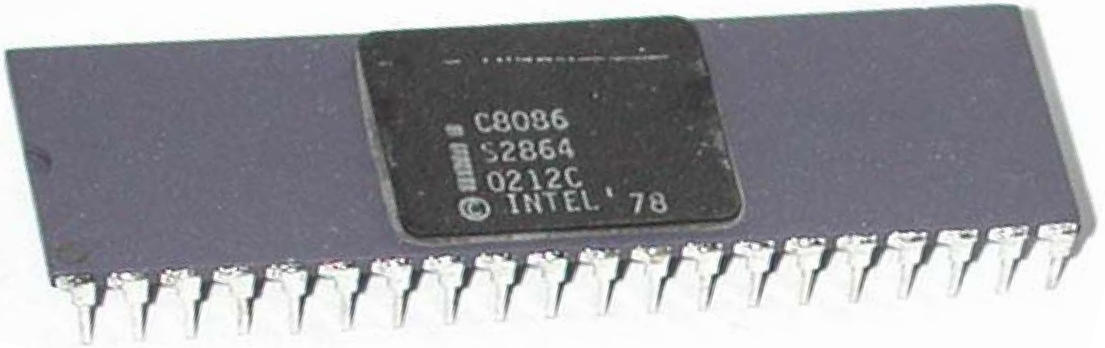
- It requires single phase clock with 33% duty cycle to provide internal timing.
- 8086 is designed to operate in two modes, Minimum and Maximum.
- It can prefetches upto 6 instruction bytes from memory and queues them in order to speed up instruction execution.
- It requires +5V power supply.
- A 40 pin dual in line package.

Next Page

Minimum and Maximum Modes:

- The minimum mode is selected by applying logic 1 to the MN / MX# input pin. This is a single microprocessor configuration.
- The maximum mode is selected by applying logic 0 to the MN / MX# input pin. This is a multi micro processors configuration.

Intel C8086



Intel C8086
5 MHz
40-pin ceramic DIP
Rare Intel C8086 processor in purple ceramic DIP package with side-brazed pins.

Internal Architecture of 8086

- 8086 has two blocks BIU and EU.
- The BIU performs all bus operations such as instruction fetching, reading and writing operands for memory and calculating the addresses of the memory operands. The instruction bytes are transferred to the instruction queue.
- EU executes instructions from the instruction system byte queue.

Next Page

- Both units operate asynchronously to give the 8086 an overlapping instruction fetch and execution mechanism which is called as Pipelining. This results in efficient use of the system bus and system performance.
- BIU contains Instruction queue, Segment registers, Instruction pointer, Address adder.
- EU contains Control circuitry, Instruction decoder, ALU, Pointer and Index register, Flag register.

Next Page

- **BUS INTERFACR UNIT:**
- It provides a full 16 bit bidirectional data bus and 20 bit address bus.
- The bus interface unit is responsible for performing all external bus operations.

Specifically it has the following functions:

- Instruction fetch, Instruction queuing, Operand fetch and storage, Address relocation and Bus control.
- The BIU uses a mechanism known as an instruction stream queue to implement a ***pipeline architecture.***

Next Page

- This queue permits prefetch of up to six bytes of instruction code. When ever the queue of the BIU is not full, it has room for at least two more bytes and at the same time the EU is not requesting it to read or write operands from memory, the BIU is free to look ahead in the program by prefetching the next sequential instruction.
- These prefetching instructions are held in its FIFO queue. With its 16 bit data bus, the BIU fetches two instruction bytes in a single memory cycle.
- After a byte is loaded at the input end of the queue, it automatically shifts up through the FIFO to the empty location nearest the output.

Next Page

- The EU accesses the queue from the output end. It reads one instruction byte after the other from the output of the queue. If the queue is full and the EU is not requesting access to operand in memory.
- These intervals of no bus activity, which may occur between bus cycles are known as ***Idle state***.
- If the BIU is already in the process of fetching an instruction when the EU request it to read or write operands from memory or I/O, the BIU first completes the instruction fetch bus cycle before initiating the operand read / write cycle.

Next Page

- The BIU also contains a dedicated adder which is used to generate the 20bit physical address that is output on the address bus. This address is formed by adding an appended 16 bit segment address and a 16 bit offset address.
- For example**:** The physical address of the next instruction to be fetched is formed by combining the current contents of the code segment CS register and the current contents of the instruction pointer IP register.
- The BIU is also responsible for generating bus control signals such as those for memory read or write and I/O read or write.

Next Page

- **EXECUTION UNIT** : The Execution unit is responsible for decoding and executing all instructions.
- The EU extracts instructions from the top of the queue in the BIU, decodes them, generates operands if necessary, passes them to the BIU and requests it to perform the read or write bys cycles to memory or I/O and perform the operation specified by the instruction on the operands.
- During the execution of the instruction, the EU tests the status and control flags and updates them based on the results of executing the instruction.

Next Page

- If the queue is empty, the EU waits for the next instruction byte to be fetched and shifted to top of the queue.
- When the EU executes a branch or jump instruction, it transfers control to a location corresponding to another set of sequential instructions.
- Whenever this happens, the BIU automatically resets the queue and then begins to fetch instructions from this new location to refill the queue.

COMMON SIGNALS

Name	Function	Type
$AD_{15} - AD_{0}$	Address/ Data Bus	Bidirectional 3 - state
$A_{19}/S_{6} - A_{16}/S_{3}$	Address / Status	Output 3 - State
$\overline{BHE}/S_{7}$	Bus High Enable / Status	Output 3- State
$MN / \overline{MX}$	Minimum / Maximum Mode Control	Input
$\overline{RD}$	Read Control	Output 3- State
TEST	Wait On Test Control	Input
READY	Wait State Controls	Input
RESET	System Reset	Input
NMI	Non - Maskable Interrupt Request	Input
INTR	Interrupt Request	Input
CLK	System Clock	Input
Vcc	+ 5 V	Input
GND	Ground	

Minimum Mode Signals (MN/ $\overline{MX}$ = Vcc)

Name	Function	Type
HOLD	Hold Request	Input
HLDA	Hold Acknowledge	Output
$\overline{WR}$	Write Control	Output, 3- state
M/$\overline{IO}$	Memory or IO Control	Output, 3-State
DT/$\overline{R}$	Data Transmit / Receiver	Output, 3- State
$\overline{DEN}$	Date Enable	Output, 3-State
ALE	Address Latch Enable	Output
$\overline{INTA}$	Interrupt Acknowledge	Output

Maximum mode signals (MN / $\overline{MX}$ = GND)

Name	Function	Type
RQ / $\overline{GT1, 0}$	Request / Grant Bus Access Control	Bidirectional
$\overline{LOCK}$	Bus Priority Lock Control	Output, 3- State
$\overline{S_2}$ – $\overline{S_0}$	Bus Cycle Status	Output, 3- State
QS1, QS0	Instruction Queue Status	Output

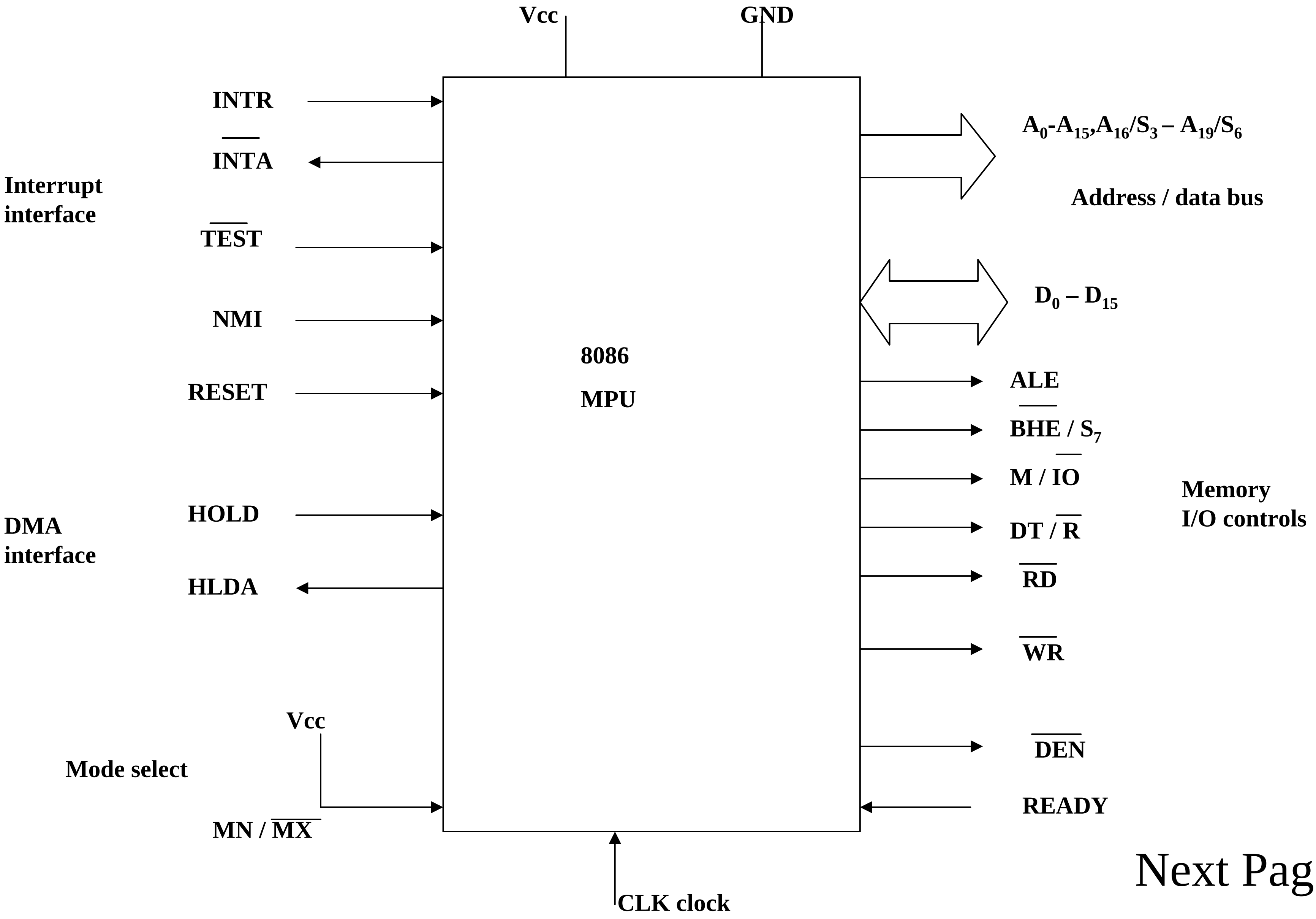
Minimum Mode Interface

- When the Minimum mode operation is selected, the 8086 provides all control signals needed to implement the memory and I/O interface.
- The minimum mode signal can be divided into the following basic groups : address/data bus, status, control, interrupt and DMA.
- **Address/Data Bus** : these lines serve two functions. As an address bus is 20 bits long and consists of signal lines A_0 through A_{19}. A_{19} represents the MSB and A_0 LSB. A 20bit address gives the 8086 a 1Mbyte memory address space. More over it has an independent I/O address space which is 64K bytes in length.

Next Page

- The 16 data bus lines D_0 through D_{15} are actually multiplexed with address lines A_0 through A_{15} respectively. By multiplexed we mean that the bus work as an address bus during first machine cycle and as a data bus during next machine cycles. D_{15} is the MSB and D_0 LSB.
- When acting as a data bus, they carry read/write data for memory, input/output data for I/O devices, and interrupt type codes from an interrupt controller.

Next Page



Next Page

Block Diagram of the Minimum Mode 8086 MPU

- **Status signal** : The four most significant address lines A_{19} through A_{16} are also multiplexed but in this case with status signals S_6 through S_3. These status bits are output on the bus at the same time that data are transferred over the other bus lines.
- Bit S_4 and S_3 together from a 2 bit binary code that identifies which of the 8086 internal segment registers are used to generate the physical address that was output on the address bus during the current bus cycle.
- Code S_4S_3 = 00 identifies a register known as ***extra segment register*** as the source of the segment address.

Next Page

S_4	S_3	Segment Register
0	0	Extra
0	1	Stack
1	0	Code / none
1	1	Data

Memory segment status codes.

Next Page

- Status line S_5 reflects the status of another internal characteristic of the 8086. It is the logic level of the internal enable flag. The last status bit S_6 is always at the logic 0 level.
- **Control Signals** : The control signals are provided to support the 8086 memory I/O interfaces. They control functions such as when the bus is to carry a valid address in which direction data are to be transferred over the bus, when valid write data are on the bus and when to put read data on the system bus.

Next Page

- ALE is a pulse to logic 1 that signals external circuitry when a valid address word is on the bus. This address must be latched in external circuitry on the 1-to-0 edge of the pulse at ALE.
- Another control signal that is produced during the bus cycle is $\overline{\mathrm{BHE}}$ bank high enable. Logic 0 on this used as a memory enable signal for the most significant byte half of the data bus D_8 through D_1. These lines also serves a second function, which is as the S_7 status line.
- Using the $\overline{\mathrm{M}}$/IO and DT/$\overline{\mathrm{R}}$ lines, the 8086 signals which type of bus cycle is in progress and in which direction data are to be transferred over the bus.

Next Page

- The logic level of $\overline{M}$/IO tells external circuitry whether a memory or I/O transfer is taking place over the bus. Logic 1 at this output signals a memory operation and logic 0 an I/O operation.
- The direction of data transfer over the bus is signaled by the logic level output at DT/$\overline{R}$. When this line is logic 1 during the data transfer part of a bus cycle, the bus is in the transmit mode. Therefore, data are either written into memory or output to an I/O device.
- On the other hand, logic 0 at DT/$\overline{R}$ signals that the bus is in the receive mode. This corresponds to reading data from memory or input of data from an input port.

Next Page

- The signal read $\overline{\text{RD}}$ and write $\overline{\text{WR}}$ indicates that a read bus cycle or a write bus cycle is in progress. The 8086 switches $\overline{\text{WR}}$ to logic 0 to signal external device that valid write or output data are on the bus.
- On the other hand, $\overline{\text{RD}}$ indicates that the 8086 is performing a read of data of the bus. During read operations, one other control signal is also supplied. This is $\overline{\text{DEN}}$ (data enable) and it signals external devices when they should put data on the bus.
- There is one other control signal that is involved with the memory and I/O interface. This is the READY signal.

Next Page

- READY signal is used to insert wait states into the bus cycle such that it is extended by a number of clock periods. This signal is provided by an external clock generator device and can be supplied by the memory or I/O sub-system to signal the 8086 when they are ready to permit the data transfer to be completed.
- **Interrupt signals** : The key interrupt interface signals are interrupt request (INTR) and interrupt acknowledge ($\overline{\text{INTA}}$).
- INTR is an input to the 8086 that can be used by an external device to signal that it need to be serviced.

Next Page

- Logic 1 at INTR represents an active interrupt request. When an interrupt request has been recognized by the 8086, it indicates this fact to external circuit with pulse to logic 0 at the $\overline{\text{INTA}}$ output.
- The $\overline{\text{TEST}}$ input is also related to the external interrupt interface. Execution of a WAIT instruction causes the 8086 to check the logic level at the $\overline{\text{TEST}}$ input.
- If the logic 1 is found, the MPU suspend operation and goes into the idle state. The 8086 no longer executes instructions, instead it repeatedly checks the logic level of the $\overline{\text{TEST}}$ input waiting for its transition back to logic 0.

Next Page

- As $\overline{\text{TEST}}$ switches to 0, execution resume with the next instruction in the program. This feature can be used to synchronize the operation of the 8086 to an event in external hardware.
- There are two more inputs in the interrupt interface: the nonmaskable interrupt NMI and the reset interrupt RESET.
- On the 0-to-1 transition of NMI control is passed to a nonmaskable interrupt service routine. The RESET input is used to provide a hardware reset for the 8086. Switching RESET to logic 0 initializes the internal register of the 8086 and initiates a reset service routine.

Next Page

- **DMA Interface signals** :The direct memory access DMA interface of the 8086 minimum mode consist of the HOLD and HLDA signals.
- When an external device wants to take control of the system bus, it signals to the 8086 by switching HOLD to the logic 1 level. At the completion of the current bus cycle, the 8086 enters the hold state. In the hold state, signal lines AD_0 through AD_{15}, A_{16}/S_3 through A_{19}/S_6, $\overline{\text{BHE}}$, $\overline{\text{M}}$/IO, DT/$\overline{\text{R}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$, $\overline{\text{DEN}}$ and INTR are all in the high Z state. The 8086 signals external device that it is in this state by switching its HLDA output to logic 1 level.

Next Page

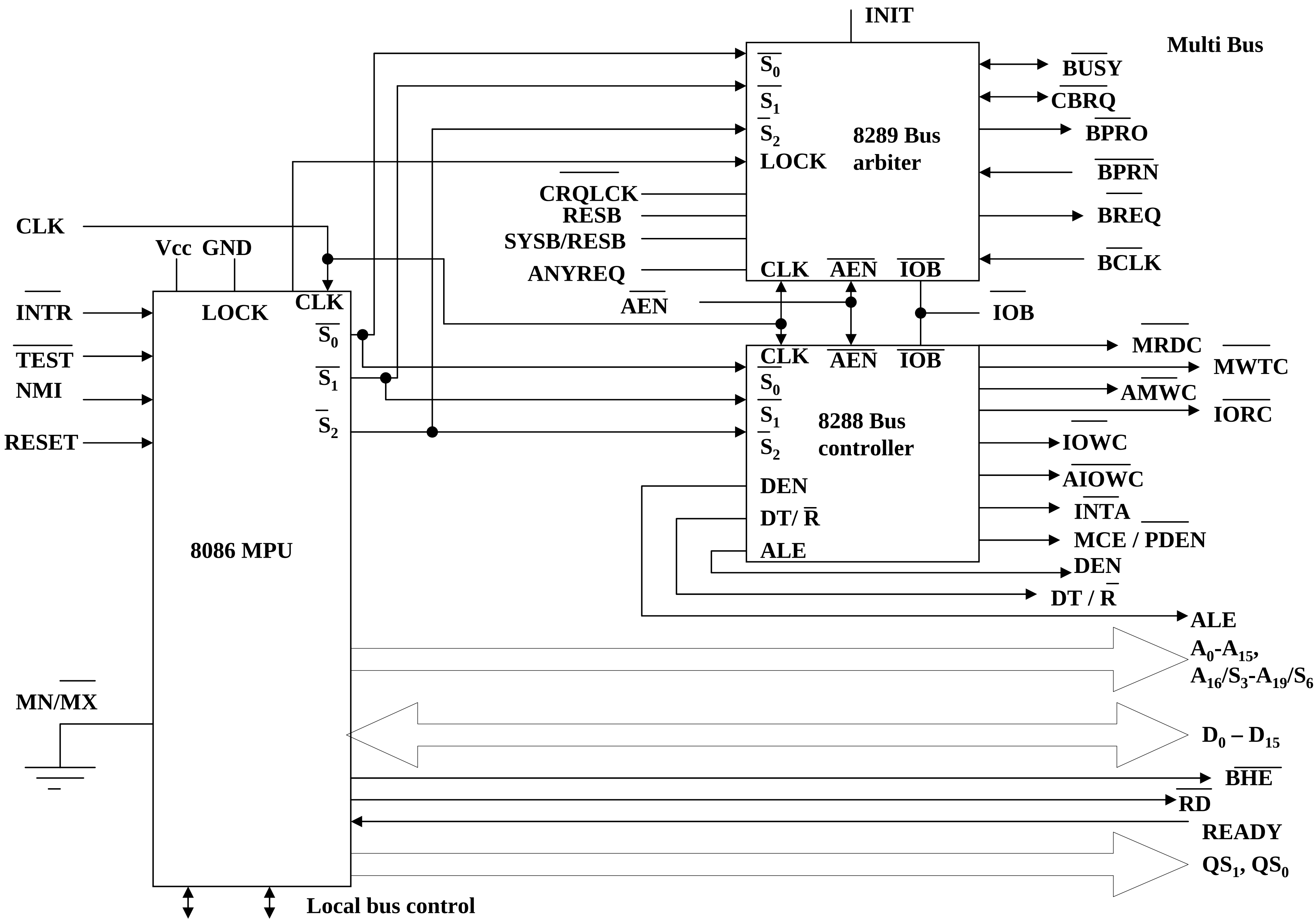
Maximum Mode Interface

- When the 8086 is set for the maximum-mode configuration, it provides signals for implementing a multiprocessor / coprocessor system environment.
- By multiprocessor environment we mean that one microprocessor exists in the system and that each processor is executing its own program.
- Usually in this type of system environment, there are some system resources that are common to all processors.
- They are called as ***global resources***. There are also other resources that are assigned to specific processors. These are known as ***local or private resources***.

Next Page

- Coprocessor also means that there is a second processor in the system. In this two processor does not access the bus at the same time.
- One passes the control of the system bus to the other and then may suspend its operation.
- In the maximum-mode 8086 system, facilities are provided for implementing allocation of global resources and passing bus control to other microprocessor or coprocessor.

Next Page



$\overline{RQ}$ / $\overline{GT}_1$ $\overline{RQ}$ / $\overline{GT}_0$

8086 Maximum mode Block Diagram

- **8288 Bus Controller – Bus Command and Control Signals**: 8086 does not directly provide all the signals that are required to control the memory, I/O and interrupt interfaces.
- Specially the $\overline{\text{WR}}$, M/$\overline{\text{IO}}$, DT/$\overline{\text{R}}$, D$\overline{\text{EN}}$, A$\overline{\text{LE}}$ and I$\overline{\text{NT}}$A, signals are no longer produced by the 8086. Instead it outputs three status signals $\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$ prior to the initiation of each bus cycle. This 3- bit bus status code identifies which type of bus cycle is to follow.
- $\overline{S_2}\,\overline{S_1}\,\overline{S_0}$ are input to the external bus controller device, the bus controller generates the appropriately timed command and control signals.

Next Page

Status Inputs			CPU Cycles	8288 Command
$\overline{S}_2$	$\overline{S}_1$	$\overline{S}_0$		
0	0	0	Interrupt Acknowledge	$\overline{\text{INTA}}$
0	0	1	Read I/O Port	$\overline{\text{IORC}}$
0	1	0	Write I/O Port	$\overline{\text{IOWC}}$, $\overline{\text{AIOWC}}$
0	1	1	Halt	None
1	0	0	Instruction Fetch	$\overline{\text{MRDC}}$
1	0	1	Read Memory	$\overline{\text{MRDC}}$
1	1	0	Write Memory	$\overline{\text{MWTC}}$, $\overline{\text{AMWC}}$
1	1	1	Passive	None

Bus Status Codes

Next Page

- The 8288 produces one or two of these eight command signals for each bus cycles. For instance, when the 8086 outputs the code $\overline{S}_2\overline{S}_1\overline{S}_0$ equals 001, it indicates that an ***I/O read cycle*** is to be performed.
- In the code 111 is output by the 8086, it is signaling that no bus activity is to take place.
- The control outputs produced by the 8288 are DEN, $DT/\overline{R}$ and ALE. These 3 signals provide the same functions as those described for the minimum system mode. This set of bus commands and control signals is compatible with the Multibus and industry standard for interfacing microprocessor systems.

Next Page

- **8289 Bus Arbiter – Bus Arbitration and Lock Signals** : This device permits processors to reside on the system bus. It does this by implementing the Multibus arbitration protocol in an 8086-based system.
- Addition of the 8288 bus controller and 8289 bus arbiter frees a number of the 8086 pins for use to produce control signals that are needed to support multiple processors.
- Bus priority lock ($\overline{\text{LOCK}}$) is one of these signals. It is input to the bus arbiter together with status signals $\overline{S}_0$ through $\overline{S}_2$.

Next Page

- ***The output of 8289 are bus arbitration signals***: *bus busy* ($\overline{\text{BUSY}}$), *common bus request* ($\overline{\text{CBRQ}}$), *bus priority out* ($\overline{\text{BPRO}}$), *bus priority in* ($\overline{\text{BPRN}}$), *bus request* ($\overline{\text{BREQ}}$) and *bus clock* ($\overline{\text{BCLK}}$).
- They correspond to the bus exchange signals of the Multibus and are used to lock other processor off the system bus during the execution of an instruction by the 8086.
- In this way the processor can be assured of uninterrupted access to common system resources such as ***global memory.***

Next Page

- **Queue Status Signals** : Two new signals that are produced by the 8086 in the maximum-mode system are queue status outputs QS_0 and QS_1. Together they form a 2-bit queue status code, QS_1QS_0.
- Following table shows the four different queue status.

Next Page

QS_1	QS_0	Queue Status
0 (low)	0	No Operation. During the last clock cycle, nothing was taken from the queue.
0	1	First Byte. The byte taken from the queue was the first byte of the instruction.
1 (high)	0	Queue Empty. The queue has been reinitialized as a result of the execution of a transfer instruction.
1	1	Subsequent Byte. The byte taken from the queue was a subsequent byte of the instruction.

Queue status codes

Next Page

- **Local Bus Control Signal – Request / Grant Signals**: In a maximum mode configuration, the minimum mode HOLD, HLDA interface is also changed. These two are replaced by request/grant lines $\overline{RQ}/\ \overline{GT}_0$ and $\overline{RQ}/\ \overline{GT}_1$, respectively. They provide a prioritized bus access mechanism for accessing the local bus.

Internal Registers of 8086

- The 8086 has four groups of the user accessible internal registers. They are the instruction pointer, four data registers, four pointer and index register, four segment registers.
- The 8086 has a total of fourteen 16-bit registers including a 16 bit register called the ***status register***, with 9 of bits implemented for status and control flags.

Next Page

- Most of the registers contain data/instruction offsets within 64 KB memory segment. There are four different 64 KB segments for instructions, stack, data and extra data. To specify where in 1 MB of processor memory these 4 segments are located the processor uses four segment registers:
- **Code segment** (CS) is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly. The CS register is automatically updated during far jump, far call and far return instructions.

Next Page

- **Stack segment** (SS) is a 16-bit register containing address of 64KB segment with program stack. By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction.
- **Data segment** (DS) is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment. DS register can be changed directly using POP and LDS instructions.

Next Page

- **Extra segment** (ES) is a 16-bit register containing address of 64KB segment, usually with program data. By default, the processor assumes that the DI register references the ES segment in string manipulation instructions. ES register can be changed directly using POP and LES instructions.
- It is possible to change default segments used by general and index registers by prefixing instructions with a CS, SS, DS or ES prefix.
- All general registers of the 8086 microprocessor can be used for arithmetic and logic operations. The general registers are:

Next Page

- **Accumulator** register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the low-order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations and string manipulation.
- **Base** register consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. BL in this case contains the low-order byte of the word, and BH contains the high-order byte. BX register usually contains a data pointer used for based, based indexed or register indirect addressing.

Next Page

- **Count** register consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX. When combined, CL register contains the low-order byte of the word, and CH contains the high-order byte. Count register can be used in Loop, shift/rotate instructions and as a counter in string manipulation,.
- **Data** register consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX. When combined, DL register contains the low-order byte of the word, and DH contains the high-order byte. Data register can be used as a port number in I/O operations. In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.

Next Page

- The following registers are both general and index registers:
- **Stack Pointer** (SP) is a 16-bit register pointing to program stack.
- **Base Pointer** (BP) is a 16-bit register pointing to data in stack segment. BP register is usually used for based, based indexed or register indirect addressing.
- **Source Index** (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data address in string manipulation instructions.

Next Page

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

Other registers:

- **Instruction Pointer** (IP) is a 16-bit register.
- **Flags** is a 16-bit register containing 9 one bit flags.
- **Overflow Flag** (OF) - set if the result is too large positive number, or is too small negative number to fit into destination operand.

Next Page

- **Direction Flag** (DF) - if set then string manipulation instructions will auto-decrement index registers. If cleared then the index registers will be auto-incremented.
- **Interrupt-enable Flag** (IF) - setting this bit enables maskable interrupts.
- **Single-step Flag** (TF) - if set then single-step interrupt will occur after the next instruction.
- **Sign Flag** (SF) - set if the most significant bit of the result is set.
- **Zero Flag** (ZF) - set if the result is zero.

Next Page

- **Auxiliary carry Flag** (AF) - set if there was a carry from or borrow to bits 0-3 in the AL register.
- **Parity Flag** (PF) - set if parity (the number of "1" bits) in the low-order byte of the result is even.
- **Carry Flag** (CF) - set if there was a carry from or borrow to the most significant bit during last result calculation.

Addressing Modes

- **Implied** - the data value/data address is implicitly associated with the instruction.
- **Register** - references the data in a register or in a register pair.
- **Immediate** - the data is provided in the instruction.
- **Direct** - the instruction operand specifies the memory address where data is located.
- **Register indirect** - instruction specifies a register containing an address, where data is located. This addressing mode works with SI, DI, BX and BP registers.
- **Based** :- 8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP), the resulting value is a pointer to location where data resides.

Next Page

- **Indexed** :- 8-bit or 16-bit instruction operand is added to the contents of an index register (SI or DI), the resulting value is a pointer to location where data resides.
- **Based Indexed** :- the contents of a base register (BX or BP) is added to the contents of an index register (SI or DI), the resulting value is a pointer to location where data resides.
- **Based Indexed with displacement** :- 8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP) and index register (SI or DI), the resulting value is a pointer to location where data resides.

Memory

- Program, data and stack memories occupy the same memory space. As the most of the processor instructions use 16-bit pointers the processor can effectively address only 64 KB of memory.
- To access memory outside of 64 KB the CPU uses special segment registers to specify where the code, stack and data 64 KB segments are positioned within 1 MB of memory (see the "Registers" section below).
- 16-bit pointers and data are stored as:
  address: low-order byte
  address+1: high-order byte

Next Page

- 32-bit addresses are stored in "segment: offset" format as:
  address: low-order byte of segment
  address+1: high-order byte of segment
  address+2: low-order byte of offset
  address+3: high-order byte of offset

- Physical memory address pointed by segment: offset pair is calculated as:
- address = (<segment> * 16) + <offset>

Next Page

- **Program memory -** program can be located anywhere in memory. Jump and call instructions can be used for short jumps within currently selected 64 KB code segment, as well as for far jumps anywhere within 1 MB of memory.
- All conditional jump instructions can be used to jump within approximately +127 to -127 bytes from current instruction.
- **Data memory -** the processor can access data in any one out of 4 available segments, which limits the size of accessible memory to 256 KB (if all four segments point to different 64 KB blocks).

Next Page

- Accessing data from the Data, Code, Stack or Extra segments can be usually done by prefixing instructions with the DS:, CS:, SS: or ES: (some registers and instructions by default may use the ES or SS segments instead of DS segment).
- Word data can be located at odd or even byte boundaries. The processor uses two memory accesses to read 16-bit word located at odd byte boundaries. Reading word data from even byte boundaries requires only one memory access.

Next Page

- **Stack memory** can be placed anywhere in memory. The stack can be located at odd memory addresses, but it is not recommended for performance reasons (see "Data Memory" above).

Reserved locations:

- 0000h - 03FFh are reserved for interrupt vectors. Each interrupt vector is a 32-bit pointer in format segment: offset.
- FFFF0h - FFFFFh - after RESET the processor always starts program execution at the FFFF0h address.

Interrupts

The processor has the following interrupts:

- **INTR** is a maskable hardware interrupt. The interrupt can be enabled/disabled using STI/CLI instructions or using more complicated method of updating the FLAGS register with the help of the POPF instruction.
- When an interrupt occurs, the processor stores FLAGS register into stack, disables further interrupts, fetches from the bus one byte representing interrupt type, and jumps to interrupt processing routine address of which is stored in location 4 * <interrupt type>. Interrupt processing routine should return with the IRET instruction.

Next Page

- **NMI** is a non-maskable interrupt. Interrupt is processed in the same way as the INTR interrupt. Interrupt type of the NMI is 2, i.e. the address of the NMI processing routine is stored in location 0008h. This interrupt has higher priority then the maskable interrupt.
- **Software interrupts** can be caused by:
- INT instruction - breakpoint interrupt. This is a type 3 interrupt.
- INT <interrupt number> instruction - any one interrupt from available 256 interrupts.
- INTO instruction - interrupt on overflow

Next Page

- Single-step interrupt - generated if the TF flag is set. This is a type 1 interrupt. When the CPU processes this interrupt it clears TF flag before calling the interrupt processing routine.
- **Processor exceptions**: Divide Error (Type 0), Unused Opcode (type 6) and Escape opcode (type 7).
- Software interrupt processing is the same as for the hardware interrupts.

Minimum Mode 8086 System

- In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its $\mathrm{MN}/\overline{\mathrm{MX}}$ pin to logic 1.
- In this mode, all the control signals are given out by the microprocessor chip itself. There is a single microprocessor in the minimum mode system.
- The remaining components in the system are latches, transreceivers, clock generator, memory and I/O devices. Some type of chip selection logic may be required for selecting memory or I/O devices, depending upon the address map of the system.

- Latches are generally buffered output D-type flip-flops like 74LS373 or 8282. They are used for separating the valid address from the multiplexed address/data signals and are controlled by the ALE signal generated by 8086.
- Transreceivers are the bidirectional buffers and some times they are called as data amplifiers. They are required to separate the valid data from the time multiplexed address/data signals.
- They are controlled by two signals namely, $\overline{DEN}$ and DT/$\overline{R}$.

- The $\overline{\text{DEN}}$ signal indicates the direction of data, i.e. from or to the processor. The system contains memory for the monitor and users program storage.
- Usually, EPROM are used for monitor storage, while RAM for users program storage. A system may contain I/O devices.

- The clock generator generates the clock from the crystal oscillator and then shapes it and divides to make it more precise so that it can be used as an accurate timing reference for the system.
- The clock generator also synchronizes some external signal with the system clock. The general system organisation is as shown in below fig.
- It has 20 address lines and 16 data lines, the 8086 CPU requires three octal address latches and two octal data buffers for the complete address and data separation.

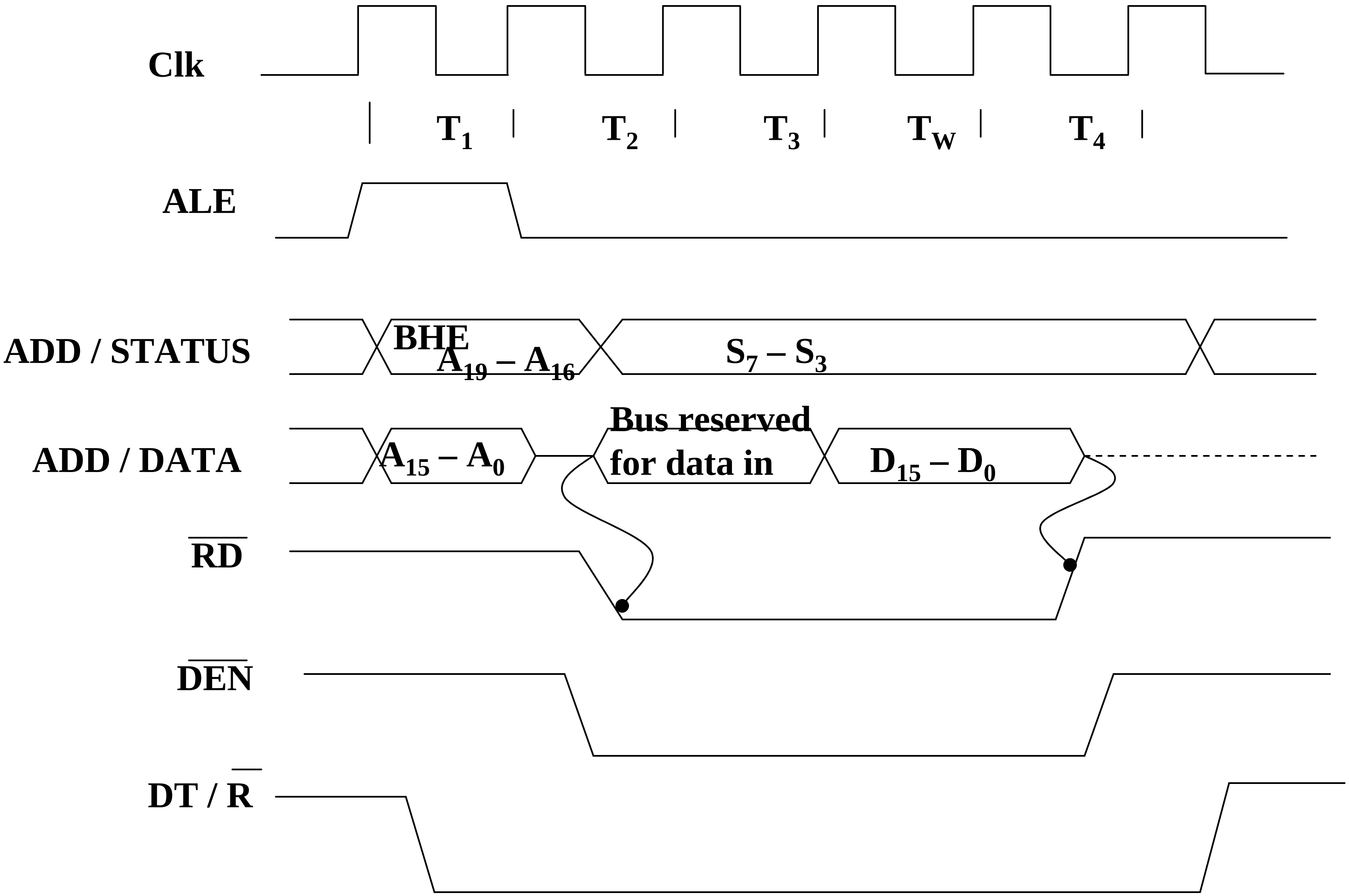
- The working of the minimum mode configuration system can be better described in terms of the timing diagrams rather than qualitatively describing the operations.
- The opcode fetch and read cycles are similar. Hence the timing diagram can be categorized in two parts, the first is the timing diagram for read cycle and the second is the timing diagram for write cycle.
- The read cycle begins in T_1 with the assertion of address latch enable (ALE) signal and also $M / \overline{IO}$ signal. During the negative going edge of this signal, the valid address is latched on the local bus.

- The $\overline{\text{BHE}}$ and $\overline{\text{A}_0}$ signals address low, high or both bytes. From T_1 to T_4 , the M/IO signal indicates a memory or I/O operation.
- At T_2, the address is removed from the local bus and is sent to the output. The bus is then tristated. The read ($\overline{\text{RD}}$) control signal is also activated in T_2.
- The read ($\overline{\text{RD}}$) signal causes the address device to enable its data bus drivers. After $\overline{\text{RD}}$ goes low, the valid data is available on the data bus.
- The addressed device will drive the READY line high. When the processor returns the read signal to high level, the addressed device will again tristate its bus drivers.

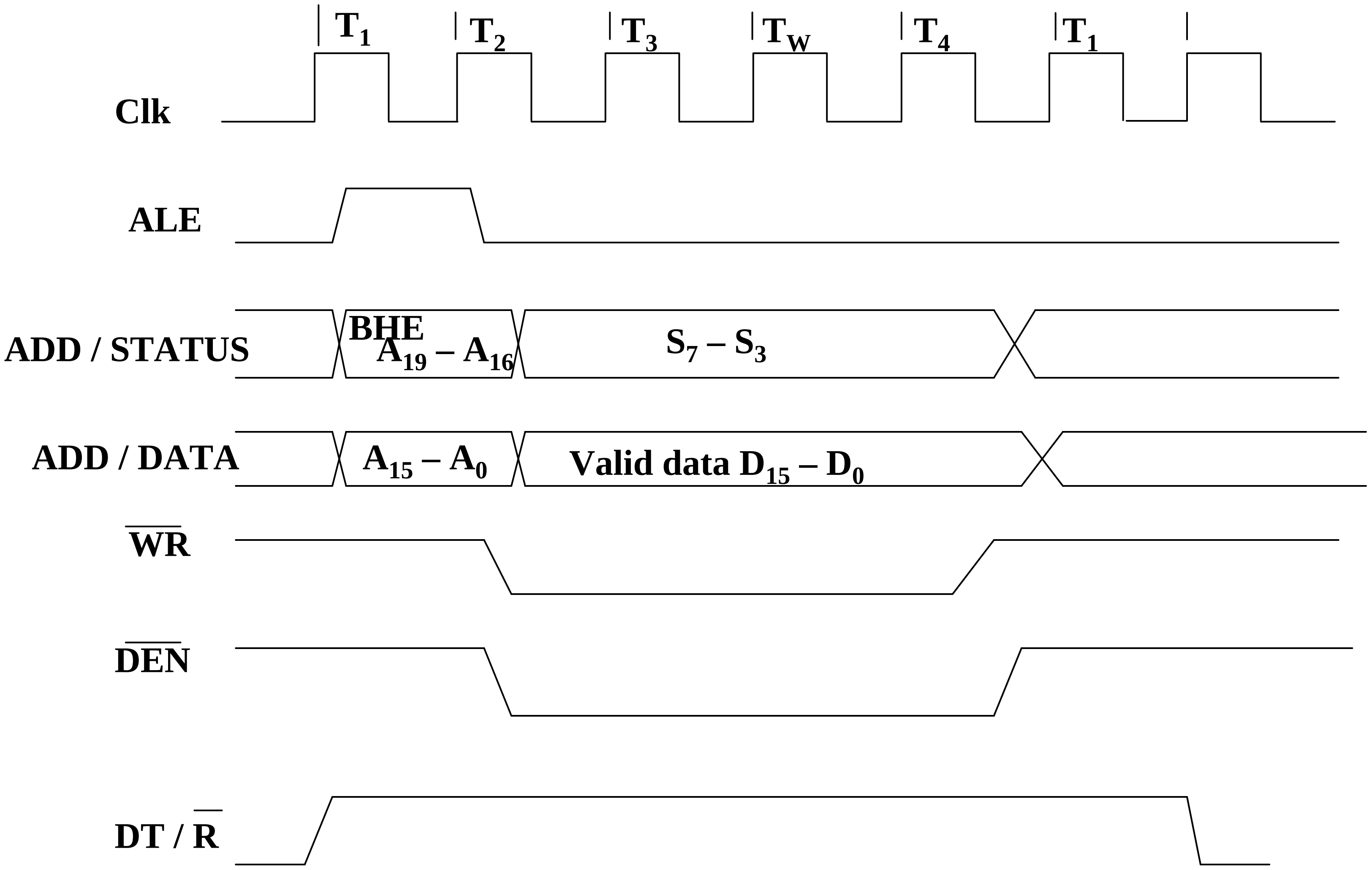
- A write cycle also begins with the assertion of ALE and the emission of the address. The M/$\overline{\text{IO}}$ signal is again asserted to indicate a memory or I/O operation. In T_2, after sending the address in T_1, the processor sends the data to be written to the addressed location.
- The data remains on the bus until middle of T_4 state. The $\overline{\text{WR}}$ becomes active at the beginning of T_2 (unlike $\overline{\text{RD}}$ is somewhat delayed in T_2 to provide time for floating).
- The $\overline{\text{BHE}}$ and A_0 signals are used to select the proper byte or bytes of memory or I/O word to be read or write.
- The M/$\overline{\text{IO}}$, $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals indicate the type of data transfer as specified in table below.

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

Data Transfer table

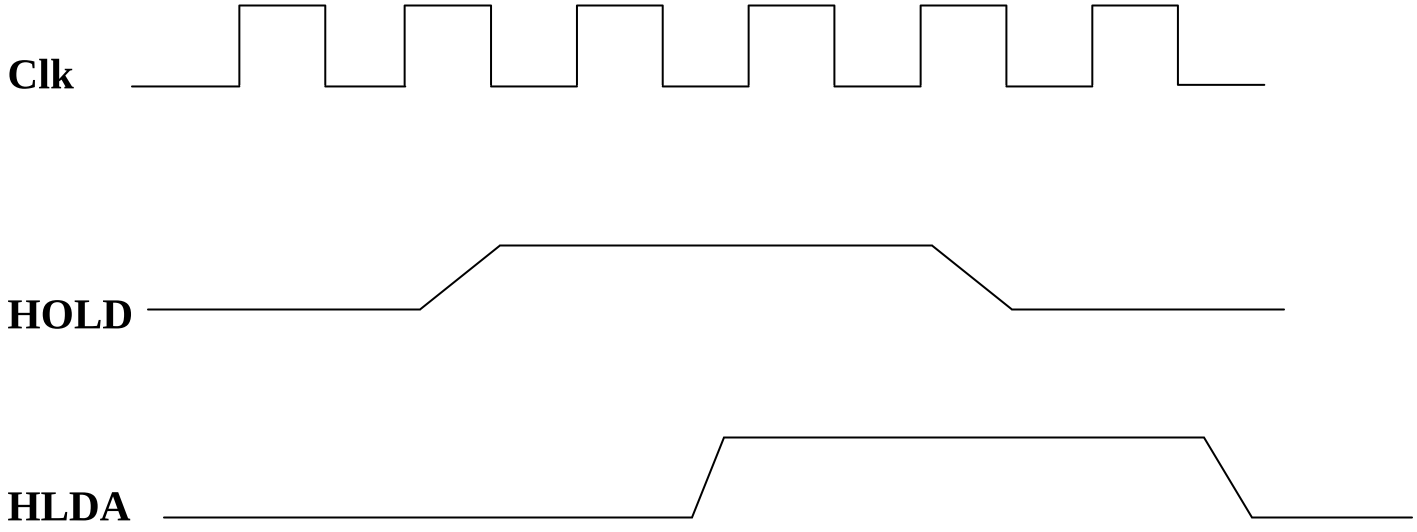


Read Cycle Timing Diagram for Minimum Mode



Write Cycle Timing Diagram for Minimum Mode

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



Bus Request and Bus Grant Timings in Minimum Mode System

Maximum Mode 8086 System

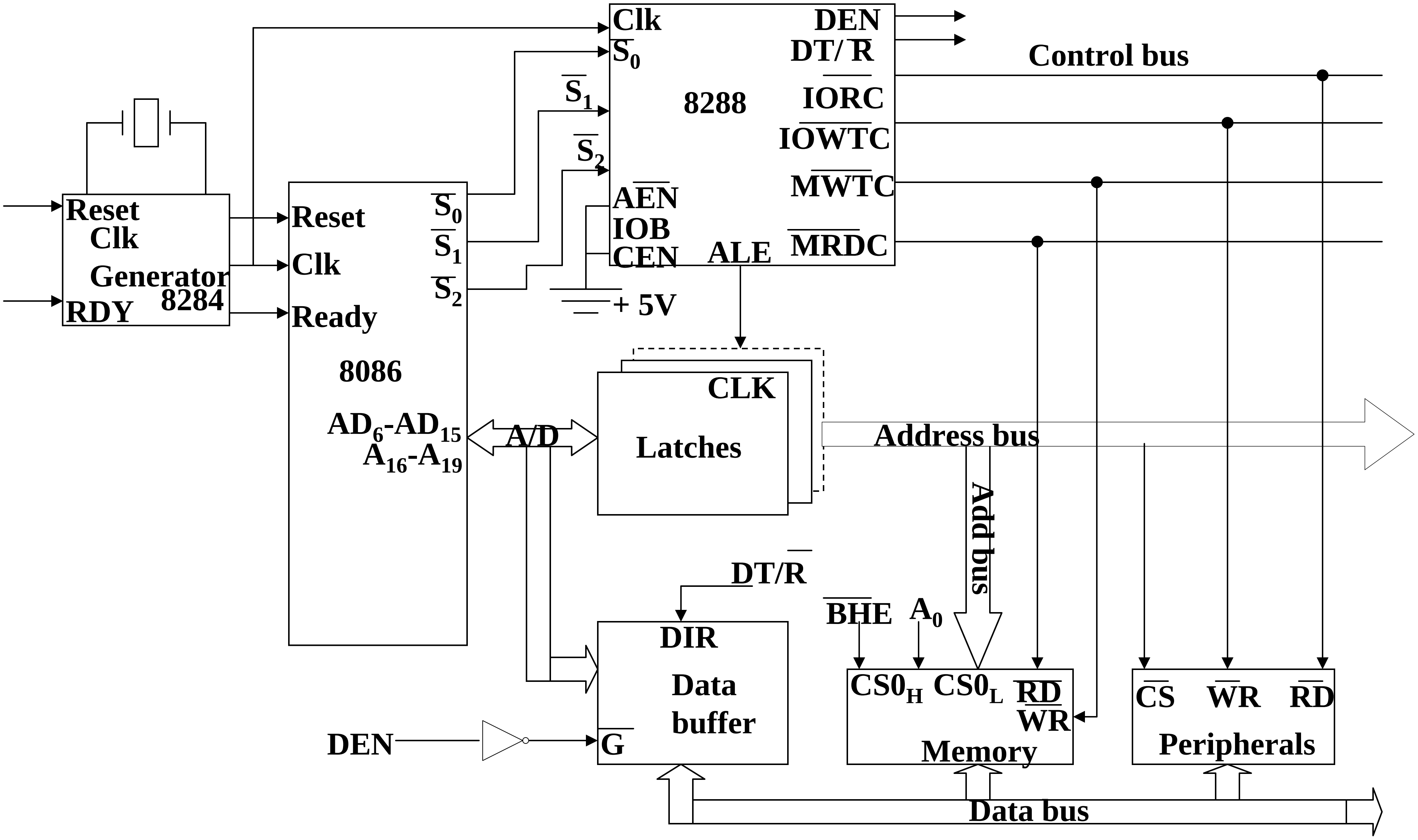
- In the maximum mode, the 8086 is operated by strapping the MN/$\overline{\text{MX}}$ pin to ground.
- In this mode, the processor derives the status signal $\overline{S}_2$, $\overline{S}_1$, $\overline{S}_0$. Another chip called bus controller derives the control signal using this status information .
- In the maximum mode, there may be more than one microprocessor in the system configuration.
- The components in the system are same as in the minimum mode system.

- The basic function of the bus controller chip IC8288, is to derive control signals like RD and WR (for memory and I/O devices), $\overline{\text{DEN}}$, DT/$\overline{\text{R}}$, ALE etc. using the information by the processor on the status lines.
- The bus controller chip has input lines $\overline{\text{S}}_2$, $\overline{\text{S}}_1$, $\overline{\text{S}}_0$ and CLK. These inputs to 8288 are driven by CPU.
- It derives the outputs ALE, $\overline{\text{DEN}}$, DT/$\overline{\text{R}}$, $\overline{\text{MRDC}}$, $\overline{\text{MWTC}}$, $\overline{\text{AMWC}}$, $\overline{\text{IORC}}$, $\overline{\text{IOWC}}$ and $\overline{\text{AIOWC}}$. The $\overline{\text{AEN}}$, IOB and CEN pins are specially useful for multiprocessor systems.

- AEN and $\overline{\text{IOB}}$ are generally grounded. CEN pin is usually tied to +5V. The significance of the MCE/$\overline{\text{PDEN}}$ output depends upon the status of the IOB pin.
- If IOB is grounded, it acts as master cascade enable to control cascade 8259A, else it acts as peripheral data enable used in the multiple bus configurations.
- $\overline{\text{INTA}}$ pin used to issue two interrupt acknowledge pulses to the interrupt controller or to an interrupting device.

- $\overline{\text{IORC}}$, $\overline{\text{IOWC}}$ are I/O read command and I/O write command signals respectively . These signals enable an IO interface to read or write the data from or to the address port.
- The $\overline{\text{MRDC}}$, $\overline{\text{MWTC}}$ are memory read command and memory write command signals respectively and may be used as memory read or write signals.
- All these command signals instructs the memory to accept or send data from or to the bus.
- For both of these write command signals, the advanced signals namely $\overline{\text{AIOWC}}$ and $\overline{\text{AMWTC}}$ are available.

- They also serve the same purpose, but are activated one clock cycle earlier than the $\overline{\text{IOWC}}$ and $\overline{\text{MWTC}}$ signals respectively.
- The maximum mode system timing diagrams are divided in two portions as read (input) and write (output) timing diagrams.
- The address/data and address/status timings are similar to the minimum mode.
- ALE is asserted in T_1, just like minimum mode. The only difference lies in the status signal used and the available control and advanced command signals.

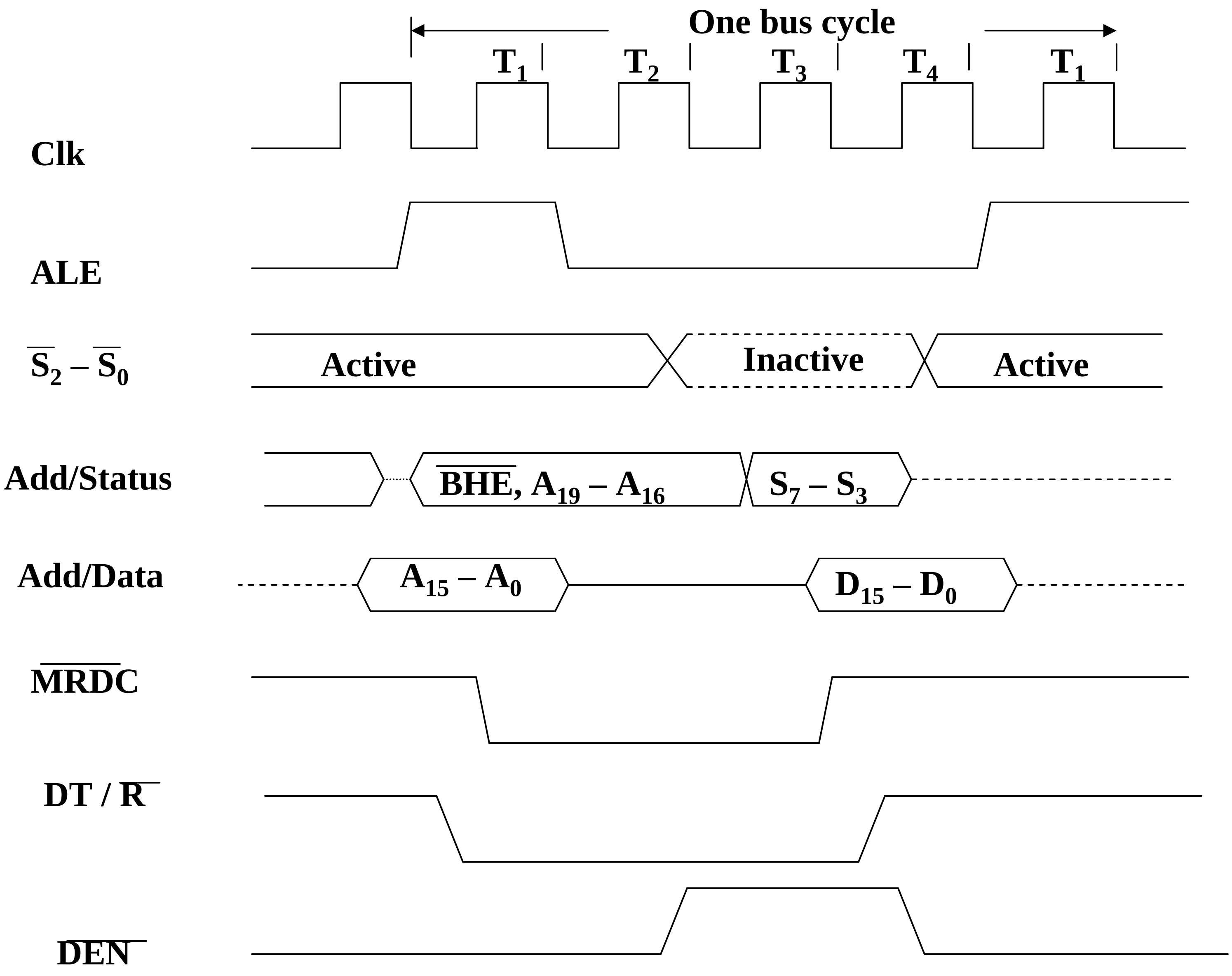


Maximum Mode 8086 System.

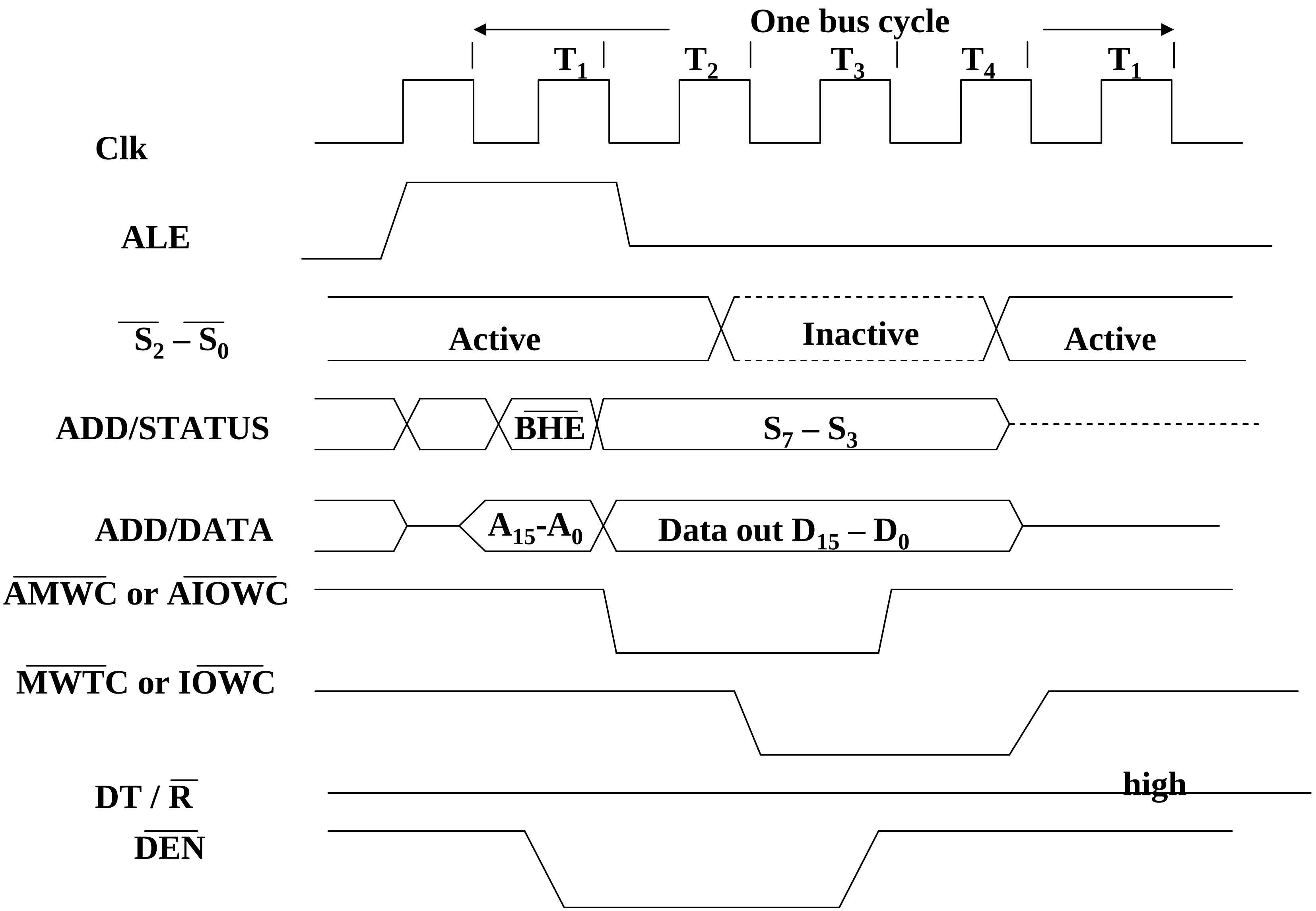
- Here the only difference between in timing diagram between minimum mode and maximum mode is the status signals used and the available control and advanced command signals.
- $\overline{R_0}$, $\overline{S_1}$, $\overline{S_2}$ are set at the beginning of bus cycle.8288 bus controller will output a pulse as on the ALE and apply a required signal to its DT / R pin during T_1.

- In T_2, 8288 will set $\overline{\text{DEN}}$=1 thus enabling transceivers, and for an input it will activate $\overline{\text{MRDC}}$ or $\overline{\text{IORC}}$. These signals are activated until T_4. For an output, the $\overline{\text{AMWC}}$ or $\overline{\text{AIOWC}}$ is activated from T_2 to T_4 and $\overline{\text{MWTC}}$ or $\overline{\text{IOWC}}$ is activated from T_3 to T_4.
- The status bit S_0 to S_2 remains active until T_3 and become passive during T_3 and T_4.
- If reader input is not activated before T_3, wait state will be inserted between T_3 and T_4.

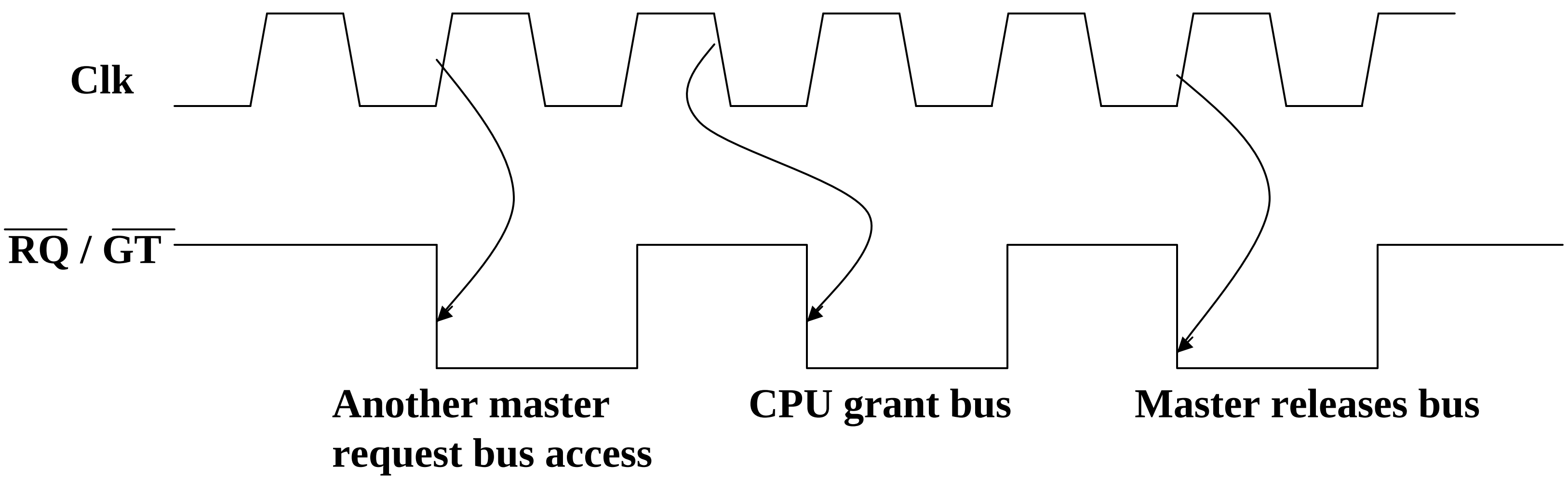
- **Timings for $\overline{RQ}$/ $\overline{GT}$ Signals :** The request/grant response sequence contains a series of three pulses. The request/grant pins are checked at each rising pulse of clock input.
- When a request is detected and if the condition for HOLD request are satisfied, the processor issues a grant pulse over the $\overline{RQ}/\overline{GT}$ pin immediately during T_4 (current) or T_1 (next) state.
- When the requesting master receives this pulse, it accepts the control of the bus, it sends a release pulse to the processor using $\overline{RQ}/\overline{GT}$ pin.



Memory Read Timing in Maximum Mode



Memory Write Timing in Maximum mode.



$\overline{\text{RQ}}/\overline{\text{GT}}$ Timings in Maximum Mode.

MODULE 1

ARCHITECTURE OF MICROPROCESSORS

Contents

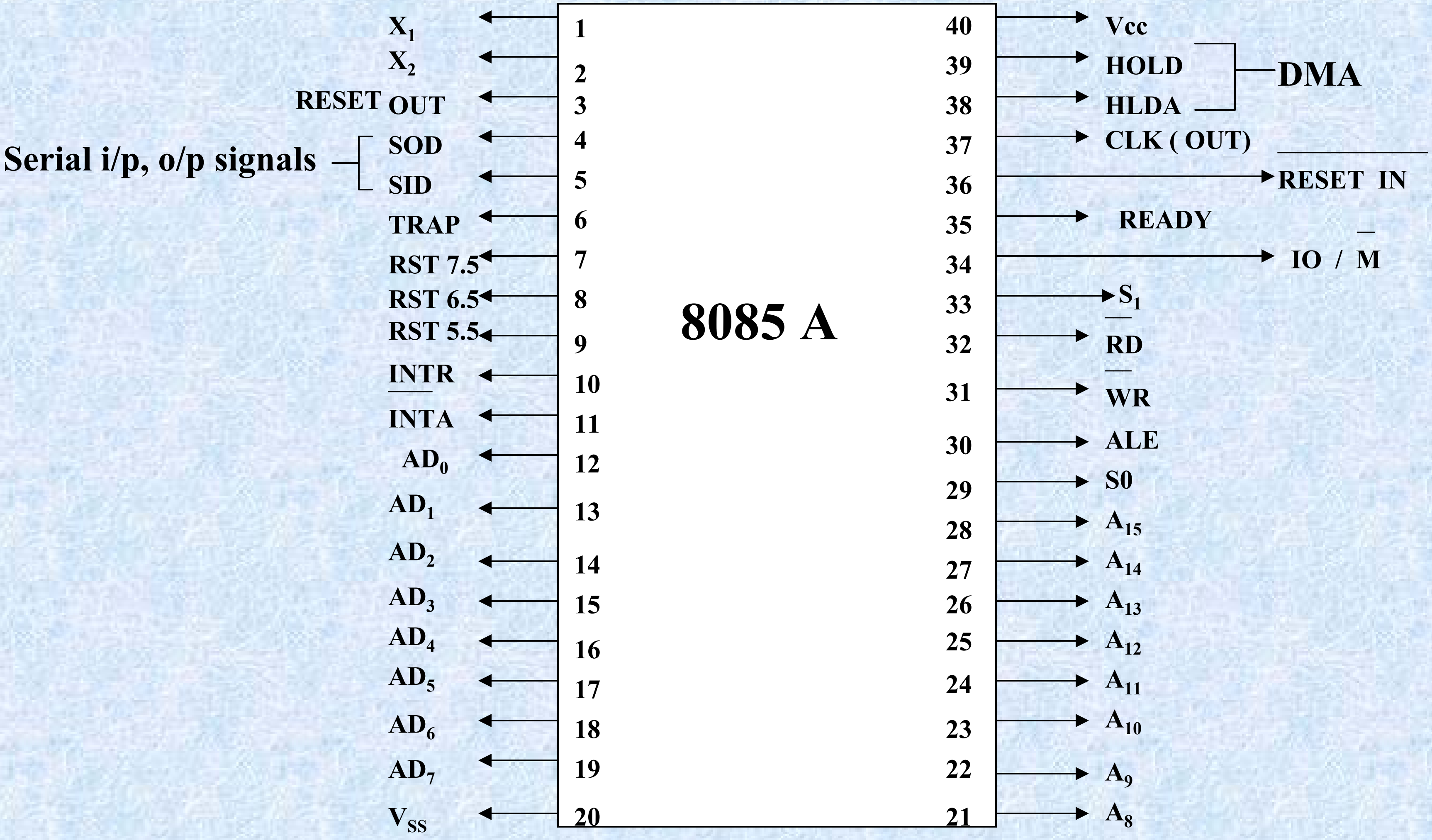
- General definitions
- Overview of 8085 microprocessor
- Overview of 8086 microprocessor
- Signals and pins of 8086 microprocessor

Overview of 8085 microprocessor

➢ 8085 Architecture

- Pin Diagram
- Functional Block Diagram

Pin Diagram of 8085



8085 A

Serial i/p, o/p signals

X_1 1

X_2 2

RESET OUT 3

SOD 4

SID 5

TRAP 6

RST 7.5 7

RST 6.5 8

RST 5.5 9

INTR 10

$\overline{\text{INTA}}$ 11

AD_0 12

AD_1 13

AD_2 14

AD_3 15

AD_4 16

AD_5 17

AD_6 18

AD_7 19

V_{SS} 20

40 Vcc

39 HOLD

38 HLDA

DMA

37 CLK (OUT)

36 $\overline{\text{RESET IN}}$

35 READY

34 IO / $\overline{\text{M}}$

33 S_1

32 $\overline{\text{RD}}$

31 $\overline{\text{WR}}$

30 ALE

29 S0

28 A_{15}

27 A_{14}

26 A_{13}

25 A_{12}

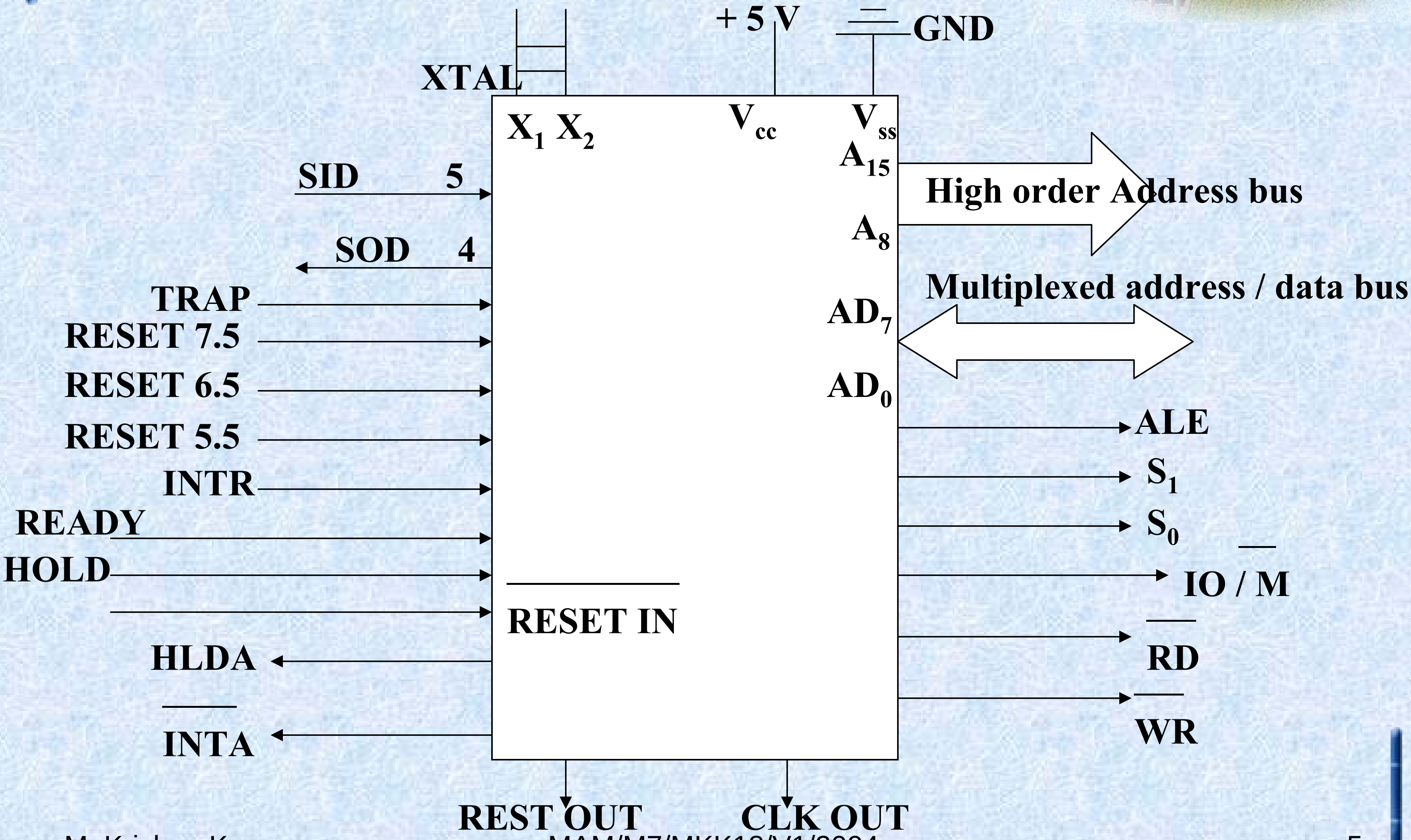
24 A_{11}

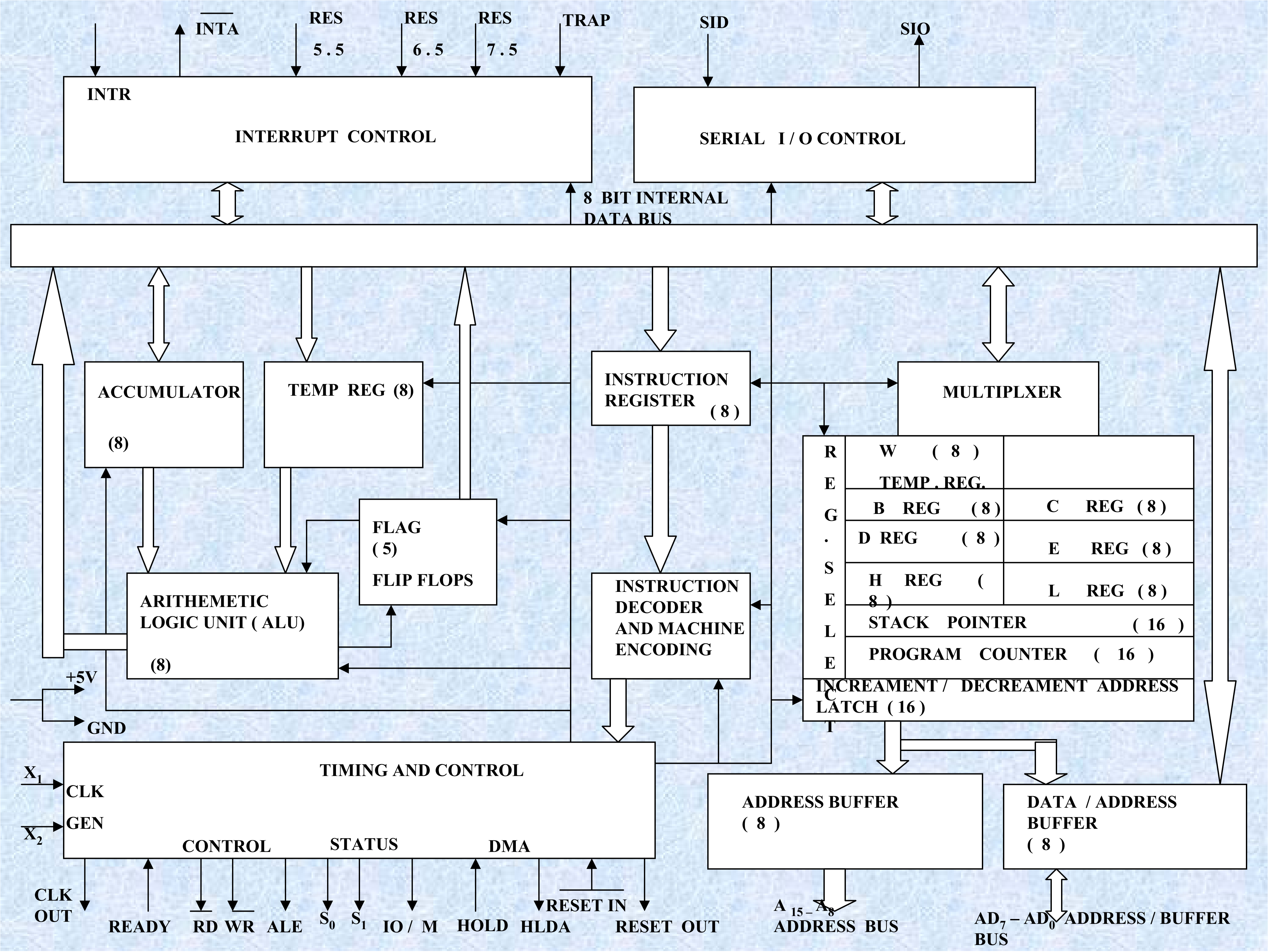
23 A_{10}

22 A_9

21 A_8

Signal Groups of 8085




INTA
RES 5.5
RES 6.5
RES 7.5
TRAP
SID
SIO
INTR
INTERRUPT CONTROL
SERIAL I / O CONTROL
8 BIT INTERNAL DATA BUS
ACCUMULATOR (8)
TEMP REG (8)
INSTRUCTION REGISTER (8)
MULTIPLXER
FLAG (5) FLIP FLOPS
ARITHEMETIC LOGIC UNIT (ALU) (8)
INSTRUCTION DECODER AND MACHINE ENCODING
REG. SELECT
W (8) TEMP . REG.
B REG (8)
C REG (8)
D REG (8)
E REG (8)
H REG (8)
L REG (8)
STACK POINTER (16)
PROGRAM COUNTER (16)
INCREAMENT / DECREAMENT ADDRESS LATCH (16)
+5V
GND
TIMING AND CONTROL
X1
X2
CLK GEN
CONTROL
STATUS
DMA
CLK OUT
READY
RD
WR
ALE
S0
S1
IO / M
HOLD
HLDA
RESET IN
RESET OUT
ADDRESS BUFFER (8)
DATA / ADDRESS BUFFER (8)
A15 – A8 ADDRESS BUS
AD7 – AD0 ADDRESS / BUFFER BUS

Flag Registers

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	**Z**		**AC**		**P**		**CY**

General Purpose Registers

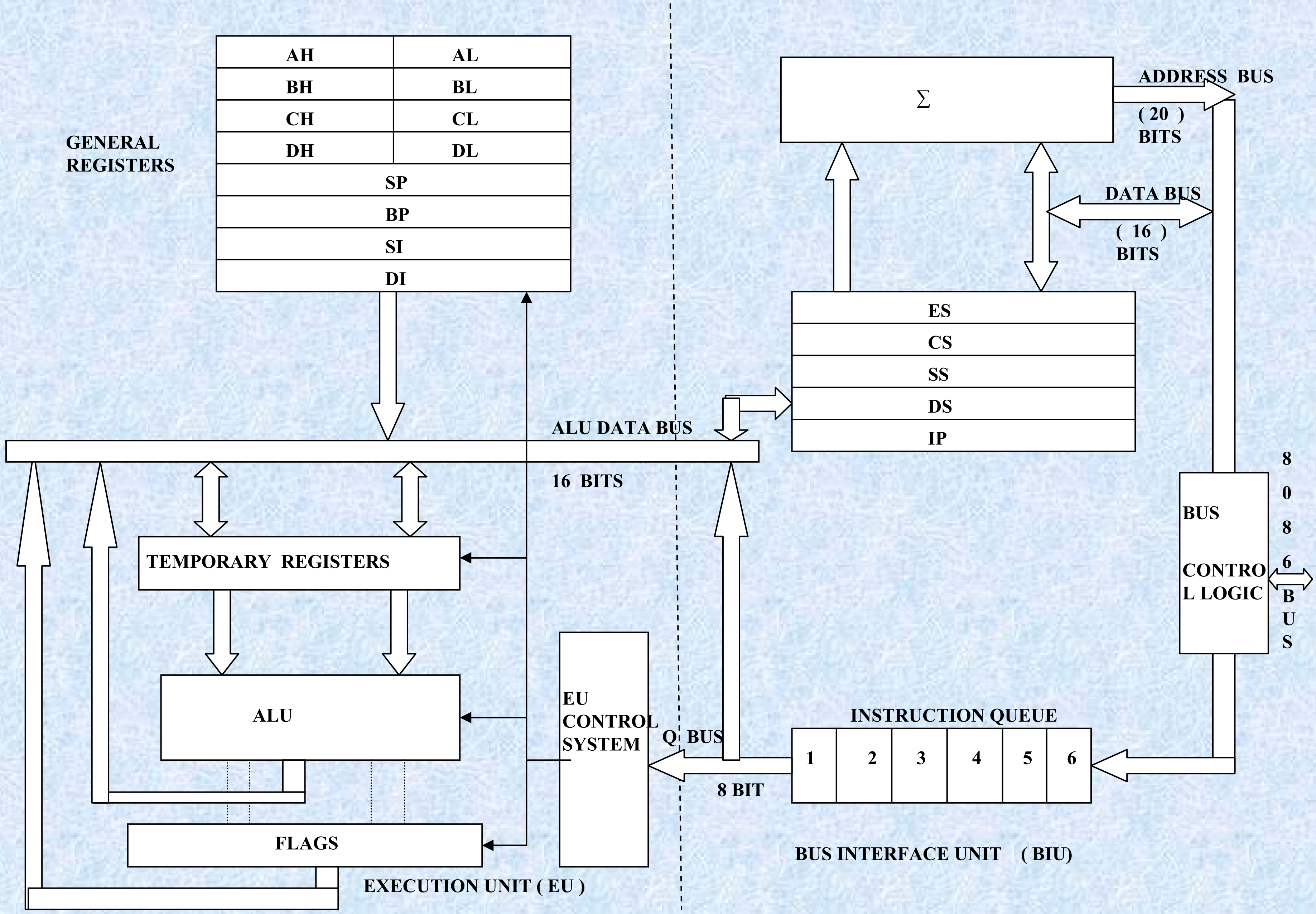
INDIVIDUAL	**B, C, D, E, H, L**
COMBININATON	**B & C, D & E, H & L**

Overview of
8086 Microprocessor

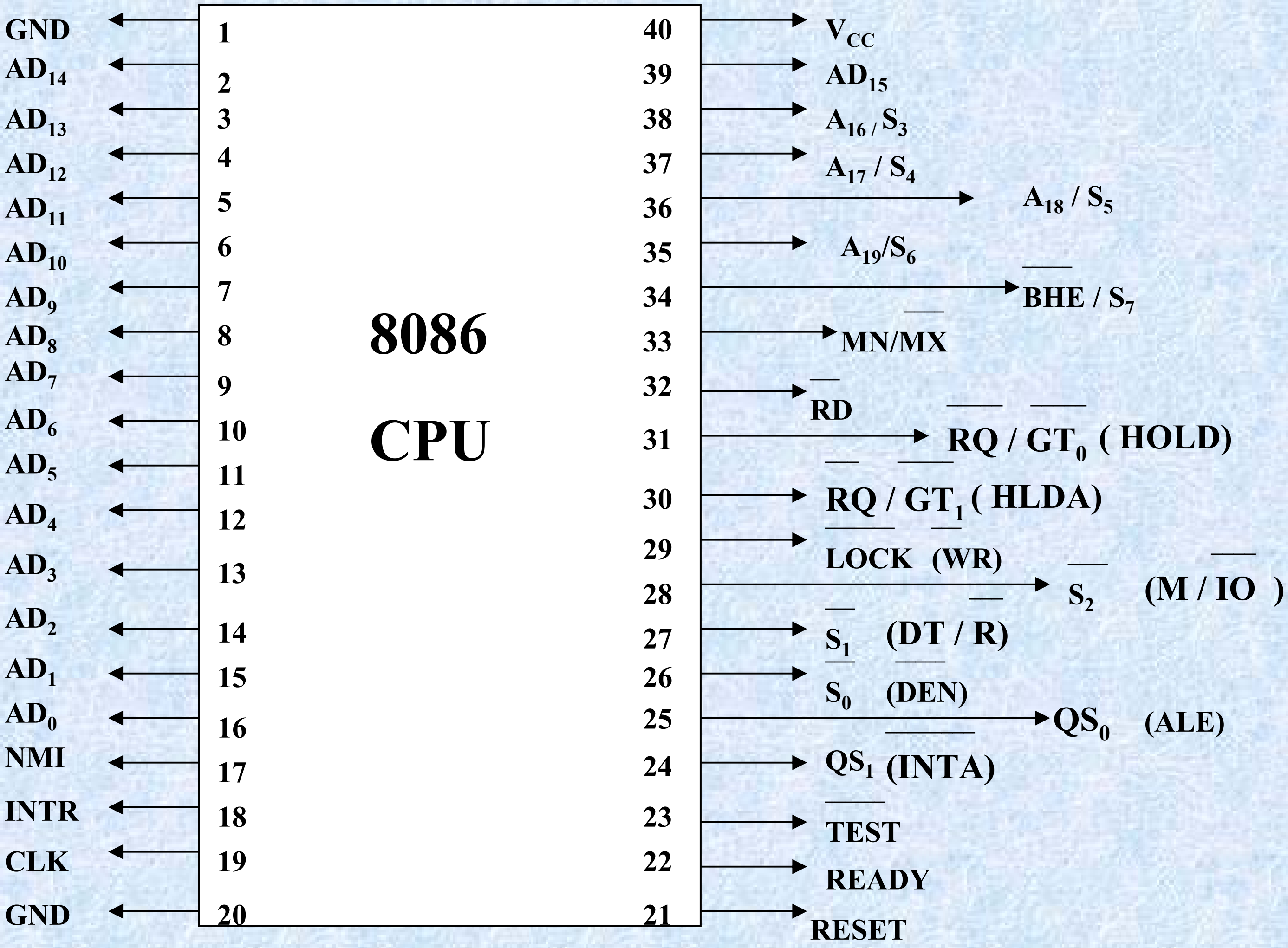
➢ 8086 Architecture

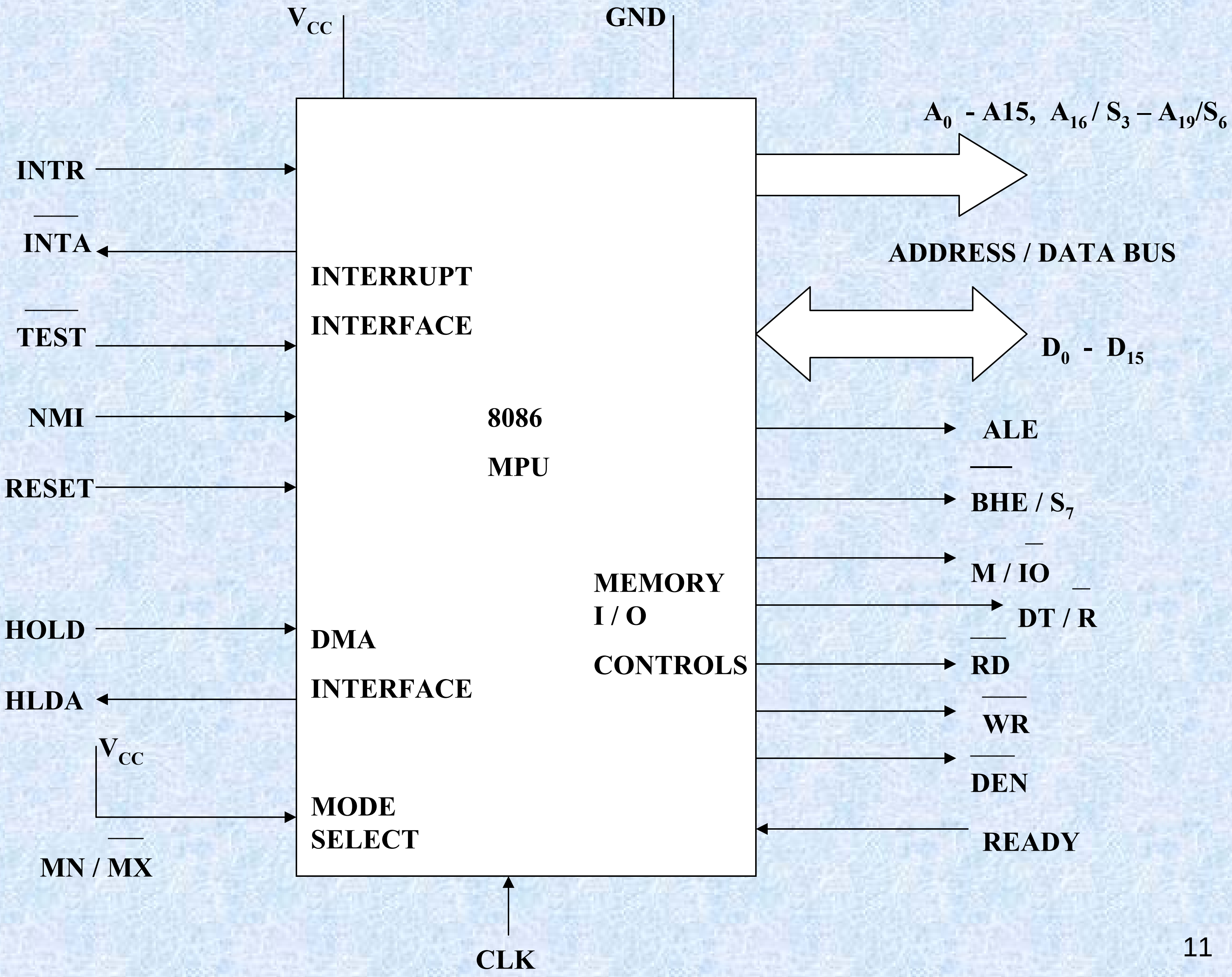
- Pin Diagram

- Functional Block Diagram


GENERAL REGISTERS
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
ALU DATA BUS
16 BITS
TEMPORARY REGISTERS
ALU
FLAGS
EU CONTROL SYSTEM
EXECUTION UNIT (EU)
Σ
ADDRESS BUS
(20) BITS
DATA BUS
(16) BITS
ES
CS
SS
DS
IP
BUS CONTROL LOGIC
8086 BUS
Q BUS
8 BIT
INSTRUCTION QUEUE
1
2
3
4
5
6
BUS INTERFACE UNIT (BIU)

Pin Diagram of 8086




V_{CC}
GND
A_0 - A15, $A_{16}/S_3 - A_{19}/S_6$
INTR
$\overline{INTA}$
ADDRESS / DATA BUS
INTERRUPT
INTERFACE
$\overline{TEST}$
D_0 - D_{15}
NMI
8086
MPU
ALE
RESET
$\overline{BHE}/S_7$
$M/\overline{IO}$
MEMORY
I / O
CONTROLS
$DT/\overline{R}$
HOLD
DMA
INTERFACE
$\overline{RD}$
HLDA
$\overline{WR}$
V_{CC}
$\overline{DEN}$
MODE
SELECT
READY
$MN/\overline{MX}$
CLK

Signal Description of 8086

- The Microprocessor 8086 is a 16-bit CPU available in different clock rates and packaged in a 40 pin CERDIP or plastic package.
- The 8086 operates in single processor or multiprocessor configuration to achieve high performance. The pins serve a particular function in minimum mode (single processor mode) and other function in maximum mode configuration (multiprocessor mode).
- The 8086 signals can be categorised in three groups. The first are the signal having common functions in minimum as well as maximum mode.

- The second are the signals which have special functions for minimum mode and third are the signals having special functions for maximum mode.
- The following signal descriptions are common for both modes.
- **AD_{15}-AD_0** : These are the time multiplexed memory I/O address and data lines.
- Address remains on the lines during T_1 state, while the data is available on the data bus during T_2, T_3, T_w and T_4.
- These lines are active high and float to a tristate during interrupt acknowledge and local bus hold acknowledge cycles.

- **A_{19}/S_6,A_{18}/S_5,A_{17}/S_4,A_{16}/S_3** : These are the time multiplexed address and status lines.
- During T_1 these are the most significant address lines for memory operations.
- During I/O operations, these lines are low. During memory or I/O operations, status information is available on those lines for T_2,T_3,T_w and T_4.
- The status of the interrupt enable flag bit is updated at the beginning of each clock cycle.

- The S_4 and S_3 combinedly indicate which segment register is presently being used for memory accesses as in below fig.
- These lines float to tri-state off during the local bus hold acknowledge. The status line S_6 is always low .
- The address bit are separated from the status bit using latches controlled by the ALE signal.

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

- $\overline{\mathbf{BHE}}/\mathbf{S_7}$: The bus high enable is used to indicate the transfer of data over the higher order (D_{15}-D_8) data bus as shown in table. It goes low for the data transfer over D_{15}-D_8 and is used to derive chip selects of odd address memory bank or peripherals. BHE is low during T_1 for read, write and interrupt acknowledge cycles, whenever a byte is to be transferred on higher byte of data bus. The status information is available during T_2, T_3 and T_4. The signal is active low and tristated during hold. It is low during T_1 for the first pulse of the interrupt acknowledge cycle.

BHE	**A_0**	**Indication**
0	**0**	**Whole word**
0	**1**	**Upper byte from or to even address**
1	**0**	**Lower byte from or to even address**
1	**1**	**None**

- **$\overline{\mathbf{RD}}$ – Read** : This signal on low indicates the peripheral that the processor is performing s memory or I/O read operation. RD is active low and shows the state for T_2, T_3, T_w of any read cycle. The signal remains tristated during the hold acknowledge.
- **READY** : This is the acknowledgement from the slow device or memory that they have completed the data transfer. The signal made available by the devices is synchronized by the 8284A clock generator to provide ready input to the 8086. the signal is active high.

- **INTR-Interrupt Request** : This is a triggered input. This is sampled during the last clock cycles of each instruction to determine the availability of the request. If any interrupt request is pending, the processor enters the interrupt acknowledge cycle.
- This can be internally masked by resulting the interrupt enable flag. This signal is active high and internally synchronized.
- $\overline{\textbf{TEST}}$: This input is examined by a ‘WAIT’ instruction. If the TEST pin goes low, execution will continue, else the processor remains in an idle state. The input is synchronized internally during each clock cycle on leading edge of clock.

- **NMI- Nonmaskable 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. This input is internally synchronized.
- **RESET** : This input causes the processor to terminate the current activity and start execution from FFF0H. The signal is active high and must be active for at least four clock cycles. It restarts execution when the RESET returns low. RESET is also internally synchronized.
- **Vcc** +5V power supply for the operation of the internal circuit.
- **GND** ground for internal circuit.

- **CLK-** Clock Input : The clock input provides the basic timing for processor operation and bus control activity. Its an asymmetric square wave with 33% duty cycle.
- **MN/$\overline{\textbf{MX}}$** : The logic level at this pin decides whether the processor is to operate in either minimum or maximum mode.
- *The following pin functions are for the **minimum mode** operation of 8086.*
- **M/$\overline{\textbf{IO}}$ – Memory/IO** : This is a status line logically equivalent to S_2 in maximum mode. When it is low, it indicates the CPU is having an I/O operation, and when it is high, it indicates that the CPU is having a memory operation. This line becomes active high in the previous T_4 and remains active till final T_4 of the current cycle. It is tristated during local bus “hold acknowledge “.

- **$\overline{\text{INT}}\text{A}$ – Interrupt Acknowledge** : This signal is used as a read strobe for interrupt acknowledge cycles. i.e. when it goes low, the processor has accepted the interrupt.
- **ALE – Address Latch Enable** : This output signal indicates the availability of the valid address on the address/data lines, and is connected to latch enable input of latches. This signal is active high and is never tristated.
- **DT/$\overline{\text{R}}$ – Data Transmit/Receive**: This output is used to decide the direction of data flow through the transreceivers (bidirectional buffers). When the processor sends out data, this signal is high and when the processor is receiving data, this signal is low.

- **DEN – Data Enable** : This signal indicates the availability of valid data over the address/data lines. It is used to enable the transreceivers (bidirectional buffers) to separate the data from the multiplexed address/data signal. It is active from the middle of T_2 until the middle of T_4. This is tristated during ' hold acknowledge' cycle.
- **HOLD, HLDA- 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. HOLD is an asynchronous input, and is should be externally synchronized.
- If the DMA request is made while the CPU is performing a memory or I/O cycle, it will release the local bus during T_4 provided :

1. The request occurs on or before T_2 state of the current cycle.
2. The current cycle is not operating over the lower byte of a word.
3. The current cycle is not the first acknowledge of an interrupt acknowledge sequence.

4. A Lock instruction is not being executed.

- ***The following pin function are applicable for maximum mode operation of 8086***.
- **$\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$ – Status Lines** : These are the status lines which reflect the type of operation, being carried out by the processor. These become activity during T_4 of the previous cycle and active during T_1 and T_2 of the current bus cycles.

S_2	S_1	S_0	Indication
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1	0	0	Code Access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive

- $\overline{\textbf{LOCK}}$: This output pin indicates that other system bus master will be prevented from gaining the system bus, while the $\overline{\text{LOCK}}$ signal is low.
- The $\overline{\text{LOCK}}$ signal is activated by the ‘LOCK’ prefix instruction and remains active until the completion of the next instruction. When the CPU is executing a critical instruction which requires the system bus, the LOCK prefix instruction ensures that other processors connected in the system will not gain the control of the bus.
- The 8086, while executing the prefixed instruction, asserts the bus lock signal output, which may be connected to an external bus controller.

- **QS_1, QS_0 – Queue Status**: These lines give information about the status of the code-prefetch queue. These are active during the CLK cycle after while the queue operation is performed.
- This modification in a simple fetch and execute architecture of a conventional microprocessor offers an added advantage of pipelined processing of the instructions.
- The 8086 architecture has 6-byte instruction prefetch queue. Thus even the largest (6-bytes) instruction can be prefetched from the memory and stored in the prefetch. This results in a faster execution of the instructions.
- In 8085 an instruction is fetched, decoded and executed and only after the execution of this instruction, the next one is fetched.

- By prefetching the instruction, there is a considerable speeding up in instruction execution in 8086. This is known as ***instruction pipelining***.
- At the starting the CS:IP is loaded with the required address from which the execution is to be started. Initially, the queue will be empty an the microprocessor starts a fetch operation to bring one byte (the first byte) of instruction code, if the CS:IP address is odd or two bytes at a time, if the CS:IP address is even.
- The first byte is a complete opcode in case of some instruction (one byte opcode instruction) and is a part of opcode, in case of some instructions (two byte opcode instructions), the remaining part of code lie in second byte.

- But the first byte of an instruction is an opcode. When the first byte from the queue goes for decoding and interpretation, one byte in the queue becomes empty and subsequently the queue is updated.
- The microprocessor does not perform the next fetch operation till at least two bytes of instruction queue are emptied. The instruction execution cycle is never broken for fetch operation. After decoding the first byte, the decoding circuit decides whether the instruction is of single opcode byte or double opcode byte.
- If it is single opcode byte, the next bytes are treated as data bytes depending upon the decoded instruction length, otherwise, the next byte in the queue is treated as the second byte of the instruction opcode.

- The second byte is then decoded in continuation with the first byte to decide the instruction length and the number of subsequent bytes to be treated as instruction data.
- The queue is updated after every byte is read from the queue but the fetch cycle is initiated by BIU only if at least two bytes of the queue are empty and the EU may be concurrently executing the fetched instructions.
- The next byte after the instruction is completed is again the first opcode byte of the next instruction. A similar procedure is repeated till the complete execution of the program.

- The fetch operation of the next instruction is overlapped with the execution of the current instruction. As in the architecture, there are two separate units, namely Execution unit and Bus interface unit.
- While the execution unit is busy in executing an instruction, after it is completely decoded, the bus interface unit may be fetching the bytes of the next instruction from memory, depending upon the queue status.

QS_1	QS_0	Indication
0	0	No operation
0	1	First byte of the opcode from the queue
1	0	Empty queue
1	1	Subsequent byte from the queue

- **$\overline{\mathrm{RQ}}/\overline{\mathrm{GT}}_0$, $\overline{\mathrm{RQ}}/\overline{\mathrm{GT}}_1$ – Request/Grant :** These pins are used by the other local bus master in maximum mode, to force the processor to release the local bus at the end of the processor current bus cycle.
- Each of the pin is bidirectional with $\overline{\mathrm{RQ}}/\overline{\mathrm{GT}}_0$ having higher priority than $\overline{\mathrm{RQ}}/\overline{\mathrm{GT}}_1$.
- $\overline{\mathrm{RQ}}/\overline{\mathrm{GT}}$ pins have internal pull-up resistors and may be left unconnected.
- Request/Grant sequence is as follows:

1. A pulse of one clock wide from another bus master requests the bus access to 8086.

2. During T_4(current) or T_1(next) clock cycle, a pulse one clock wide from 8086 to the requesting master, indicates that the 8086 has allowed the local bus to float and that it will enter the ‘hold acknowledge’ state at next cycle. The CPU bus interface unit is likely to be disconnected from the local bus of the system.
3. A one clock wide pulse from the another master indicates to the 8086 that the hold request is about to end and the 8086 may regain control of the local bus at the next clock cycle. Thus each master to master exchange of the local bus is a sequence of 3 pulses. There must be at least one dead clock cycle after each bus exchange.

- The request and grant pulses are active low.

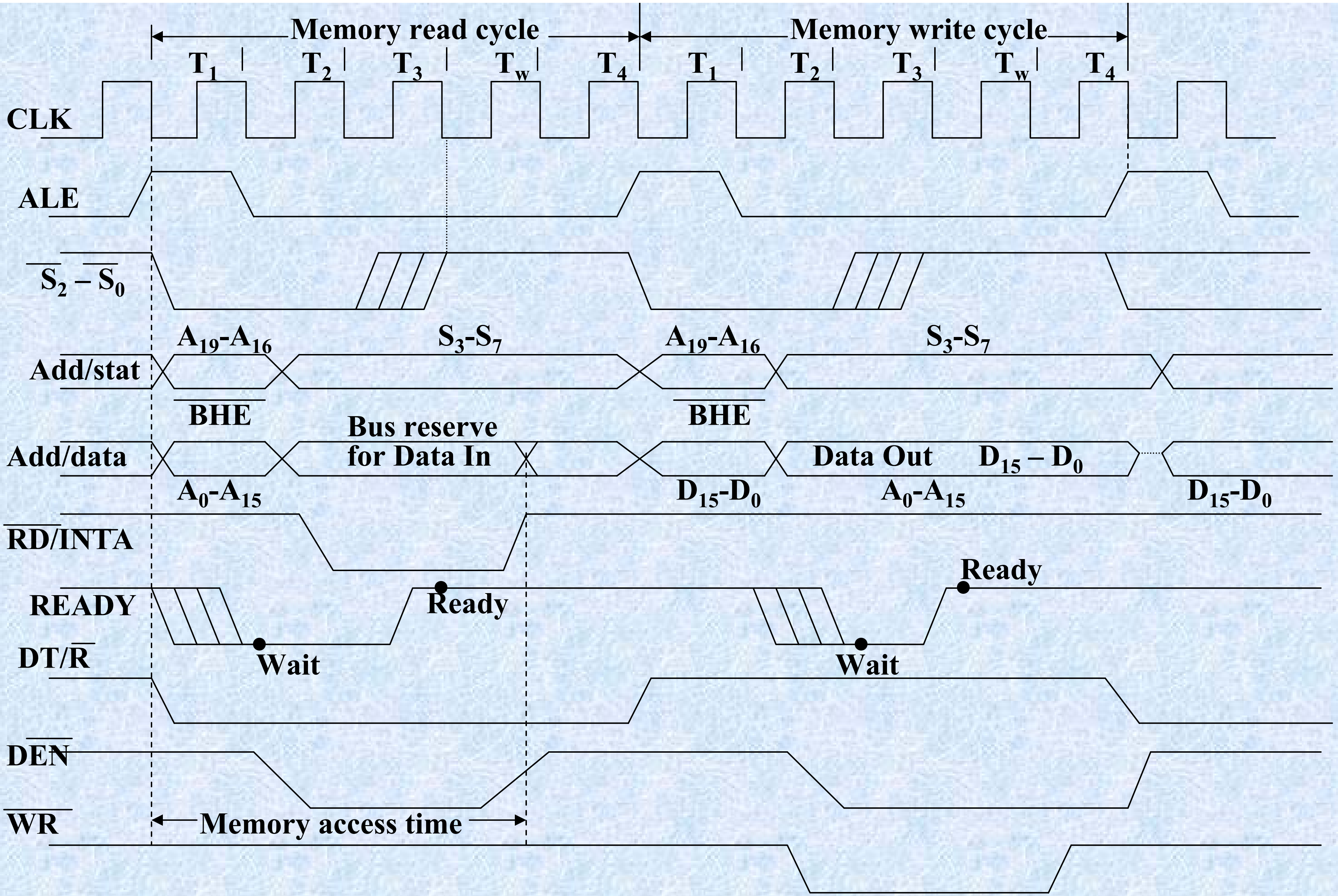
- For the bus request those are received while 8086 is performing memory or I/O cycle, the granting of the bus is governed by the rules as in case of HOLD and HLDA in minimum mode.

General Bus Operation

- The 8086 has a combined address and data bus commonly referred as a time multiplexed address and data bus.
- The main reason behind multiplexing address and data over the same pins is the maximum utilisation of processor pins and it facilitates the use of 40 pin standard DIP package.
- The bus can be demultiplexed using a few latches and transreceivers, when ever required.
- Basically, all the processor bus cycles consist of at least four clock cycles. These are referred to as T_1, T_2, T_3, T_4. The address is transmitted by the processor during T_1. It is present on the bus only for one cycle.

- During T_2, i.e. the next cycle, the bus is tristated for changing the direction of bus for the following data read cycle. The data transfer takes place during T_3, T_4.
- In case, an address device is slow ‘NOT READY’ status the wait status T_w are inserted between T_3 and T_4. These clock states during wait period are called ***idle states*** (T_i), ***wait states*** (T_w) or ***inactive states***. The processor used these cycles for internal housekeeping.
- The address latch enable (ALE) signal is emitted during T_1 by the processor (minimum mode) or the bus controller (maximum mode) depending upon the status of the $MN/\overline{MX}$ input.

- The negative edge of this ALE pulse is used to separate the address and the data or status information. In maximum mode, the status lines $\overline{S}_0$, $\overline{S}_1$ and $\overline{S}_2$ are used to indicate the type of operation.
- Status bits $\overline{S}_3$ to $\overline{S}_7$ are multiplexed with higher order address bits and the BHE signal. Address is valid during T_1 while status bits S_3 to S_7 are valid during T_2 through T_4.



General Bus Operation Cycle in Maximum Mode

8085 Microprocessor

Contents

❖General definitions

❖Overview of 8085 microprocessor

❖Overview of 8086 microprocessor

❖Signals and pins of 8086 microprocessor

- The salient features of 8085 µp are:

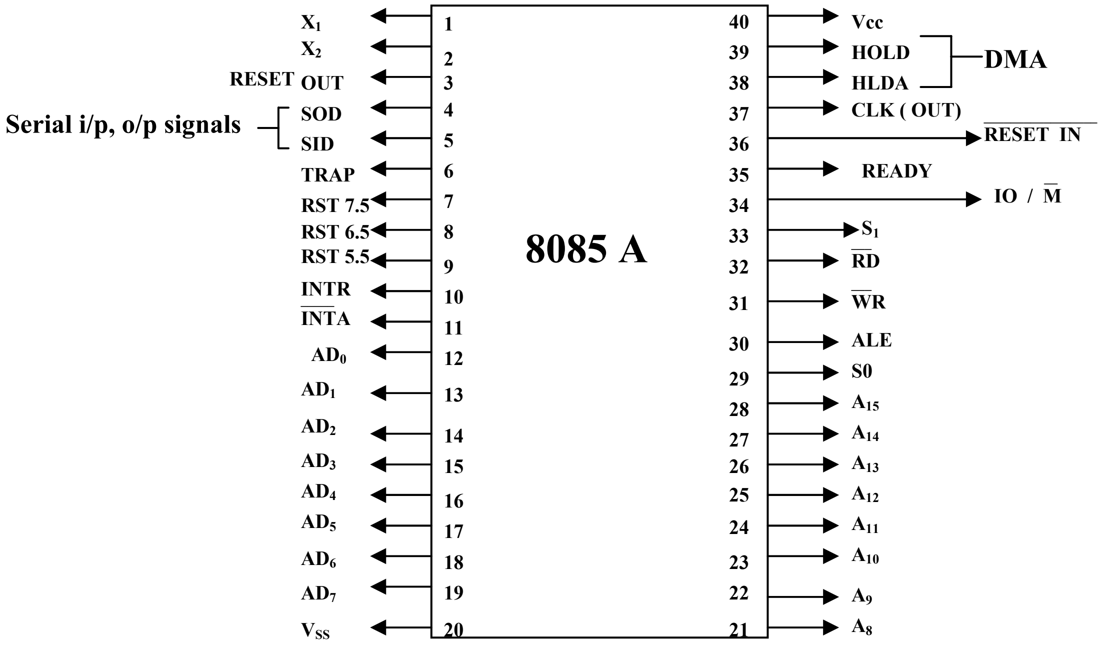
- It is a 8 bit microprocessor.
- It is manufactured with N-MOS technology.
- It has 16-bit address bus and hence can address up to 216 = 65536 bytes (64KB) memory locations through A_0-A_{15}.
- The first 8 lines of address bus and 8 lines of data bus are multiplexed $AD_0 – AD_7$.
- Data bus is a group of 8 lines $D_0 – D_7$.
- It supports external interrupt request.
- A 16 bit program counter (PC)
- A 16 bit stack pointer (SP)
- Six 8-bit general purpose register arranged in pairs: BC, DE, HL.
- It requires a signal +5V power supply and operates at 3.2 MHZ single phase clock.
- It is enclosed with 40 pins DIP (Dual in line package).

Overview of 8085 microprocessor

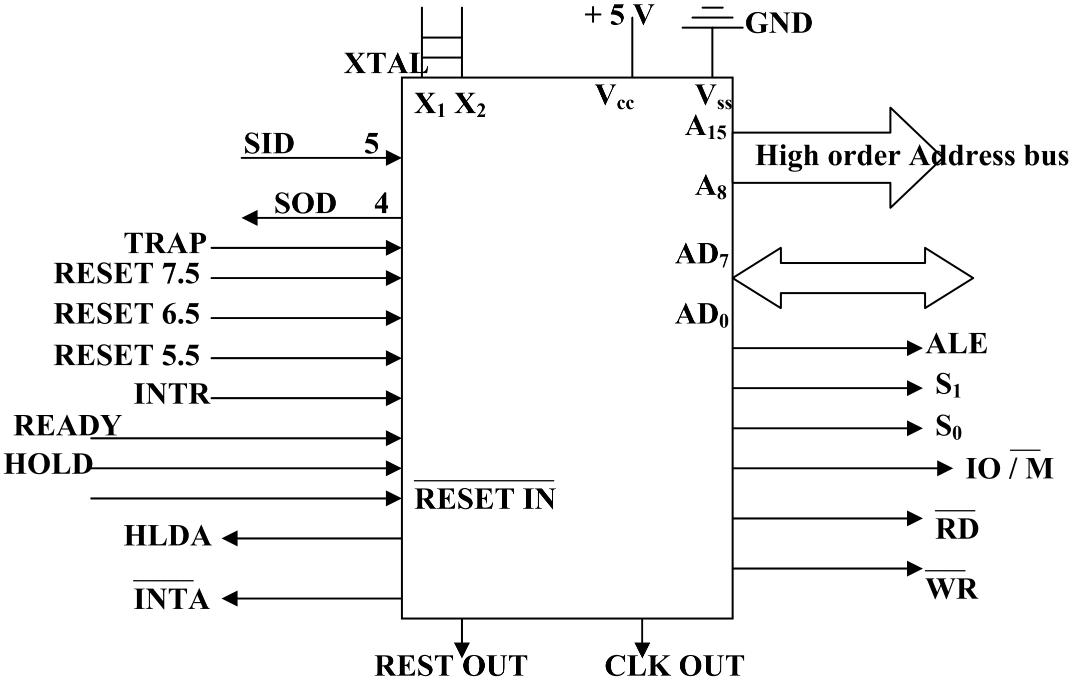
➢ 8085 Architecture

- Pin Diagram

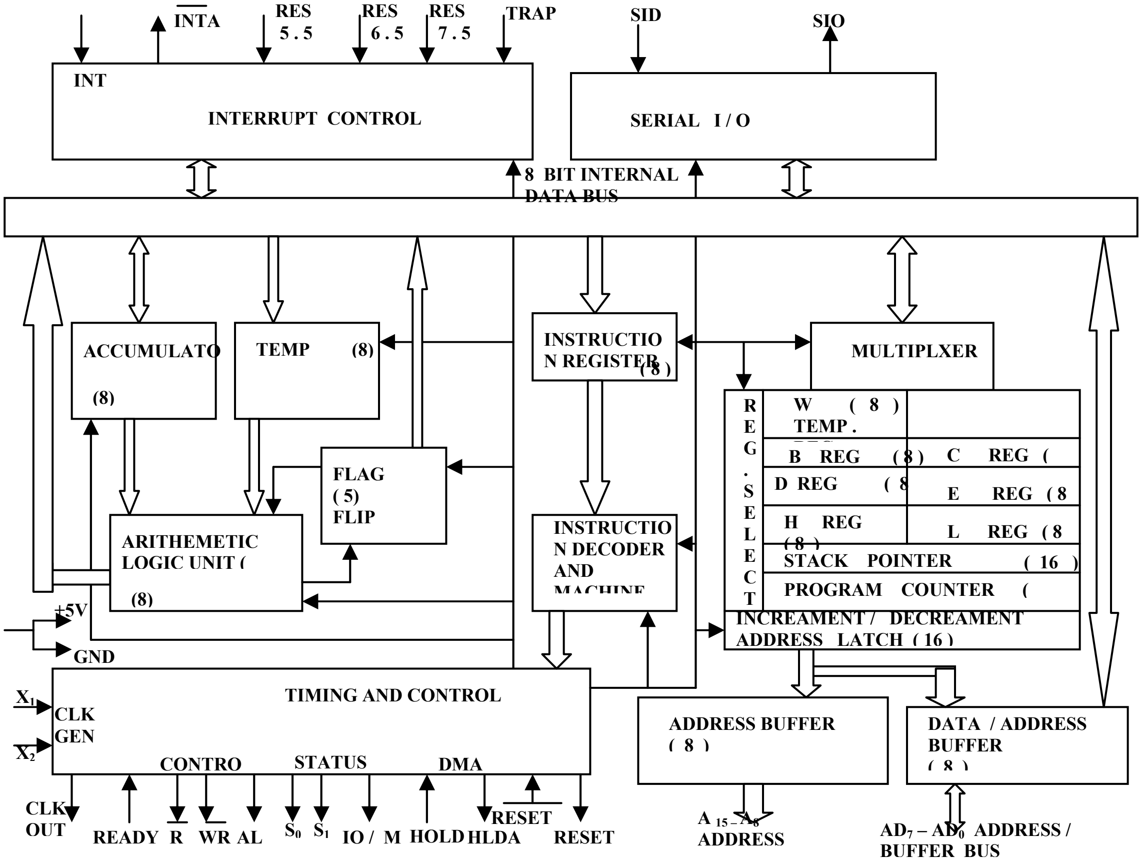
- Functional Block Diagram



Pin Diagram of 8085



Signal Groups of 8085



Block Diagram

Flag Registers

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z		AC		P		CY

General Purpose Registers

INDIVIDUAL	**B, C, D, E, H, L**
COMBININATON	**B & C, D & E, H & L**

Memory

- Program, data and stack memories occupy the same memory space. The total addressable memory size is 64 KB.
- **Program memory** - program can be located anywhere in memory. Jump, branch and call instructions use 16-bit addresses, i.e. they can be used to jump/branch anywhere within 64 KB. All jump/branch instructions use absolute addressing.
- **Data memory** - the processor always uses 16-bit addresses so that data can be placed anywhere.
- **Stack memory** is limited only by the size of memory. Stack grows downward.
- First 64 bytes in a zero memory page should be reserved for vectors used by RST instructions.

Interrupts

- The processor has 5 interrupts. They are presented below in the order of their priority (from lowest to highest):
- **INTR** is maskable 8080A compatible interrupt. When the interrupt occurs the processor fetches from the bus one instruction, usually one of these instructions:
- One of the 8 RST instructions (RST_0 - RST_7). The processor saves current program counter into stack and branches to memory location N * 8 (where N is a 3-bit number from 0 to 7 supplied with the RST instruction).

- **CALL** instruction (3 byte instruction). The processor calls the subroutine, address of which is specified in the second and third bytes of the instruction.
- **RST5.5** is a maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 2CH (hexadecimal) address.
- **RST6.5** is a maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 34H (hexadecimal) address.
- **RST7.5** is a maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 3CH (hexadecimal) address.
- **TRAP** is a non-maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 24H (hexadecimal) address.
- All maskable interrupts can be enabled or disabled using EI and DI instructions. RST 5.5, RST6.5 and RST7.5 interrupts can be enabled or disabled individually using SIM instruction.

Reset Signals

- **RESET IN**: When this signal goes low, the program counter (PC) is set to Zero, μp is reset and resets the interrupt enable and HLDA flip-flops.
- The data and address buses and the control lines are 3-stated during RESET and because of asynchronous nature of RESET, the processor internal registers and flags may be altered by RESET with unpredictable results.
- RESET IN is a Schmitt-triggered input, allowing connection to an R-C network for power-on RESET delay.
- Upon power-up, RESET IN must remain low for at least 10 ms after minimum Vcc has been reached.
- For proper reset operation after the power – up duration, RESET IN should be kept low a minimum of three clock periods.
- The CPU is held in the reset condition as long as RESET IN is applied. Typical Power-on RESET RC values R1 = 75KΩ, C1 = 1μF.
- **RESET OUT**: This signal indicates that μp is being reset. This signal can be used to reset other devices. The signal is synchronized to the processor clock and lasts an integral number of clock periods.

Serial communication Signal

- **SID - Serial Input Data Line**: The data on this line is loaded into accumulator bit 7 whenever a RIM instruction is executed.
- **SOD – Serial Output Data Line**: The SIM instruction loads the value of bit 7 of the accumulator into SOD latch if bit 6 (SOE) of the accumulator is 1.

DMA Signals

- **HOLD**: Indicates that another master is requesting the use of the address and data buses. The CPU, upon receiving the hold request, will relinquish the use of the bus as soon as the completion of the current bus transfer.
- Internal processing can continue. The processor can regain the bus only after the HOLD is removed.
- When the HOLD is acknowledged, the Address, Data RD, WR and IO/M lines are 3-stated.
- **HLDA: Hold Acknowledge**: Indicates that the CPU has received the HOLD request and that it will relinquish the bus in the next clock cycle.
- HLDA goes low after the Hold request is removed. The CPU takes the bus one half-clock cycle after HLDA goes low.
- **READY:** This signal Synchronizes the fast CPU and the slow memory, peripherals.
- If READY is high during a read or write cycle, it indicates that the memory or peripheral is ready to send or receive data.
- If READY is low, the CPU will wait an integral number of clock cycle for READY to go high before completing the read or write cycle.
- READY must conform to specified setup and hold times.

Registers

- **Accumulator** or A register is an 8-bit register used for arithmetic, logic, I/O and load/store operations.
- **Flag Register** has five 1-bit flags.
- **Sign** - set if the most significant bit of the result is set.
- **Zero** - set if the result is zero.
- **Auxiliary carry** - set if there was a carry out from bit 3 to bit 4 of the result.
- **Parity** - set if the parity (the number of set bits in the result) is even.
- **Carry** - set if there was a carry during addition, or borrow during subtraction/comparison/rotation.

General Registers

- 8-bit B and 8-bit C registers can be used as one 16-bit BC register pair. When used as a pair the C register contains low-order byte. Some instructions may use BC register as a data pointer.
- 8-bit D and 8-bit E registers can be used as one 16-bit DE register pair. When used as a pair the E register contains low-order byte. Some instructions may use DE register as a data pointer.
- 8-bit H and 8-bit L registers can be used as one 16-bit HL register pair. When used as a pair the L register contains low-order byte. HL register usually contains a data pointer used to reference memory addresses.

- **Stack pointer** is a 16 bit register. This register is always decremented/incremented by 2 during push and pop.
- **Program counter** is a 16-bit register.

Instruction Set

- 8085 instruction set consists of the following instructions:
- Data moving instructions.
- Arithmetic - add, subtract, increment and decrement.
- Logic - AND, OR, XOR and rotate.
- Control transfer - conditional, unconditional, call subroutine, return from subroutine and restarts.
- Input/Output instructions.
- Other - setting/clearing flag bits, enabling/disabling interrupts, stack operations, etc.

Addressing mode

- **Register** - references the data in a register or in a register pair.
  Register indirect - instruction specifies register pair containing address, where the data is located.
  Direct, Immediate - 8 or 16-bit data.

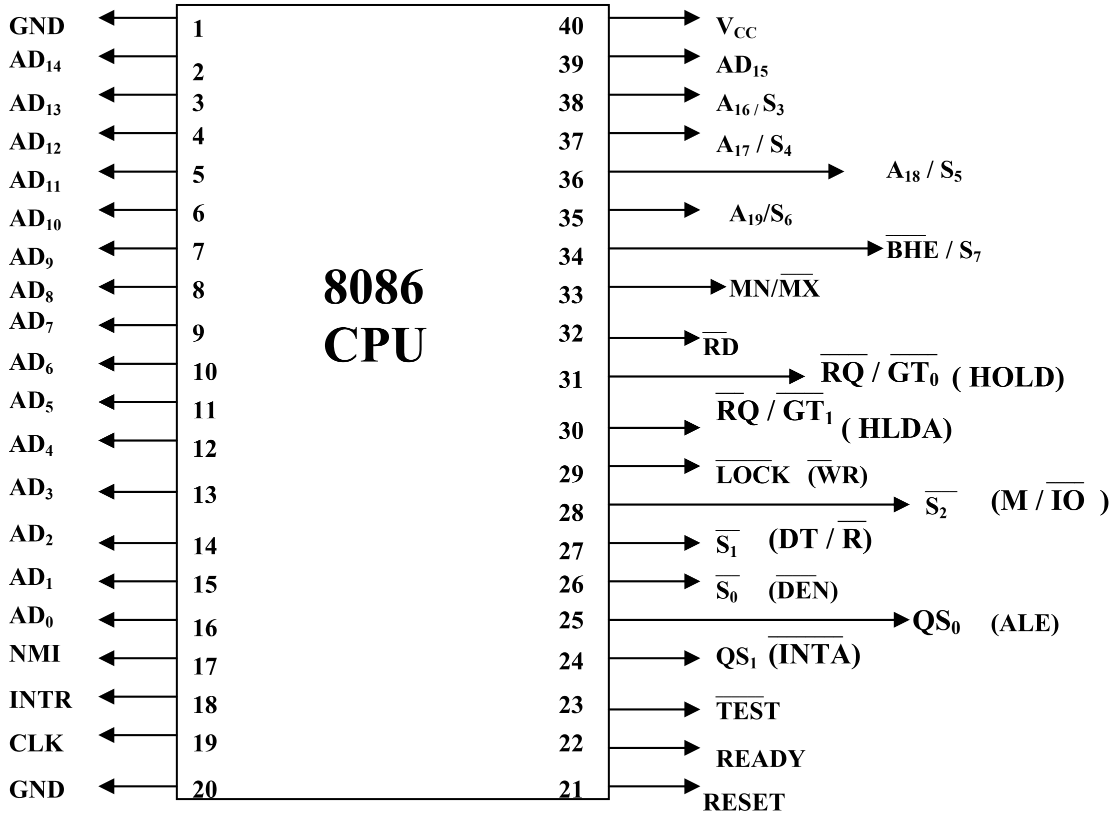
8086 Microprocessor

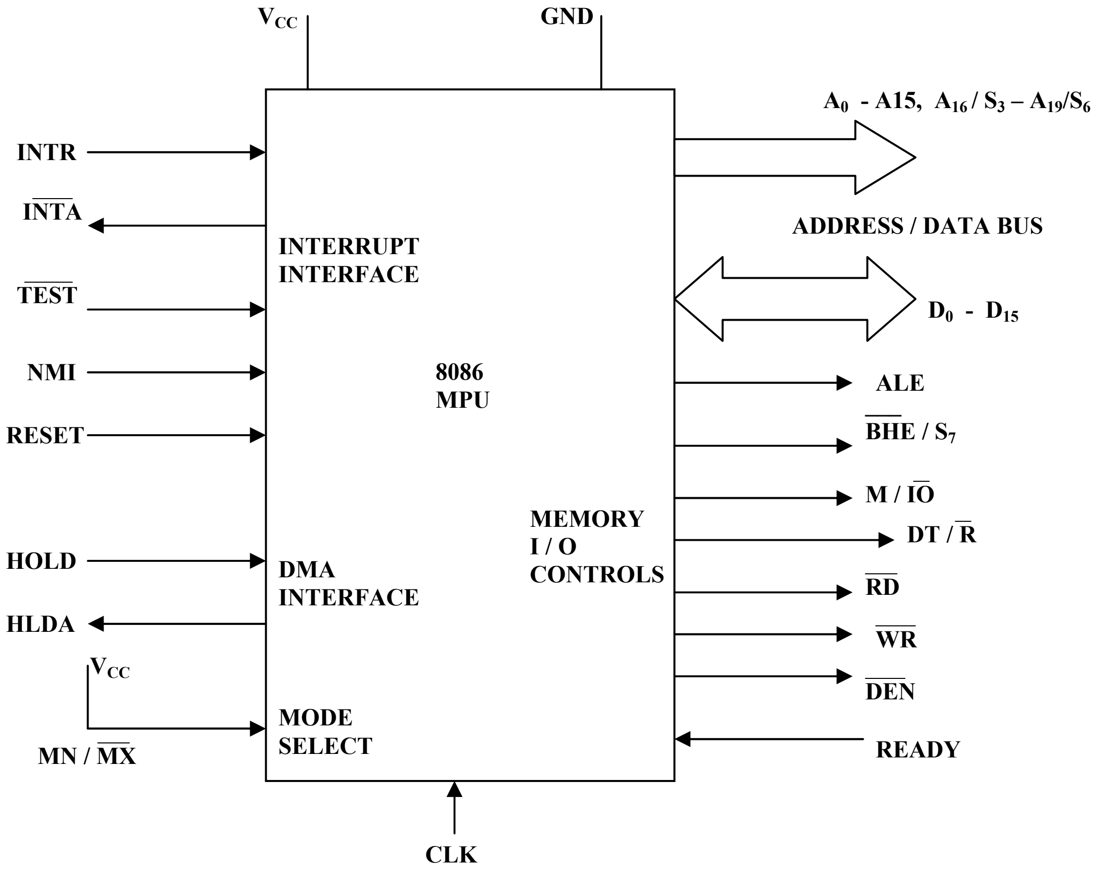
•It is a 16-bit µp.
•8086 has a 20 bit address bus can access up to 2^{20} memory locations (1 MB) .
•It can support up to 64K I/O ports.
•It provides 14, 16 -bit registers.
•It has multiplexed address and data bus AD_0- AD_{15} and A_{16} – A_{19}.
•It requires single phase clock with 33% duty cycle to provide internal timing.
•8086 is designed to operate in two modes, Minimum and Maximum.
•It can prefetches upto 6 instruction bytes from memory and queues them in order to speed up instruction execution.
•It requires +5V power supply.
•A 40 pin dual in line package

Minimum and Maximum Modes:

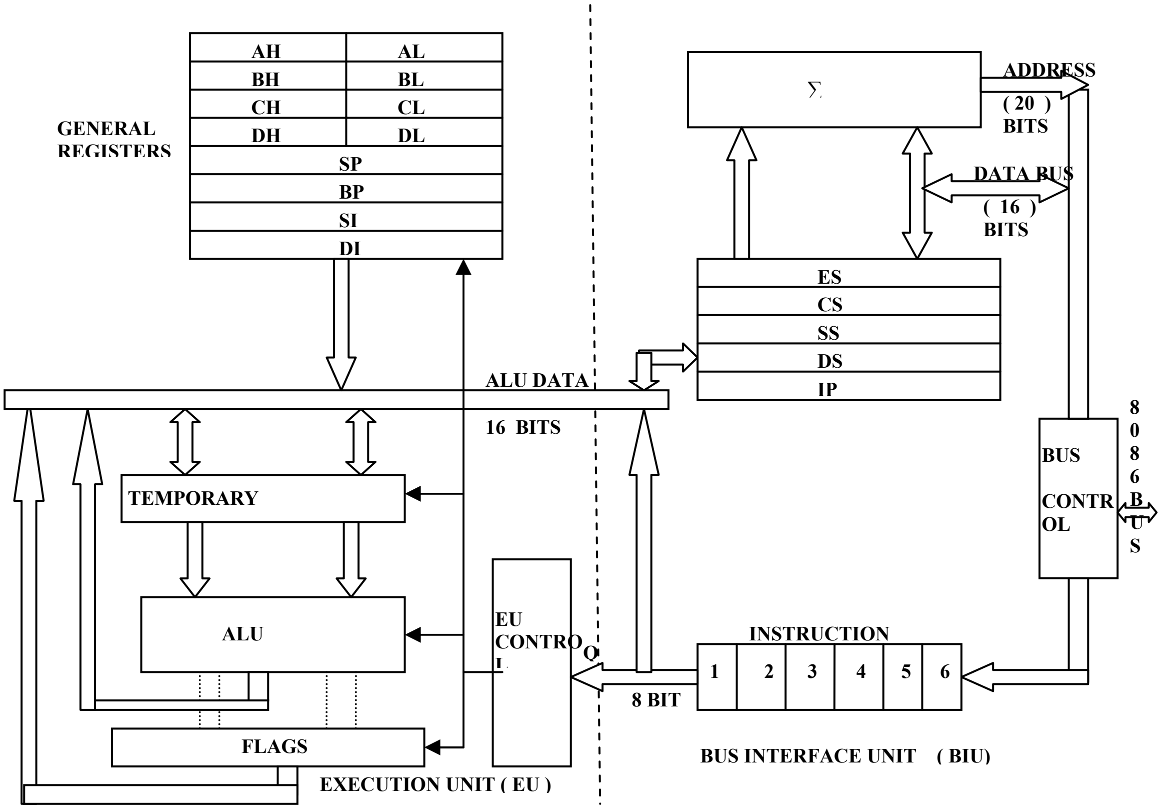
•The minimum mode is selected by applying logic 1 to the MN / MX# input pin. This is a single microprocessor configuration.
• The maximum mode is selected by applying logic 0 to the MN / MX# input pin. This is a multi micro processors configuration.

Pin Diagram of 8086





Signal Groups of 8086



Block Diagram of 8086

Internal Architecture of 8086

•8086 has two blocks BIU and EU.
•The BIU performs all bus operations such as instruction fetching, reading and writing operands for memory and calculating the addresses of the memory operands. The instruction bytes are transferred to the instruction queue.
•EU executes instructions from the instruction system byte queue.
•Both units operate asynchronously to give the 8086 an overlapping instruction fetch and execution mechanism which is called as Pipelining. This results in efficient use of the system bus and system performance.
•BIU contains Instruction queue, Segment registers, Instruction pointer, Address adder.
•EU contains Control circuitry, Instruction decoder, ALU, Pointer and Index register, Flag register.

•BUS INTERFACR UNIT:

• It provides a full 16 bit bidirectional data bus and 20 bit address bus.
•The bus interface unit is responsible for performing all external bus operations.
Specifically it has the following functions:

•Instruction fetch, Instruction queuing, Operand fetch and storage, Address relocation and Bus control.
•The BIU uses a mechanism known as an instruction stream queue to implement a ***pipeline architecture.***
•This queue permits prefetch of up to six bytes of instruction code. When ever the queue of the BIU is not full, it has room for at least two more bytes and at the same time the EU is not requesting it to read or write operands from memory, the BIU is free to look ahead in the program by prefetching the next sequential instruction.
•These prefetching instructions are held in its FIFO queue. With its 16 bit data bus, the BIU fetches two instruction bytes in a single memory cycle.
•After a byte is loaded at the input end of the queue, it automatically shifts up through the FIFO to the empty location nearest the output.
•The EU accesses the queue from the output end. It reads one instruction byte after the other from the output of the queue. If the queue is full and the EU is not requesting access to operand in memory.
•These intervals of no bus activity, which may occur between bus cycles are known as ***Idle state***.
•If the BIU is already in the process of fetching an instruction when the EU request it to read or write operands from memory or I/O, the BIU first completes the instruction fetch bus cycle before initiating the operand read / write cycle.
•The BIU also contains a dedicated adder which is used to generate the 20bit physical address that is output on the address bus. This address is formed by adding an appended 16 bit segment address and a 16 bit offset address.
•For example**:** The physical address of the next instruction to be fetched is formed by combining the current contents of the code segment CS register and the current contents of the instruction pointer IP register.
•The BIU is also responsible for generating bus control signals such as those for memory read or write and I/O read or write.

•EXECUTION UNIT

The Execution unit is responsible for decoding and executing all instructions.
•The EU extracts instructions from the top of the queue in the BIU, decodes them, generates operands if necessary, passes them to the BIU and requests it to perform the read or write bys cycles to memory or I/O and perform the operation specified by the instruction on the operands.
•During the execution of the instruction, the EU tests the status and control flags and updates them based on the results of executing the instruction.
•If the queue is empty, the EU waits for the next instruction byte to be fetched and shifted to top of the queue.
•When the EU executes a branch or jump instruction, it transfers control to a location corresponding to another set of sequential instructions.
•Whenever this happens, the BIU automatically resets the queue and then begins to fetch instructions from this new location to refill the queue.

Signal Description of 8086

- The Microprocessor 8086 is a 16-bit CPU available in different clock rates and packaged in a 40 pin CERDIP or plastic package.
- The 8086 operates in single processor or multiprocessor configuration to achieve high performance. The pins serve a particular function in minimum mode (single processor mode) and other function in maximum mode configuration (multiprocessor mode).
- The 8086 signals can be categorised in three groups. The first are the signal having common functions in minimum as well as maximum mode.
- The second are the signals which have special functions for minimum mode and third are the signals having special functions for maximum mode.
- The following signal descriptions are common for both modes.
- **AD_{15}-AD_0** : These are the time multiplexed memory I/O address and data lines.
- Address remains on the lines during T_1 state, while the data is available on the data bus during T_2, T_3, T_w and T_4.
- These lines are active high and float to a tristate during interrupt acknowledge and local bus hold acknowledge cycles.
- **A_{19}/S_6,A_{18}/S_5,A_{17}/S_4,A_{16}/S_3** : These are the time multiplexed address and status lines.
- During T_1 these are the most significant address lines for memory operations.
- During I/O operations, these lines are low. During memory or I/O operations, status information is available on those lines for T_2,T_3,T_w and T_4.
- The status of the interrupt enable flag bit is updated at the beginning of each clock cycle.
- The S_4 and S_3 combinedly indicate which segment register is presently being used for memory accesses as in below fig.
- These lines float to tri-state off during the local bus hold acknowledge. The status line S_6 is always low .
- The address bit are separated from the status bit using latches controlled by the ALE signal.

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

•**BHE/S$_7$** : The bus high enable is used to indicate the transfer of data over the higher order (D$_{15}$-D$_8$) data bus as shown in table. It goes low for the data transfer over D$_{15}$-D$_8$ and is used to derive chip selects of odd address memory bank or peripherals. BHE is low during T$_1$ for read, write and interrupt acknowledge cycles, whenever a byte is to be transferred on higher byte of data bus. The status information is available during T$_2$, T$_3$ and T$_4$. The signal is active low and tristated during hold. It is low during T$_1$ for the first pulse of the interrupt acknowledge cycle.

0	**0**	**Whole word**
0	**1**	**Upper byte from or to even address**
1	**0**	**Lower byte from or to even address**

•**RD** – **Read** : This signal on low indicates the peripheral that the processor is performing s memory or I/O read operation. RD is active low and shows the state for T$_2$, T$_3$, T$_w$ of any read cycle. The signal remains tristated during the hold acknowledge.
•**READY** : This is the acknowledgement from the slow device or memory that they have completed the data transfer. The signal made available by the devices is synchronized by the 8284A clock generator to provide ready input to the 8086. the signal is active high.

•**INTR-Interrupt Request** : This is a triggered input. This is sampled during the last clock cycles of each instruction to determine the availability of the request. If any interrupt request is pending, the processor enters the interrupt acknowledge cycle.
•This can be internally masked by resulting the interrupt enable flag. This signal is active high and internally synchronized.
•**TEST** : This input is examined by a ‘WAIT’ instruction. If the TEST pin goes low, execution will continue, else the processor remains in an idle state. The input is synchronized internally during each clock cycle on leading edge of clock.

•**CLK-** Clock Input : The clock input provides the basic timing for processor operation and bus control activity. Its an asymmetric square wave with 33% duty cycle.
•**MN/MX** : The logic level at this pin decides whether the processor is to operate in either minimum or maximum mode.

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

•**M/IO – Memory/IO** : This is a status line logically equivalent to S$_2$ in maximum mode. When it is low, it indicates the CPU is having an I/O operation, and when it is high, it indicates that the CPU is having a memory operation. This line becomes active high in the previous T_4 and remains active till final T_4 of the current cycle. It is tristated during local bus “hold acknowledge “.

•**INTA – Interrupt Acknowledge** : This signal is used as a read strobe for interrupt acknowledge cycles. i.e. when it goes low, the processor has accepted the interrupt.
•**ALE – Address Latch Enable** : This output signal indicates the availability of the valid address on the address/data lines, and is connected to latch enable input of latches. This signal is active high and is never tristated.
•**DT/R – Data Transmit/Receive**: This output is used to decide the direction of data flow through the transreceivers (bidirectional buffers). When the processor sends out data, this signal is high and when the processor is receiving data, this signal is low.

•**DEN – Data Enable** : This signal indicates the availability of valid data over the address/data lines. It is used to enable the transreceivers (bidirectional buffers) to separate the data from the multiplexed address/data signal. It is active from the middle of T_2 until the middle of T_4. This is tristated during ‘ hold acknowledge’ cycle.
•**HOLD, HLDA- 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. HOLD is an asynchronous input, and is should be externally synchronized.
•If the DMA request is made while the CPU is performing a memory or I/O cycle, it will release the local bus during T_4 provided :
1.The request occurs on or before T_2 state of the current cycle.
2.The current cycle is not operating over the lower byte of a word.
3.The current cycle is not the first acknowledge of an interrupt acknowledge sequence.

4. A Lock instruction is not being executed.

•***The following pin function are applicable for maximum mode operation of 8086***.

•**S_2, S_1, S_0 – Status Lines** : These are the status lines which reflect the type of operation, being carried out by the processor. These become activity during T_4 of the previous cycle and active during T_1 and T_2 of the current bus cycles.

S_2	S_1	S_0	Indication
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1	0	0	Code Access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive

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

•By prefetching the instruction, there is a considerable speeding up in instruction execution in 8086. This is known as ***instruction pipelining***.
•At the starting the CS:IP is loaded with the required address from which the execution is to be started. Initially, the queue will be empty an the microprocessor starts a fetch operation to bring one byte (the first byte) of instruction code, if the CS:IP address is odd or two bytes at a time, if the CS:IP address is even.
•The first byte is a complete opcode in case of some instruction (one byte opcode instruction) and is a part of opcode, in case of some instructions (two byte opcode instructions), the remaining part of code lie in second byte.
•The second byte is then decoded in continuation with the first byte to decide the instruction length and the number of subsequent bytes to be treated as instruction data.
•The queue is updated after every byte is read from the queue but the fetch cycle is initiated by BIU only if at least two bytes of the queue are empty and the EU may be concurrently executing the fetched instructions.
•The next byte after the instruction is completed is again the first opcode byte of the next instruction. A similar procedure is repeated till the complete execution of the program.

•The fetch operation of the next instruction is overlapped with the execution of the current instruction. As in the architecture, there are two separate units, namely Execution unit and Bus interface unit.
•While the execution unit is busy in executing an instruction, after it is completely decoded, the bus interface unit may be fetching the bytes of the next instruction from memory, depending upon the queue status.

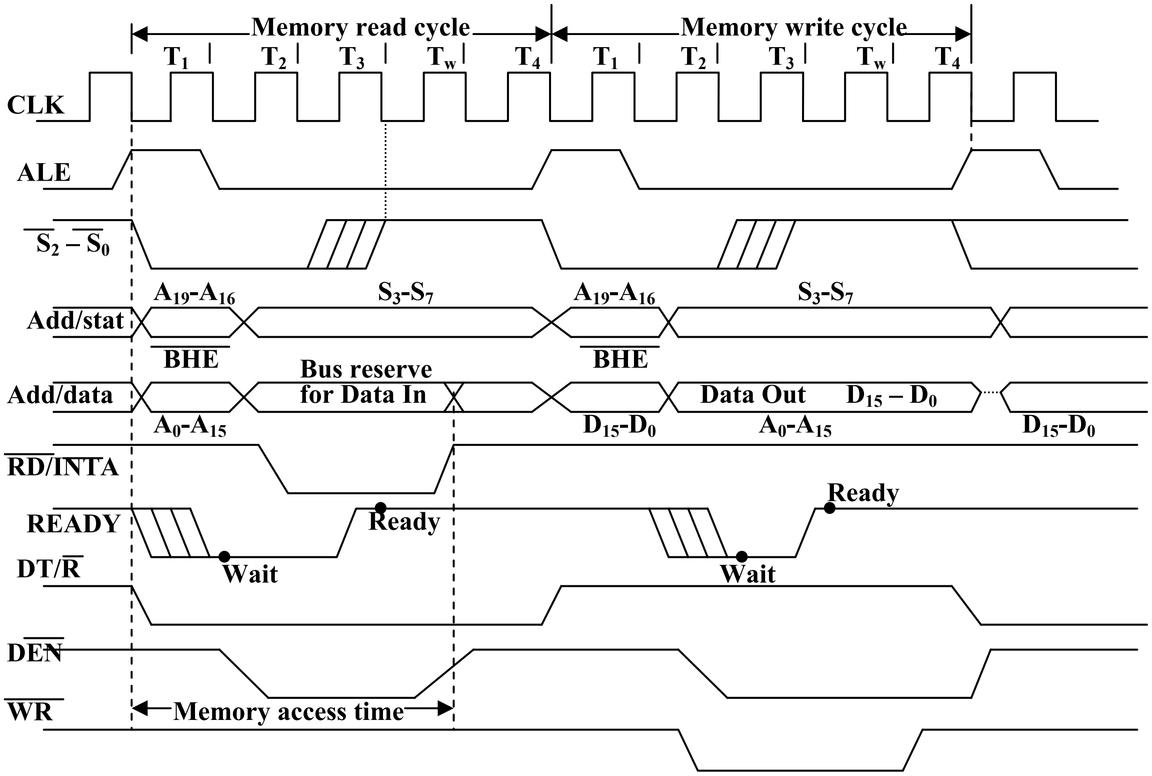
QS_1	QS_0	Indication
0	0	**No operation**
0	1	**First byte of the opcode from the queue**
1	0	**Empty queue**
1	1	**Subsequent byte from the queue**

•**RQ/GT$_0$, RQ/GT$_1$ – Request/Grant :** These pins are used by the other local bus master in maximum mode, to force the processor to release the local bus at the end of the processor current bus cycle.
•Each of the pin is bidirectional with RQ/GT$_0$ having higher priority than RQ/GT$_1$.
•RQ/GT pins have internal pull-up resistors and may be left unconnected.
•Request/Grant sequence is as follows:
1.A pulse of one clock wide from another bus master requests the bus access to 8086.
2.During T$_4$(current) or T$_1$(next) clock cycle, a pulse one clock wide from 8086 to the requesting master, indicates that the 8086 has allowed the local bus to float and that it will enter the 'hold acknowledge' state at next cycle. The CPU bus interface unit is likely to be disconnected from the local bus of the system.
3.A one clock wide pulse from the another master indicates to the 8086 that the hold request is about to end and the 8086 may regain control of the local bus at the next clock cycle. Thus each master to master exchange of the local bus is a sequence of 3 pulses. There must be at least one dead clock cycle after each bus exchange.
•The request and grant pulses are active low.
•For the bus request those are received while 8086 is performing memory or I/O cycle, the granting of the bus is governed by the rules as in case of HOLD and HLDA in minimum mode.

General Bus Operation

•The 8086 has a combined address and data bus commonly referred as a time multiplexed address and data bus.

• The main reason behind multiplexing address and data over the same pins is the maximum utilisation of processor pins and it facilitates the use of 40 pin standard DIP package.
•The bus can be demultiplexed using a few latches and transreceivers, when ever required.
•Basically, all the processor bus cycles consist of at least four clock cycles. These are referred to as T_1, T_2, T_3, T_4. The address is transmitted by the processor during T_1. It is present on the bus only for one cycle.
•The negative edge of this ALE pulse is used to separate the address and the data or status information. In maximum mode, the status lines S_0, S_1 and S_2 are used to indicate the type of operation.
•Status bits S_3 to S_7 are multiplexed with higher order address bits and the BHE signal. Address is valid during T_1 while status bits S_3 to S_7 are valid during T_2 through T_4.

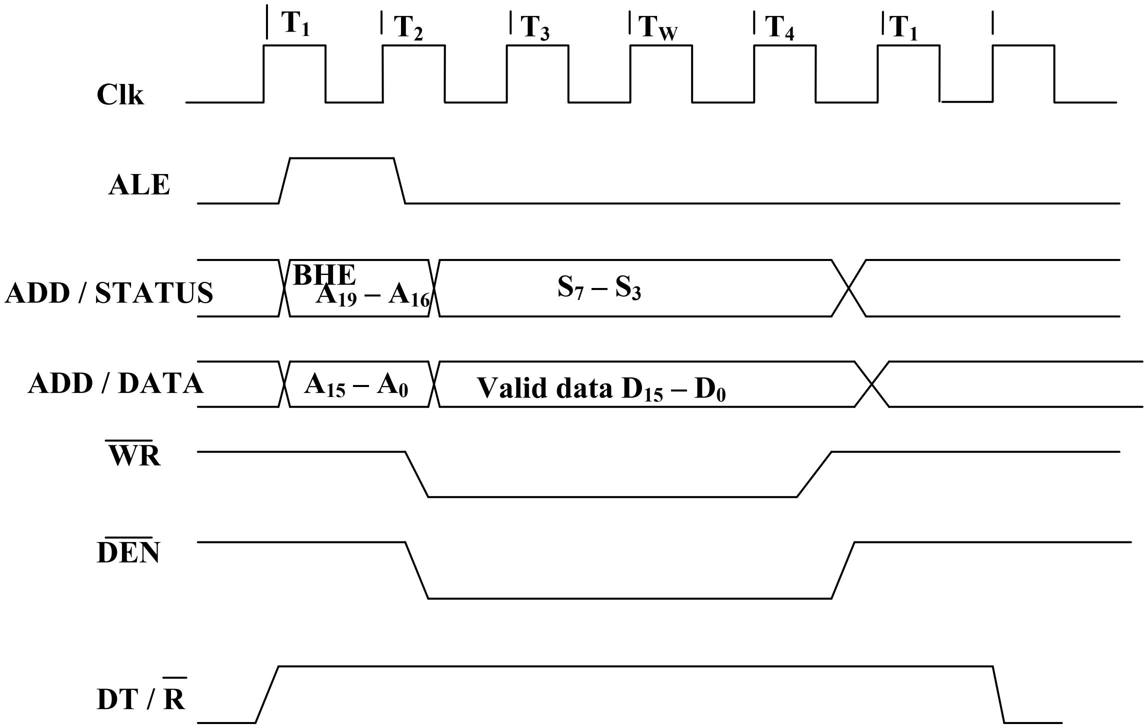


General Bus Operation Cycle in Maximum Mode

Minimum Mode 8086 System

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

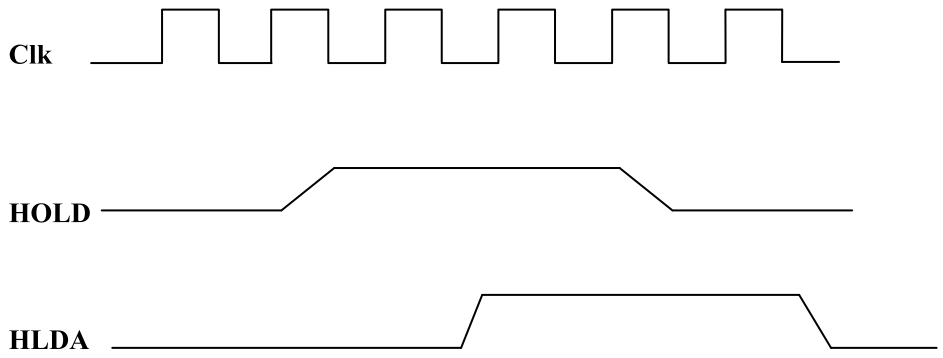
•The remaining components in the system are latches, transreceivers, clock generator, memory and I/O devices. Some type of chip selection logic may be required for selecting memory or I/O devices, depending upon the address map of the system.
•Latches are generally buffered output D-type flip-flops like 74LS373 or 8282. They are used for separating the valid address from the multiplexed address/data signals and are controlled by the ALE signal generated by 8086.
•Transreceivers are the bidirectional buffers and some times they are called as data amplifiers. They are required to separate the valid data from the time multiplexed address/data signals.
•They are controlled by two signals namely, DEN and DT/R.
•The DEN signal indicates the direction of data, i.e. from or to the processor. The system contains memory for the monitor and users program storage.
•Usually, EPROM are used for monitor storage, while RAM for users program storage. A system may contain I/O devices.
•
•The working of the minimum mode configuration system can be better described in terms of the timing diagrams rather than qualitatively describing the operations.
•The opcode fetch and read cycles are similar. Hence the timing diagram can be categorized in two parts, the first is the timing diagram for read cycle and the second is the timing diagram for write cycle.
•The read cycle begins in T_1 with the assertion of address latch enable (ALE) signal and also M / IO signal. During the negative going edge of this signal, the valid address is latched on the local bus.
•The BHE and A_0 signals address low, high or both bytes. From T_1 to T_4 , the M/IO signal indicates a memory or I/O operation.
•At T_2, the address is removed from the local bus and is sent to the output. The bus is then tristated. The read (RD) control signal is also activated in T_2.
•The read (RD) signal causes the address device to enable its data bus drivers. After RD goes low, the valid data is available on the data bus.
•The addressed device will drive the READY line high. When the processor returns the read signal to high level, the addressed device will again tristate its bus drivers.
•A write cycle also begins with the assertion of ALE and the emission of the address. The M/IO signal is again asserted to indicate a memory or I/O operation. In T_2, after sending the address in T_1, the processor sends the data to be written to the addressed location.
•The data remains on the bus until middle of T_4 state. The WR becomes active at the beginning of T_2 (unlike RD is somewhat delayed in T_2 to provide time for floating).
•The BHE and A_0 signals are used to select the proper byte or bytes of memory or I/O word to be read or write.
•The M/IO, RD and WR signals indicate the type of data transfer as specified in table below.



Write Cycle Timing Diagram for Minimum Mode

•***Hold Response sequence***: The HOLD pin is checked at leading edge of each clock pulse. If it is received active by the processor before T_4 of the previous cycle or during T_1 state of the current cycle, the CPU activates HLDA in the next clock cycle and for succeeding bus cycles, the bus will be given to another requesting master.

•The control of the bus is not regained by the processor until the requesting master does not drop the HOLD pin low. When the request is dropped by the requesting master, the HLDA is dropped by the processor at the trailing edge of the next clock.



Bus Request and Bus Grant Timings in Minimum Mode System

Maximum Mode 8086 System

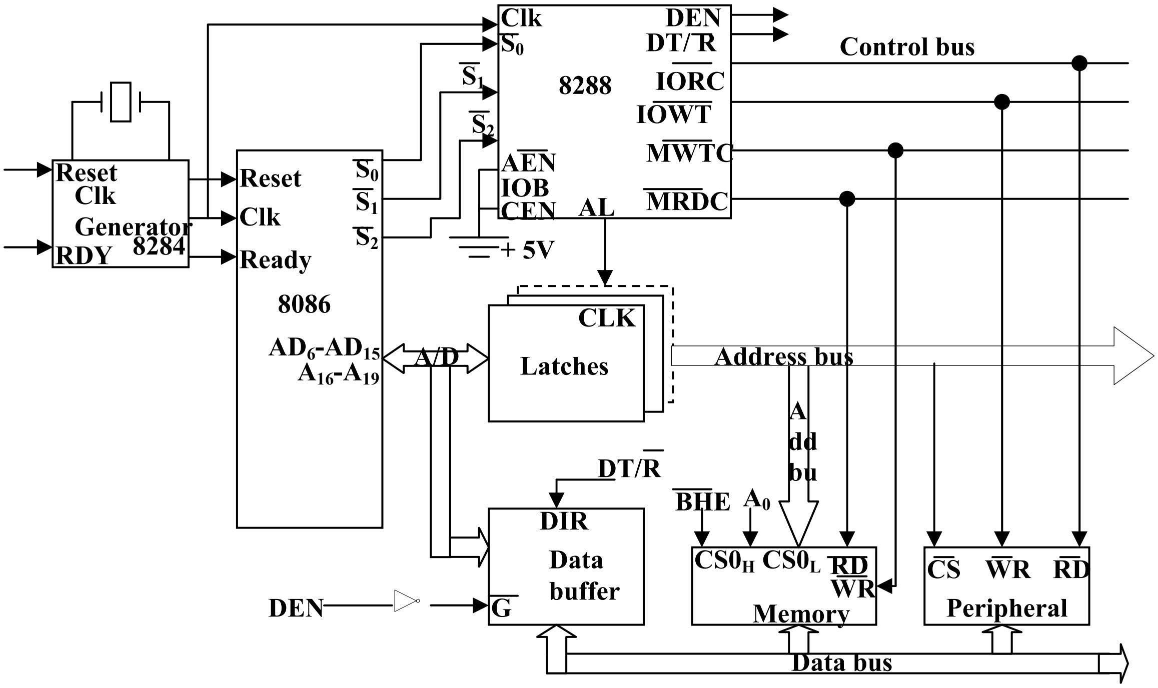
•In the maximum mode, the 8086 is operated by strapping the MN/MX pin to ground.
•In this mode, the processor derives the status signal S_2, S_1, S_0. Another chip called bus controller derives the control signal using this status information .
•In the maximum mode, there may be more than one microprocessor in the system configuration.
•The components in the system are same as in the minimum mode system.
•The basic function of the bus controller chip IC8288, is to derive control signals like RD and WR (for memory and I/O devices), DEN, DT/R, ALE etc. using the information by the processor on the status lines.
•The bus controller chip has input lines S_2, S_1, S_0 and CLK. These inputs to 8288 are driven by CPU.
•It derives the outputs ALE, DEN, DT/R, MRDC, MWTC, AMWC, IORC, IOWC and AIOWC. The AEN, IOB and CEN pins are specially useful for multiprocessor systems.
•
•AEN and IOB are generally grounded. CEN pin is usually tied to +5V. The significance of the MCE/PDEN output depends upon the status of the IOB pin.
•If IOB is grounded, it acts as master cascade enable to control cascade 8259A, else it acts as peripheral data enable used in the multiple bus configurations.
•INTA pin used to issue two interrupt acknowledge pulses to the interrupt controller or to an interrupting device.
•IORC, IOWC are I/O read command and I/O write command signals respectively . These signals enable an IO interface to read or write the data from or to the address port.

•The MRDC, MWTC are memory read command and memory write command signals respectively and may be used as memory read or write signals.

•All these command signals instructs the memory to accept or send data from or to the bus.

•For both of these write command signals, the advanced signals namely AIOWC and AMWTC are available.

•Here the only difference between in timing diagram between minimum mode and maximum mode is the status signals used and the available control and advanced command signals.



Maximum Mode 8086 System.

•R_0, S_1, S_2 are set at the beginning of bus cycle.8288 bus controller will output a pulse as on the ALE and apply a required signal to its DT / R pin during T_1.

•In T_2, 8288 will set DEN=1 thus enabling transceivers, and for an input it will activate MRDC or IORC. These signals are activated until T_4. For an output, the AMWC or AIOWC is activated from T_2 to T_4 and MWTC or IOWC is activated from T_3 to T_4.

•The status bit S_0 to S_2 remains active until T_3 and become passive during T_3 and T_4**.**

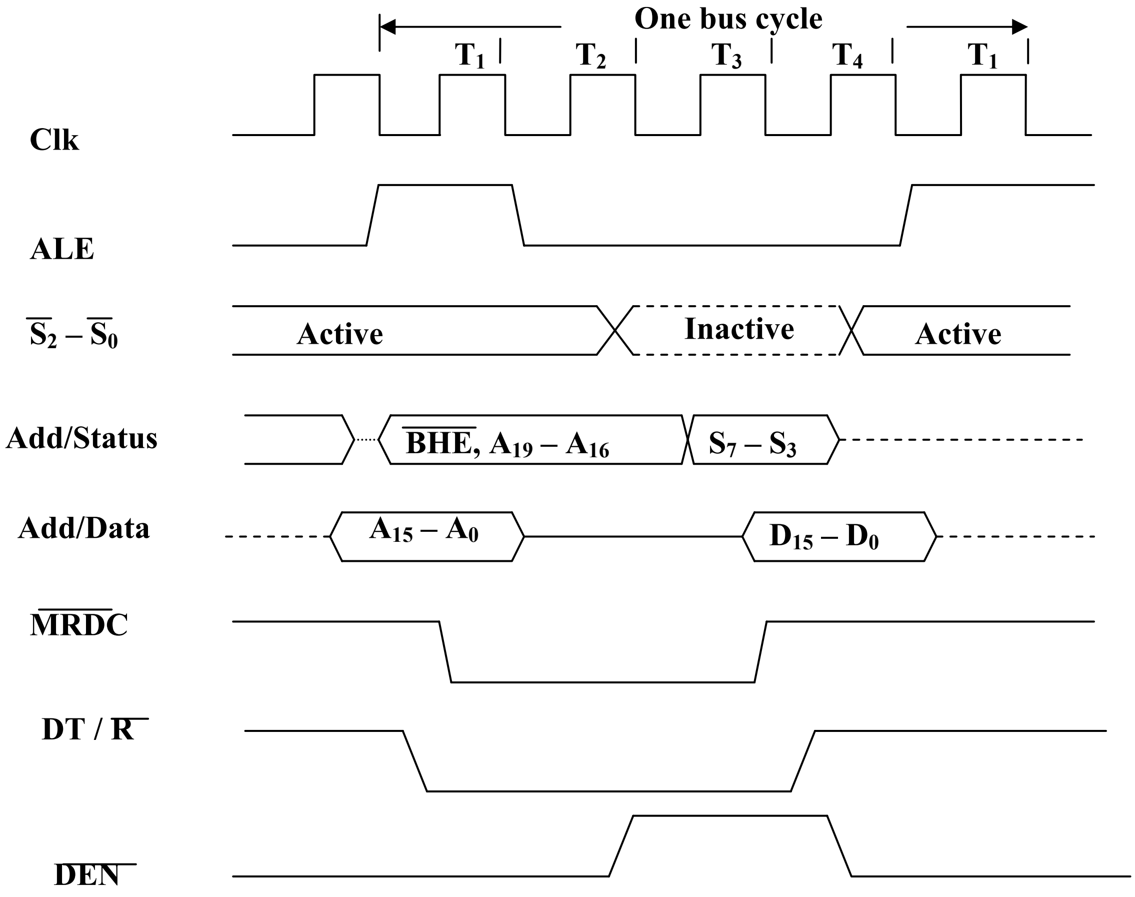
•If reader input is not activated before T_3, wait state will be inserted between T_3 and T_4.

•**Timings for RQ/ GT Signals :**

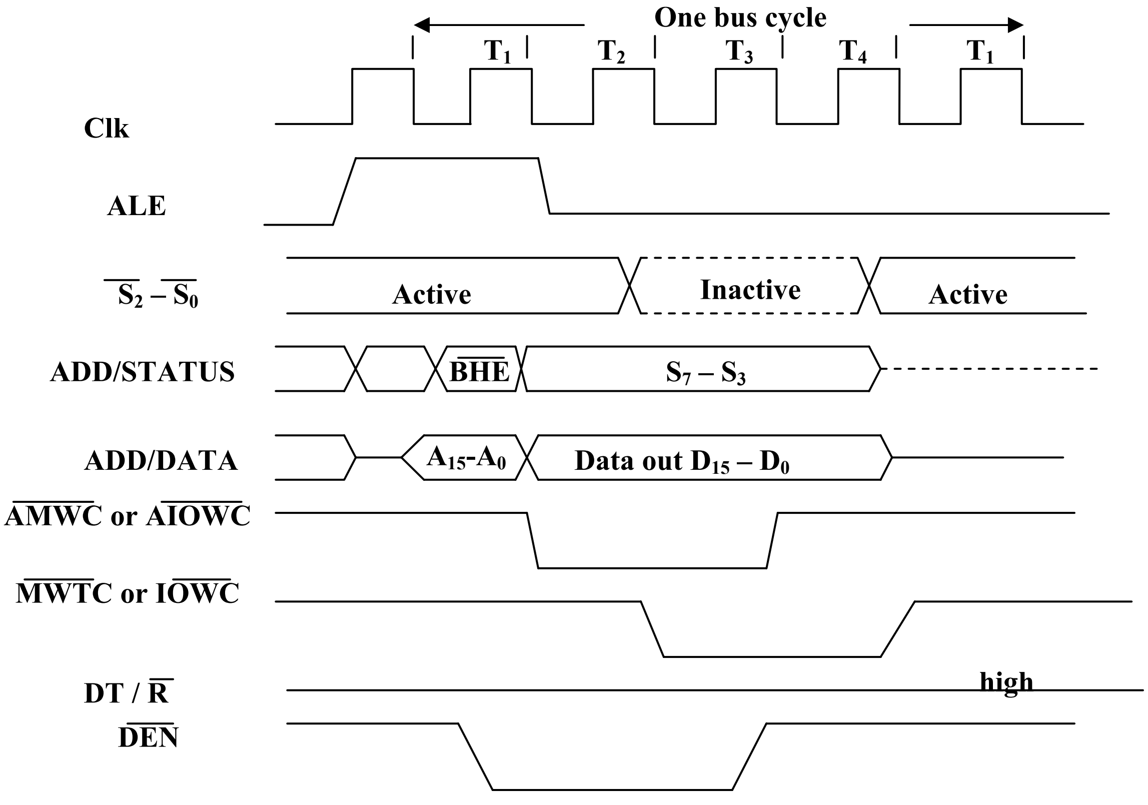
The request/grant response sequence contains a series of three pulses. The request/grant pins are checked at each rising pulse of clock input.

•When a request is detected and if the condition for HOLD request are satisfied, the processor issues a grant pulse over the RQ/GT pin immediately during T_4 (current) or T_1 (next) state.

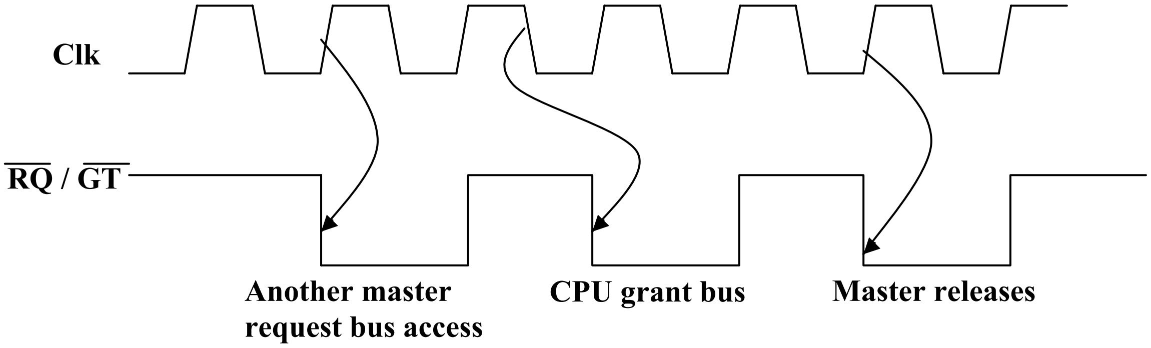
•When the requesting master receives this pulse, it accepts the control of the bus, it sends a release pulse to the processor using RQ/GT pin.



Memory Read Timing in Maximum Mode



Memory Write Timing in Maximum mode.


Clk
$\overline{RQ}$ / $\overline{GT}$
Another master request bus access
CPU grant bus
Master releases

$\overline{RQ}$/$\overline{GT}$ Timings in Maximum Mode.

Minimum Mode Interface

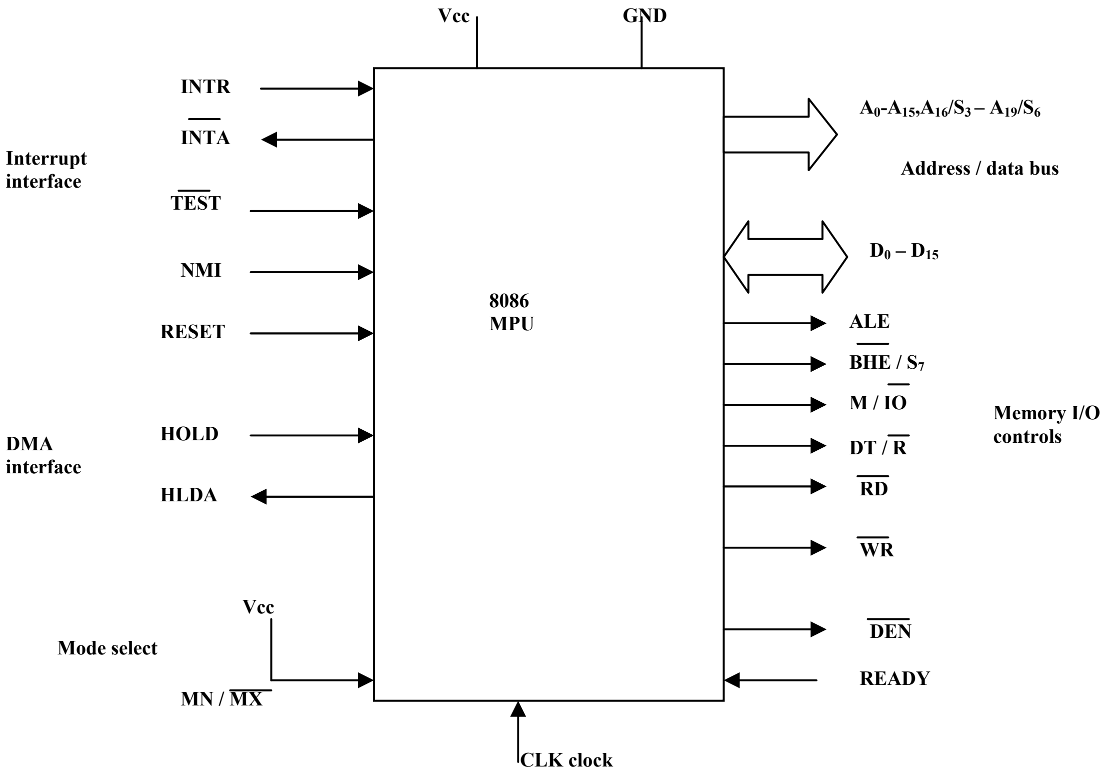
•When the Minimum mode operation is selected, the 8086 provides all control signals needed to implement the memory and I/O interface.

•The minimum mode signal can be divided into the following basic groups : address/data bus, status, control, interrupt and DMA.

•**Address/Data Bus** : these lines serve two functions. As an address bus is 20 bits long and consists of signal lines A_0 through A_{19}. A_{19} represents the MSB and A_0 LSB. A 20bit address gives the 8086 a 1Mbyte memory address space. More over it has an independent I/O address space which is 64K bytes in length.

•The 16 data bus lines D_0 through D_{15} are actually multiplexed with address lines A_0 through A_{15} respectively. By multiplexed we mean that the bus work as an address bus during first machine cycle and as a data bus during next machine cycles. D_{15} is the MSB and D_0 LSB.

•When acting as a data bus, they carry read/write data for memory, input/output data for I/O devices, and interrupt type codes from an interrupt controller.



Block Diagram of the Minimum Mode 8086 MPU

•**Status signal:**

The four most significant address lines A_{19} through A_{16} are also multiplexed but in this case with status signals S_6 through S_3. These status bits are output on the bus at the same time that data are transferred over the other bus lines.
•Bit S_4 and S_3 together from a 2 bit binary code that identifies which of the 8086 internal segment registers are used to generate the physical address that was output on the address bus during the current bus cycle.
•Code S_4S_3 = 00 identifies a register known as ***extra segment register*** as the source of the segment address.
•Status line S_5 reflects the status of another internal characteristic of the 8086. It is the logic level of the internal enable flag. The last status bit S_6 is always at the logic 0 level.

S_4	S_3	**Segment Register**
0	**0**	**Extra**
0	**1**	**Stack**
1	**0**	**Code / none**
1	**1**	**Data**

Memory segment status codes.

•**Control Signals** :

The control signals are provided to support the 8086 memory I/O interfaces. They control functions such as when the bus is to carry a valid address in which direction data are to be transferred over the bus, when valid write data are on the bus and when to put read data on the system bus.

•ALE is a pulse to logic 1 that signals external circuitry when a valid address word is on the bus. This address must be latched in external circuitry on the 1-to-0 edge of the pulse at ALE.

•Another control signal that is produced during the bus cycle is BHE bank high enable. Logic 0 on this used as a memory enable signal for the most significant byte half of the data bus D_8 through D_1. These lines also serves a second function, which is as the S_7 status line.

•Using the M/IO and DT/R lines, the 8086 signals which type of bus cycle is in progress and in which direction data are to be transferred over the bus.

•The logic level of M/IO tells external circuitry whether a memory or I/O transfer is taking place over the bus. Logic 1 at this output signals a memory operation and logic 0 an I/O operation.

•The direction of data transfer over the bus is signaled by the logic level output at DT/R. When this line is logic 1 during the data transfer part of a bus cycle, the bus is in the transmit mode. Therefore, data are either written into memory or output to an I/O device.

•On the other hand, logic 0 at DT/R signals that the bus is in the receive mode. This corresponds to reading data from memory or input of data from an input port.

•The signal read RD and write WR indicates that a read bus cycle or a write bus cycle is in progress. The 8086 switches WR to logic 0 to signal external device that valid write or output data are on the bus.

• On the other hand, RD indicates that the 8086 is performing a read of data of the bus. During read operations, one other control signal is also supplied. This is DEN (data enable) and it signals external devices when they should put data on the bus.

•There is one other control signal that is involved with the memory and I/O interface. This is the READY signal.

•READY signal is used to insert wait states into the bus cycle such that it is extended by a number of clock periods. This signal is provided by an external clock generator device and can be supplied by the memory or I/O sub-system to signal the 8086 when they are ready to permit the data transfer to be completed.

•**Interrupt signals** : The key interrupt interface signals are interrupt request (INTR) and interrupt acknowledge (INTA).

•INTR is an input to the 8086 that can be used by an external device to signal that it need to be serviced.

•Logic 1 at INTR represents an active interrupt request. When an interrupt request has been recognized by the 8086, it indicates this fact to external circuit with pulse to logic 0 at the INTA output.

•The TEST input is also related to the external interrupt interface. Execution of a WAIT instruction causes the 8086 to check the logic level at the TEST input.

•If the logic 1 is found, the MPU suspend operation and goes into the idle state. The 8086 no longer executes instructions, instead it repeatedly checks the logic level of the TEST input waiting for its transition back to logic 0.

•As TEST switches to 0, execution resume with the next instruction in the program. This feature can be used to synchronize the operation of the 8086 to an event in external hardware.

•There are two more inputs in the interrupt interface: the nonmaskable interrupt NMI and the reset interrupt RESET.

•On the 0-to-1 transition of NMI control is passed to a nonmaskable interrupt service routine. The RESET input is used to provide a hardware reset for the 8086. Switching RESET to logic 0 initializes the internal register of the 8086 and initiates a reset service routine.

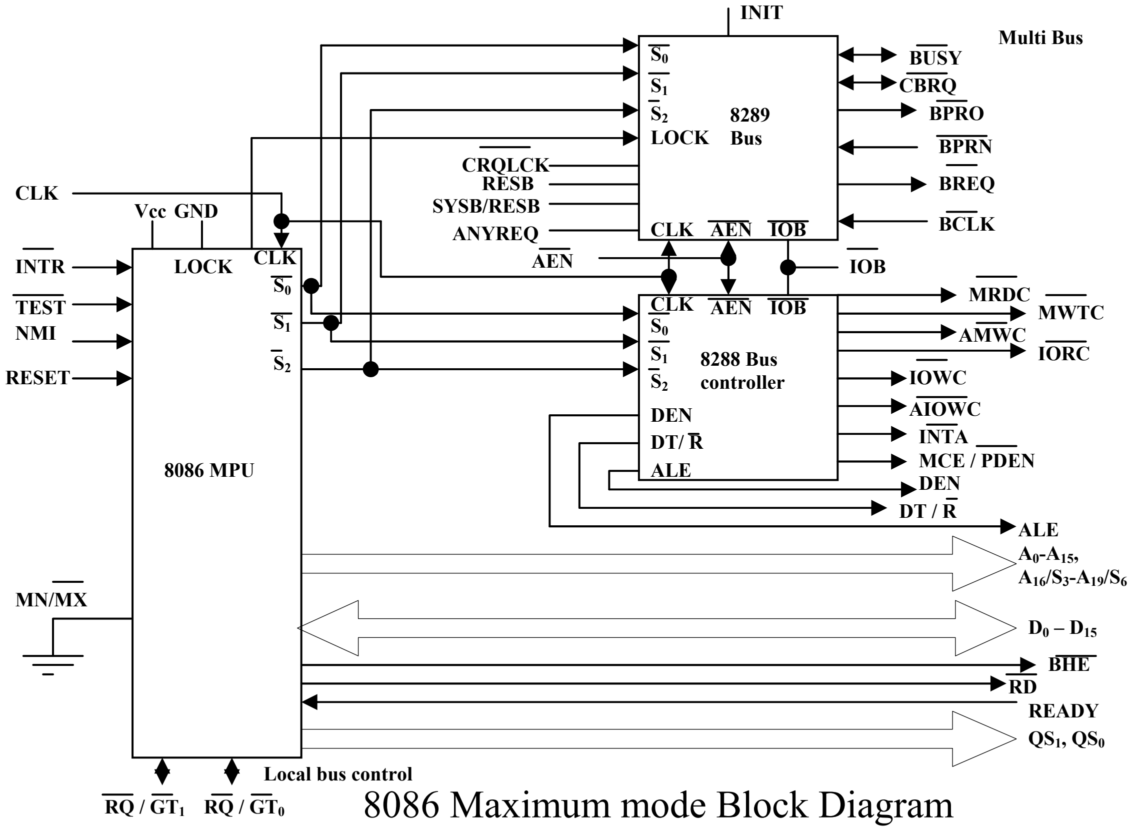
•**DMA Interface signals** :The direct memory access DMA interface of the 8086 minimum mode consist of the HOLD and HLDA signals.

•When an external device wants to take control of the system bus, it signals to the 8086 by switching HOLD to the logic 1 level. At the completion of the current bus cycle, the 8086 enters the hold state. In the hold state, signal lines AD_0 through AD_{15}, A_{16}/S_3

through A_{19}/S_6, BHE, M/IO, DT/R, RD, WR, DEN and INTR are all in the high Z state. The 8086 signals external device that it is in this state by switching its HLDA output to logic 1 level.

Maximum Mode Interface

•When the 8086 is set for the maximum-mode configuration, it provides signals for implementing a multiprocessor / coprocessor system environment.
•By multiprocessor environment we mean that one microprocessor exists in the system and that each processor is executing its own program.
• Usually in this type of system environment, there are some system resources that are common to all processors.
•They are called as ***global resources***. There are also other resources that are assigned to specific processors. These are known as ***local or private resources***.
•Coprocessor also means that there is a second processor in the system. In this two processor does not access the bus at the same time.
•One passes the control of the system bus to the other and then may suspend its operation.
•In the maximum-mode 8086 system, facilities are provided for implementing allocation of global resources and passing bus control to other microprocessor or coprocessor.



8086 Maximum mode Block Diagram

•**8288 Bus Controller – Bus Command and Control Signals**:

8086 does not directly provide all the signals that are required to control the memory, I/O and interrupt interfaces.

•Specially the WR, M/IO, DT/R, DEN, ALE and INTA, signals are no longer produced by the 8086. Instead it outputs three status signals S_0, S_1, S_2 prior to the initiation of each bus cycle. This 3- bit bus status code identifies which type of bus cycle is to follow.

•$S_2S_1S_0$ are input to the external bus controller device, the bus controller generates the appropriately timed command and control signals.

Status Inputs			**CPU Cycles**	**8288 Command**
$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$		
0	**0**	**0**	**Interrupt Acknowledge**	$\overline{\textbf{INTA}}$
0	**0**	**1**	**Read I/O Port**	$\overline{\textbf{IORC}}$
0	**1**	**0**	**Write I/O Port**	$\overline{\textbf{IOWC}}$, $\overline{\textbf{AIOWC}}$
0	**1**	**1**	**Halt**	**None**
1	**0**	**0**	**Instruction Fetch**	$\overline{\textbf{MRDC}}$
1	**0**	**1**	**Read Memory**	$\overline{\textbf{MRDC}}$
1	**1**	**0**	**Write Memory**	$\overline{\textbf{MWTC}}$, $\overline{\textbf{AMWC}}$
1	**1**	**1**	**Passive**	**None**

Bus Status Codes

•The 8288 produces one or two of these eight command signals for each bus cycles. For instance, when the 8086 outputs the code $S_2S_1S_0$ equals 001, it indicates that an ***I/O read cycle*** is to be performed.

•In the code 111 is output by the 8086, it is signaling that no bus activity is to take place.

•The control outputs produced by the 8288 are DEN, DT/R and ALE. These 3 signals provide the same functions as those described for the minimum system mode. This set of bus commands and control signals is compatible with the Multibus and industry standard for interfacing microprocessor systems.

•***The output of 8289 are bus arbitration signals***:

Bus busy (BUSY), *common bus request* (CBRQ), *bus priority out* (BPRO), *bus priority in* (BPRN), *bus request* (BREQ) and *bus clock* (BCLK).
•They correspond to the bus exchange signals of the Multibus and are used to lock other processor off the system bus during the execution of an instruction by the 8086.
•In this way the processor can be assured of uninterrupted access to common system resources such as ***global memory.***
•**Queue Status Signals** : Two new signals that are produced by the 8086 in the maximum-mode system are queue status outputs QS_0 and QS_1. Together they form a 2-bit queue status code, QS_1QS_0.
•Following table shows the four different queue status.

QS_1	QS_0	Queue Status
0 (low)	**0**	**No Operation. During the last clock cycle, nothing was taken from the queue.**
0	**1**	**First Byte. The byte taken from the queue was the first byte of the instruction.**
1 (high)	**0**	**Queue Empty. The queue has been reinitialized as a result of the execution of a transfer instruction.**
1	**1**	**Subsequent Byte. The byte taken from the queue was a subsequent byte of the instruction.**

Queue status codes

•**Local Bus Control Signal – Request / Grant Signals**: In a maximum mode configuration, the minimum mode HOLD, HLDA interface is also changed. These two are replaced by request/grant lines RQ/ GT_0 and RQ/ GT_1, respectively. They provide a prioritized bus access mechanism for accessing the local bus.

Internal Registers of 8086

•The 8086 has four groups of the user accessible internal registers. They are the instruction pointer, four data registers, four pointer and index register, four segment registers.

•The 8086 has a total of fourteen 16-bit registers including a 16 bit register called the ***status register***, with 9 of bits implemented for status and control flags.
•Most of the registers contain data/instruction offsets within 64 KB memory segment. There are four different 64 KB segments for instructions, stack, data and extra data. To specify where in 1 MB of processor memory these 4 segments are located the processor uses four segment registers:
•**Code segment** (CS) is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly. The CS register is automatically updated during far jump, far call and far return instructions.
•**Stack segment** (SS) is a 16-bit register containing address of 64KB segment with program stack. By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction.
•**Data segment** (DS) is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment. DS register can be changed directly using POP and LDS instructions.
•**Accumulator** register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the low-order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations and string manipulation.
•**Base** register consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. BL in this case contains the low-order byte of the word, and BH contains the high-order byte. BX register usually contains a data pointer used for based, based indexed or register indirect addressing.
•**Count** register consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX. When combined, CL register contains the low-order byte of the word, and CH contains the high-order byte. Count register can be used in Loop, shift/rotate instructions and as a counter in string manipulation,.
•**Data** register consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX. When combined, DL register contains the low-order byte of the word, and DH contains the high-order byte. Data register can be used as a port number in I/O operations. In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.
•The following registers are both general and index registers:
•**Stack Pointer** (SP) is a 16-bit register pointing to program stack.
•**Base Pointer** (BP) is a 16-bit register pointing to data in stack segment. BP register is usually used for based, based indexed or register indirect addressing.
•**Source Index** (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data address in string manipulation instructions.
•**Destination Index** (DI) is a 16-bit register. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions.
Other registers:

•**Instruction Pointer** (IP) is a 16-bit register.
•**Flags** is a 16-bit register containing 9 one bit flags.
•**Overflow Flag** (OF) - set if the result is too large positive number, or is too small negative number to fit into destination operand.
•**Direction Flag** (DF) - if set then string manipulation instructions will auto-decrement index registers. If cleared then the index registers will be auto-incremented.
•**Interrupt-enable Flag** (IF) - setting this bit enables maskable interrupts.
•**Single-step Flag** (TF) - if set then single-step interrupt will occur after the next instruction.
•**Sign Flag** (SF) - set if the most significant bit of the result is set.
•**Zero Flag** (ZF) - set if the result is zero.
•**Auxiliary carry Flag** (AF) - set if there was a carry from or borrow to bits 0-3 in the AL register.
•**Parity Flag** (PF) - set if parity (the number of "1" bits) in the low-order byte of the result is even.
•**Carry Flag** (CF) - set if there was a carry from or borrow to the most significant bit during last result calculation.

Addressing Modes

•**Implied** - the data value/data address is implicitly associated with the instruction.
•**Register** - references the data in a register or in a register pair.
•**Immediate** - the data is provided in the instruction.
•**Direct** - the instruction operand specifies the memory address where data is located.
•**Register indirect** - instruction specifies a register containing an address, where data is located. This addressing mode works with SI, DI, BX and BP registers.
•**Based** :- 8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP), the resulting value is a pointer to location where data resides.
•**Indexed** :- 8-bit or 16-bit instruction operand is added to the contents of an index register (SI or DI), the resulting value is a pointer to location where data resides.
•**Based Indexed** :- the contents of a base register (BX or BP) is added to the contents of an index register (SI or DI), the resulting value is a pointer to location where data resides.
•**Based Indexed with displacement** :- 8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP) and index register (SI or DI), the resulting value is a pointer to location where data resides.

Memory

•Program, data and stack memories occupy the same memory space. As the most of the processor instructions use 16-bit pointers the processor can effectively address only 64 KB of memory.
• To access memory outside of 64 KB the CPU uses special segment registers to specify where the code, stack and data 64 KB segments are positioned within 1 MB of memory (see the "Registers" section below).

•16-bit pointers and data are stored as:
address: low-order byte
address+1: high-order byte

•**Program memory** - program can be located anywhere in memory. Jump and call instructions can be used for short jumps within currently selected 64 KB code segment, as well as for far jumps anywhere within 1 MB of memory.

•All conditional jump instructions can be used to jump within approximately +127 to -127 bytes from current instruction.

•**Data memory** - the processor can access data in any one out of 4 available segments, which limits the size of accessible memory to 256 KB (if all four segments point to different 64 KB blocks).

•Accessing data from the Data, Code, Stack or Extra segments can be usually done by prefixing instructions with the DS:, CS:, SS: or ES: (some registers and instructions by default may use the ES or SS segments instead of DS segment).

•Word data can be located at odd or even byte boundaries. The processor uses two memory accesses to read 16-bit word located at odd byte boundaries. Reading word data from even byte boundaries requires only one memory access.

•**Stack memory** can be placed anywhere in memory. The stack can be located at odd memory addresses, but it is not recommended for performance reasons (see "Data Memory" above).

Reserved locations:

•0000h - 03FFh are reserved for interrupt vectors. Each interrupt vector is a 32-bit pointer in format segment: offset.

•FFFF0h - FFFFFh - after RESET the processor always starts program execution at the FFFF0h address.

Interrupts

The processor has the following interrupts:

•**INTR** is a maskable hardware interrupt. The interrupt can be enabled/disabled using STI/CLI instructions or using more complicated method of updating the FLAGS register with the help of the POPF instruction.

• When an interrupt occurs, the processor stores FLAGS register into stack, disables further interrupts, fetches from the bus one byte representing interrupt type, and jumps to interrupt processing routine address of which is stored in location 4 * <interrupt type>. Interrupt processing routine should return with the IRET instruction.

•**NMI** is a non-maskable interrupt. Interrupt is processed in the same way as the INTR interrupt. Interrupt type of the NMI is 2, i.e. the address of the NMI processing routine is stored in location 0008h. This interrupt has higher priority then the maskable interrupt.

•**Software interrupts** can be caused by:

•INT instruction - breakpoint interrupt. This is a type 3 interrupt.

•INT <interrupt number> instruction - any one interrupt from available 256 interrupts.

•INTO instruction - interrupt on overflow

•Single-step interrupt - generated if the TF flag is set. This is a type 1 interrupt. When the CPU processes this interrupt it clears TF flag before calling the interrupt processing routine.

•**Processor exceptions**: Divide Error (Type 0), Unused Opcode (type 6) and Escape opcode (type 7).

•Software interrupt processing is the same as for the hardware interrupts.

MODULE 2

ASSEMBLY LANGUAGE OF 8086

Contents

- Description of Instructions
- Assembly directives
- Algorithms with assembly software programs

Instruction Description

- **AAA** Instruction - ASCII Adjust after Addition
- **AAD** Instruction - ASCII adjust before Division
- **AAM** Instruction - ASCII adjust after Multiplication
- **AAS** Instruction - ASCII Adjust for Subtraction
- **ADC** Instruction - Add with carry.
- **ADD** Instruction - ADD destination, source
- **AND** Instruction - AND corresponding bits of two operands

Example

- **AAA** Instruction - AAA converts the result of the addition of two valid unpacked BCD digits to a valid 2-digit BCD number and takes the AL register as its implicit operand.

  Two operands of the addition must have its lower 4 bits contain a number in the range from 0-9.The AAA instruction then adjust AL so that it contains a correct BCD digit. If the addition produce carry (AF=1), the AH register is incremented and the carry CF and auxiliary carry AF flags are set to 1. If the addition did not produce a decimal carry, CF and AF are cleared to 0 and AH is not altered. In both cases the higher 4 bits of AL are cleared to 0.

AAA will adjust the result of the two ASCII characters that were in the range from 30h (“0”) to 39h(“9”).This is because the lower 4 bits of those character fall in the range of 0-9.The result of addition is not a ASCII character but it is a BCD digit.

- **Example:**

```
MOV     AH,0   ;Clear AH for MSD
MOV     AL,6   ;BCD 6 in AL
ADD     AL,5   ;Add BCD 5 to digit in AL
AAA            ;AH=1, AL=1 representing BCD 11.
```

- **AAD** Instruction - ADD converts unpacked BCD digits in the AH and AL register into a single binary number in the AX register in preparation for a division operation.

  Before executing AAD, place the Most significant BCD digit in the AH register and Last significant in the AL register. When AAD is executed, the two BCD digits are combined into a single binary number by setting AL=(AH*10)+AL and clearing AH to 0.

- **Example:**

```
MOV    AX,0205h    ;The unpacked BCD number 25
AAD                ;After AAD , AH=0 and
                   ;AL=19h (25)
```

After the division AL will then contain the unpacked BCD quotient and AH will contain the unpacked BCD remainder.

- **Example:**

```
                    ;AX=0607 unpacked BCD for 67 decimal
                    ;CH=09H
AAD                 ;Adjust to binary before division
                    ;AX=0043 = 43H =67 decimal
DIV  CH             ;Divide AX by unpacked BCD in CH
                    ;AL = quotient = 07 unpacked BCD
                    ;AH = remainder = 04 unpacked BCD
```

- **AAM** Instruction - AAM converts the result of the multiplication of two valid unpacked BCD digits into a valid 2-digit unpacked BCD number and takes AX as an implicit operand.

  To give a valid result the digits that have been multiplied must be in the range of 0 – 9 and the result should have been placed in the AX register. Because both operands of multiply are required to be 9 or less, the result must be less than 81 and thus is completely contained in AL.

  AAM unpacks the result by dividing AX by 10, placing the quotient (MSD) in AH and the remainder (LSD) in AL.

- **Example:**

```
MOV        AL, 5
MOV        BL, 7
MUL        BL      ;Multiply AL by BL , result in AX
AAM                ;After AAM, AX =0305h (BCD 35)
```

- **AAS** Instruction - AAS converts the result of the subtraction of two valid unpacked BCD digits to a single valid BCD number and takes the AL register as an implicit operand. The two operands of the subtraction must have its lower 4 bit contain number in the range from 0 to 9 .The AAS instruction then adjust AL so that it contain a correct BCD digit.

```
MOV     AX,0901H     ;BCD 91
SUB     AL, 9        ;Minus 9
AAS                  ; Give AX =0802 h (BCD  82)
```

(a)

```
                              ;AL =0011 1001 =ASCII   9
                              ;BL=0011 0101 =ASCII   5
SUB          AL, BL           ;(9 - 5) Result :
                              ;AL = 00000100 = BCD 04,CF = 0
AAS                           ;Result :
                              ;AL=00000100 =BCD  04
                              ;CF = 0 NO Borrow required
```

(b)

```
                   ;AL = 0011 0101 =ASCII    5
                   ;BL = 0011 1001 = ASCII   9
SUB   AL, BL       ;( 5 - 9 )  Result :
                   ;AL = 1111 1100 =  - 4
                   ; in 2's complement CF = 1
AAS                ;Results :
                   ;AL = 0000 0100 =BCD  04
                   ;CF = 1 borrow  needed .
```

- **ADD** Instruction - These instructions add a number from source to a number from some destination and put the result in the specified destination. The add with carry instruction ADC, also add the status of the carry flag into the result. The source and destination must be of same type , means they must be a byte location or a word location. If you want to add a byte to a word, you must copy the byte to a word location and fill the upper byte of the word with zeroes before adding.
- **EXAMPLE:**

```
ADD     AL,74H      ;Add immediate number 74H to
                    ; content of AL
```

```
ADC     CL,BL      ;Add contents of BL plus
                   ;carry  status to contents of CL.
                   ;Results in CL

ADD     DX, BX     ;Add  contents of BX to contents
                   ;of DX

ADD     DX, [SI]   ;Add word from memory at
                   ;offset [SI] in DS to contents of DX
```

```
                        ; Addition of Un Signed numbers
ADD      CL,  BL  ;CL = 01110011 =115  decimal
                  ;+ BL = 01001111 = 79 decimal
                  ;Result in CL = 11000010 = 194 decimal

                  ; Addition of Signed numbers
ADD      CL,  BL  ;CL = 01110011 = + 115  decimal
                  ;+ BL = 01001111 = +79 decimal
                  ;Result in CL = 11000010 = - 62 decimal

; Incorrect because result is too large to fit in 7 bits.
```

- **AND** Instruction - This Performs a bitwise Logical AND of two operands. The result of the operation is stored in the op1 and used to set the flags.

  AND op1, op2

  To perform a bitwise AND of the two operands, each bit of the result is set to 1 if and only if the corresponding bit in both of the operands is 1, otherwise the bit in the result I cleared to 0 .

```
AND     BH, CL       ;AND byte in CL with byte in BH
                     ;result in BH
AND     BX,00FFh     ;AND word in BX with immediate
                     ;00FFH. Mask upper byte, leave
                     ;lower unchanged
```

```
AND     CX,[SI]      ; AND word at offset [SI] in data
                     ;segment with word in CX
                     ;register . Result in CX register .

                     ;BX = 10110011 01011110
AND     BX,00FFh     ;Mask out upper 8 bits of BX
                     ;Result BX = 00000000  01011110
                     ;CF =0 , OF = 0, PF = 0, SF = 0 ,
                     ;ZF = 0
```

- **CALL** Instruction

- Direct within-segment (near or intrasegment)
- Indirect within-segment (near or intrasegment)
- Direct to another segment (far or intersegment)
- Indirect to another segment (far or intersegment)

- **CBW** Instruction - Convert signed Byte to signed word

- **CLC** Instruction - Clear the carry flag

- **CLD** Instruction - Clear direction flag

- **CLI** Instruction - Clear interrupt flag

- **CMC** Instruction - Complement the carry flag

- **CMP** Instruction - Compare byte or word- CMP destination, source.

- **CMPS/CMPSB/ CMPSW** Instruction - Compare string bytes or string words

- **CWD** Instruction - Convert Signed Word to - Signed Double word

Example

- **CALL** Instruction - This Instruction is used to transfer execution to a subprogram or procedure. There are two basic types of CALL 's : Near and Far.

  A Near CALL is a call to a procedure which is in the same code segment as the CALL instruction .

  When 8086 executes the near CALL instruction it decrements the stack pointer by two and copies the offset of the next instruction after the CALL on the stack. This offset saved on the stack is referred as the return address, because this is the address that execution will returns to after the procedure executes. A near CALL instruction will also load the instruction pointer with the offset of the first instruction in the procedure.

A RET instruction at the end of the procedure will return execution to the instruction after the CALL by coping the offset saved on the stack back to IP.

A Far CALL is a call to a procedure which is in a different from that which contains the CALL instruction . When 8086 executes the Far CALL instruction it decrements the stack pointer by two again and copies the content of CS register to the stack. It then decrements the stack pointer by two again and copies the offset contents offset of the instruction after the CALL to the stack. Finally it loads CS with segment base of the segment which contains the procedure and IP with the offset of the first instruction of the procedure in segment. A RET instruction at end of procedure will return to the next instruction after the CALL by restoring the saved CS and IP from the stack.

;Direct within-segment (near or intrasegment)

CALL MULTO ;MULTO is the name of the procedure. The assembler determines displacement of MULTO from the instruction after the CALL and codes this displacement in as part of the instruction .

;Indirect within-segment (near or intrasegment)

CALL BX ; BX contains the offset of the first instruction of the procedure .Replaces contents of word of IP with contents o register BX.

CALL WORD PTR[BX] ;Offset of first instruction of procedure is in two memory addresses in DS .Replaces contents of IP with contents of word memory location in DS pointed to by BX.

;Direct to another segment- far or intersegment.

```
CALL      SMART          ;SMART is the name of the
                         ;Procedure
SMART     PROC    FAR ; Procedure must be declare as
                         ;an far
```

- **CBW** Instruction - CBW converts the signed value in the AL register into an equivalent 16 bit signed value in the AX register by duplicating the sign bit to the left.

  This instruction copies the sign of a byte in AL to all the bits in AH. AH is then said to be the sign extension of AL.

  Example:

```
          ;AX =   00000000  10011011 = - 155 decimal
CBW       ;Convert signed byte in AL to signed word in
          ;AX.
          ;Result in AX = 11111111  10011011
          ; = - 155 decimal
```

- **CLC** Instruction - CLC clear the carry flag (CF) to 0 This instruction has no affect on the processor, registers, or other flags. It is often used to clear the CF before returning from a procedure to indicate a successful termination. It is also use to clear the CF during rotate operation involving the CF such as ADC, RCL, RCR .

  Example:

  CLC ;Clear carry flag.

- **CLD** Instruction - This instruction reset the designation flag to zero. This instruction has no effect on the registers or other flags. When the direction flag is cleared / reset SI and DI will automatically be incremented when one of the string instruction such as MOVS, CMPS, SCAS,MOVSB and STOSB executes.

Example :

```
CLD        ;Clear direction flag so that string pointers
                     ;auto increment
```

- **CLI** Instruction - This instruction resets the interrupt flag to zero. No other flags are affected. If the interrupt flag is reset , the 8086 will not respond to an interrupt signal on its INTR input. This CLI instruction has no effect on the nonmaskable interrupt input, NMI
- **CMC** Instruction - If the carry flag CF is a zero before this instruction, it will be set to a one after the instruction. If the carry flag is one before this instruction, it will be reset to a zero after the instruction executes. CMC has no effect on other flags.

  Example:

  CMC ;Invert the carry flag.

- **CWD** Instruction - CWD converts the 16 bit signed value in the AX register into an equivalent 32 bit signed value in DX: AX register pair by duplicating the sign bit to the left.

  The CWD instruction sets all the bits in the DX register to the same sign bit of the AX register. The effect is to create a 32- bit signed result that has same integer value as the original 16 bit operand.

  Example:

  Assume AX contains C435h. If the CWD instruction is executed, DX will contain FFFFh since bit 15 (MSB) of AX was 1. Both the original value of AX (C435h) and resulting value of DX : AX (FFFFC435h) represents the same signed number.

Example:

```
            ;DX = 00000000 00000000
            ;AX = 11110000 11000111 = - 3897 decimal
CWD         ;Convert signed word in AX to signed double
            ;word in DX:AX
            ;Result  DX = 11111111 11111111
            ;AX = 11110000 11000111 = -3897 decimal .
```

- **DAA** Instruction - Decimal Adjust Accumulator
- **DAS** Instruction - Decimal Adjust after Subtraction
- **DEC** Instruction - Decrement destination register or memory DEC destination.
- **DIV** Instruction - Unsigned divide-Div source
- **ESC** Instruction

- **DIV** Instruction - This instruction is used to divide an Unsigned word by a byte or to divide an unsigned double word by a word.

  When dividing a word by a byte , the word must be in the AX register. After the division AL will contains an 8-bit result (quotient) and AH will contain an 8- bit remainder. If an attempt is made to divide by 0 or the quotient is too large to fit in AL (greater than FFH), the 8086 will automatically do a type 0 interrupt .

  Example:

```
DIV        BL     ;Word in AX / byte in BL
                  ;Quotient in AL . Remainder in AH.
```

When a double word is divided by a word, the most significant word of the double word must be in DX and the least significant word of the double word must be in AX. After the division AX will contain the 16 –bit result (quotient) and DX will contain a 16 bit remainder. Again , if an attempt is made to divide by zero or quotient is too large to fit in AX (greater than FFFFH) the 8086 will do a type of 0 interrupt .

Example:

```
DIV     CX      ; (Quotient) AX= (DX:AX)/CX
                : (Reminder) DX=(DX:AX)%CX
```

For DIV the dividend must always be in AX or DX and AX, but the source of the divisor can be a register or a memory location specified by one of the 24 addressing modes.

If you want to divide a byte by a byte, you must first put the dividend byte in AL and fill AH with all 0’s . The SUB AH,AH instruction is a quick way to do.

If you want to divide a word by a word, put the dividend word in AX and fill DX with all 0’s. The SUB DX,DX instruction does this quickly.

- **Example: ; AX = 37D7H = 14, 295 decimal**

; BH = 97H = 151 decimal

DIV BH ;AX / BH

; AX = Quotient = 5EH = 94 decimal

; AH = Remainder = 65H = 101 decimal

- **ESC** Instruction - Escape instruction is used to pass instruction to a coprocessor such as the 8087 math coprocessor which shares the address and data bus with an 8086. Instruction for the coprocessor are represented by a 6 bit code embedded in the escape instruction. As the 8086 fetches instruction byte, the coprocessor also catches these bytes from data bus and puts them in its queue. The coprocessor treats all of the 8086 instruction as an NOP. When 8086 fetches an ESC instruction , the coprocessor decodes the instruction and carries out the action specified by the 6 bit code. In most of the case 8086 treats ESC instruction as an NOP.

- **HLT** Instruction - HALT processing

- **IDIV** Instruction - Divide by signed byte or word
  IDIV source

- **IMUL** Instruction - Multiply signed number-IMUL
  source

- **IN** Instruction - Copy data from a port
  IN accumulator, port

- **INC** Instruction - Increment - INC destination

- **HALT** Instruction - 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 .
- **IDIV** Instruction - This instruction is used to divide a signed word by a signed byte or to divide a signed double word by a signed word.
- **Example:**

```
IDIV        BL      ;Signed word in AX is divided by signed
                    ;byte in BL
```

- **Example:**

```
IDIV   BP       ;divide a Signed double word in DX and
                ;AX by signed word in BP
IDIV   BYTE PTR[BX]        ; divide AX by a byte at
                           ;offset [BX] in DS
```

- **A signed word divided by a signed byte**

```
              ;AX = 00000011  10101011 = 03ABH=39 decimal
              ;BL = 11010011 = D3H = - 2DH = - 45 decimal
IDIV     BL;Quotient AL= ECH = - 14H = -20 decimal
              ;Remainder AH = 27H = + 39 decimal
```

- **IMUL** Instruction - This instruction performs a signed multiplication.

  IMUL op ;In this form the accumulator is the multiplicand and op is the multiplier. op may be a register or a memory operand.

  IMUL op1, op2 ;In this form op1 is always be a register operand and op2 may be a register or a memory operand.

- **Example:**

  IMUL BH ;Signed byte in AL times multiplied by
  ;signed byte in BH and result in AX .

- **Example:**

```
                          ; 69 * 14
                          ; AL = 01000101 = 69 decimal
                          ; BL = 00001110 = 14 decimal
IMUL      BL              ;AX = 03C6H=  + 966 decimal
                          ;MSB = 0 because positive result

                          ; - 28 * 59
                          ; AL = 11100100 = - 28 decimal
                          ;BL = 00001110 =  14 decimal
IMUL      BL              ;AX = F98Ch = - 1652 decimal
                          ; MSB = 1 because negative result
```

- **IN** Instruction - This IN instruction will copy data from a port to the AL or AX register.

  For the Fixed port IN instruction type the 8 – bit port address of a port is specified directly in the instruction.

- **Example:**

```
IN   AL,0C8H       ;Input a byte from port 0C8H to AL

IN   AX, 34H       ;Input a word from port 34H to AX

A_TO_D   EQU   4AH
IN   AX, A_TO_D ;Input a word from port 4AH to AX
```

For a variable port IN instruction, the port address is loaded in DX register before IN instruction. DX is 16 bit. Port address range from 0000H – FFFFH.

- **Example:**

```
MOV     DX, 0FF78H  ;Initialize DX point to port
IN  AL, DX          ;Input a byte from a 8 bit port
                    ;0FF78H to AL


IN  AX, DX          ;Input a word from 16 bit port to
                    ;0FF78H  to AX.
```

- **INC** Instruction - INC instruction adds one to the operand and sets the flag according to the result. INC instruction is treated as an unsigned binary number.

- **Example:**

```
                ; AX = 7FFFh
    INC  AX     ;After this instruction AX = 8000h

    INC   BL    ; Add 1 to the contents of BL register
    INC   CL    ; Add 1 to the contents of CX register.
```

➢ **INT** Instruction	-	Interrupt program
➢ **INTO** Instruction	-	Interrupt on overflow.
➢ **IRET** Instruction	-	Interrupt return
➢ **JA/JNBE** Instruction	-	Jump if above/Jump if not below nor equal.
➢ **JAE/JNB/ JNC** Instructions	-	Jump if above or equal/ Jump if not below/ Jump if no carry.

- **JA / JNBE -** This instruction performs the Jump if above (or) Jump if not below or equal operations according to the condition, if CF and ZF = 0 .
- **Example:** (1)

```
CMP   AX, 4371H     ;Compare by subtracting 4371H
                    ;from AX
JA    RUN_PRESS ;Jump to label RUN_PRESS if
                    ;AX    above 4371H
      ( 2 )
CMP   AX, 4371H     ;Compare ( AX – 4371H)
JNBE  RUN_PRESS ;Jump to label RUN_PRESS if
                    ;AX not below or equal to 4371H
```

- **JAE / JNB / JNC** - This instructions performs the Jump if above or equal, Jump if not below, Jump if no carry operations according to the condition, if CF = 0.
- **Examples**:

```
1. CMP      AX, 4371H     ;Compare ( AX – 4371H)
   JAE      RUN           ;Jump to the label RUN if AX is
                          ;above or equal to 4371H .
2. CMP      AX, 4371H     ;Compare ( AX – 4371H)
   JNB      RUN_1         ;Jump to the label RUN_1 if AX
                          ;is not below than 4371H
3. ADD      AL, BL        ; Add AL, BL. If result is with in
   JNC               OK   ;acceptable range, continue
```

- **JB/JC/JNAE** Instruction - Jump if below/Jump if carry/ Jump if not above nor equal
- **JBE/JNA** Instructions - Jump if below or equal / Jump if not above
- **JCXZ** Instruction - Jump if the CX register is zero
- **JE/JZ** Instruction - Jump if equal/Jump if zero
- **JG/JNLE** Instruction - Jump if greater/Jump if not less than nor equal

- **JB/JC/JNAE** Instruction - This instruction performs the Jump if below (or) Jump if carry (or) Jump if not below/ equal operations according to the condition,

  if CF = 1

- **Example:**

```
1.  CMP    AX, 4371H    ;Compare ( AX – 4371H )
    JB     RUN_P        ;Jump to label RUN_P if AX is
                        ;below 4371H

2.  ADD    BX, CX       ;Add two words and Jump to
     JC    ERROR        ; label ERROR if CF = 1
```

- **JBE/JNA** Instruction - This instruction performs the Jump if below or equal (or) Jump if not above operations according to the condition, if CF and ZF = 1
- **Example:**

```
CMP     AX, 4371H     ;Compare ( AX – 4371H )
JBA     RUN           ;Jump to label RUN if AX is
                      ;below or equal to 4371H


CMP     AX, 4371H     ;Compare ( AX – 4371H )
JNA     RUN_R         ;Jump to label RUN_R if AX is
                      ;not above  than 4371H
```

- **JCXZ** Instruction - This instruction performs the Jump if CX register is zero. If CX does not contain all zeros, execution will simply proceed to the next instruction.
- **Example:**

```
        JCXZ   SKIP_LOOP ;If CX = 0, skip the process
NXT: SUB     [BX], 07H      ;Subtract 7 from data value
        INC     BX                  ; BX point to next value
        LOOP NXT                 ; Loop until CX = 0
        SKIP_LOOP               ;Next instruction
```

- **JE/JZ** Instruction Instruction - This instruction performs the Jump if equal (or) Jump if zero operations according to the condition if ZF = 1
- **Example:**

```
NXT:CMP      BX, DX ;Compare ( BX – DX )
     JE      DONE   ;Jump to DONE if BX = DX,
     SUB     BX, AX ;Else subtract Ax
     INC     CX      ;Increment counter
     JUMP    NXT     ;Check again
DONE: MOV    AX, CX  ;Copy count to AX
```

- **Example:**

```
IN     AL, 8FH      ;read data from port 8FH
SUB    AL, 30H      ;Subtract minimum value
JZ     STATR        ; Jump to label if result of
                    ;subtraction  was 0
```

- **JG/JNLE** Instruction - This instruction performs the Jump if greater (or) Jump if not less than or equal operations according to the condition if ZF =0 and SF = OF
- **Example:**

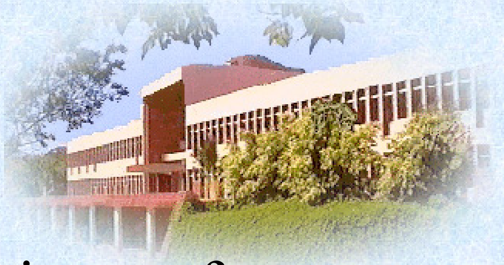
```
CMP     BL, 39H     ;Compare by subtracting
                    ;39H from BL
JG      NEXT1       ;Jump to label if BL is
                    ;more positive than 39H


CMP     BL, 39H     ;Compare by subtracting
                    ;39H from BL
JNLE    NEXT2       ;Jump to label if BL is not
                    less than or equal 39H
```

- **JGE/JNL** Instruction - Jump if greater than or equal/ Jump if not less than

- **JL/JNGE** Instruction - Jump if less than/Jump if not greater than or equal

- **JLE/JNG** Instruction - Jump if less than or equal/ Jump if not greater

- **JMP** Instruction - Unconditional jump to - specified destination



- **JGE/JNL** Instruction - This instruction performs the Jump if greater than or equal / Jump if not less than operation according to the condition if SF = OF
- **Example:**

```
CMP     BL, 39H      ;Compare by the
                     ;subtracting 39H from BL
JGE     NEXT11       ;Jump to label if BL is
                     ;more positive than 39H
                     ; or equal to 39H
CMP     BL, 39H      ;Compare by subtracting
                     ;39H from BL
JNL     NEXT22       ;Jump to label if BL is not
                     ;less than 39H
```

- **JL/JNGE** Instruction - This instruction performs the Jump if less than / Jump if not greater than or equal operation according to the condition, if SF ≠ OF
- **Example:**

```
CMP   BL, 39H        ;Compare by subtracting 39H
                     ;from BL
JL    AGAIN          ;Jump to the label if BL is more
                     ;negative than 39H


CMP   BL, 39H        ;Compare by subtracting 39H
                     ;from BL
JNGE  AGAIN1         ; Jump to the label if BL is  not
                     ;more positive than 39H or
                     ;not equal to 39H
```

- **JLE/JNG** Instruction - This instruction performs the Jump if less than or equal / Jump if not greater operation according to the condition, if ZF=1 and SF ≠ OF
- **Example:**

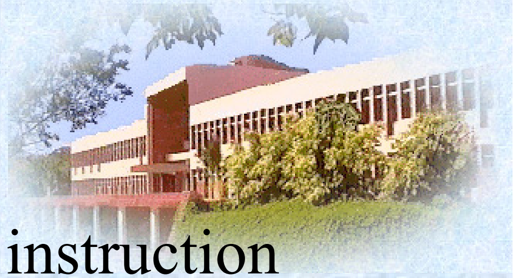
```
CMP     BL, 39h     ; Compare by subtracting 39h
                    ;from BL
JLE     NXT1        ;Jump to the label if BL is more
                    ;negative than 39h or equal to 39h


CMP     BL, 39h     ;Compare by subtracting 39h
                    ;from BL
JNG     AGAIN2      ; Jump to the label if BL is  not
                    ;more positive than 39h
```

- **JNA/JBE** Instruction - Jump if not above/Jump if below or equal

- **JNAE/JB** Instruction - Jump if not above or equal/ Jump if below

- **JNB/JNC/JAE** Instruction - Jump if not below/Jump if no carry/Jump if above or equal

- **JNE/JNZ** Instruction - Jump if not equal/Jump if not zero



- **JNE/JNZ** Instruction - This instruction performs the Jump if not equal / Jump if not zero operation according to the condition, if ZF=0
- **Example:**

```
NXT: IN      AL, 0F8H     ;Read data value from port
     CMP     AL, 72       ;Compare ( AL – 72 )
     JNE     NXT          ;Jump to NXT if AL ≠ 72
     IN      AL, 0F9H     ;Read next port when AL = 72


          MOV       BX, 2734H    ; Load BX as counter
   NXT_1:ADD        AX, 0002H    ;Add count factor to AX
          DEC       BX           ;Decrement BX
          JNZ       NXT_1        Repeat until BX = 0
```

- **JNG/JLE** Instruction - Jump if not greater/ Jump if less than or equal

- **JNGE/JL** Instruction - Jump if not greater than nor equal/Jump if less than

- **JNL/JGE** Instruction - Jump if not less than/ Jump if greater than or equal

- **JNLE/JG** Instruction - Jump if not less than nor equal to /Jump if greater than

- **JNO** Instruction – Jump if no overflow
- **JNP/JPO** Instruction – Jump if no parity/ Jump if parity odd
- **JNS** Instruction - Jump if not signed (Jump if positive)
- **JNZ/JNE** Instruction - Jump if not zero / jump if not equal
- **JO** Instruction - Jump if overflow

- **JNO** Instruction – This instruction performs the Jump if no overflow operation according to the condition, if OF=0
- **Example:**

```
        ADD   AL, BL        ; Add signed bytes in AL and BL
        JNO   DONE          ;Process done if no overflow -
        MOV   AL, 00H       ;Else load error code in AL
DONE:   OUT   24H, AL       ; Send result to display
```

- **JNP/JPO** Instruction – This instruction performs the Jump if not parity / Jump if parity odd operation according to the condition, if PF=0
- **Example:**

```
IN      AL, 0F8H     ;Read ASCII char from UART
OR      AL, AL       ;Set flags
JPO     ERROR1       ;If even parity executed, if not
                     ;send error message
```

- **JNS** Instruction - This instruction performs the Jump if not signed (Jump if positive) operation according to the condition, if SF=0
- Example:

```
DEC     AL     ;Decrement  counter
JNS     REDO   ; Jump to label REDO if counter has not
               ;decremented to FFH
```

- **JO** Instruction - This instruction performs Jump if overflow operation according to the condition OF = 0
- **Example:**

```
ADD     AL, BL      ;Add signed bits in AL and BL
JO      ERROR       ; Jump to label if overflow occur
                    ;in addition
MOV     SUM, AL     ; else put the result in memory
                    ;location named SUM
```

- **JPE/JP** Instruction - Jump if parity even/ Jump if parity

- **JPO/JNP** Instruction - Jump if parity odd/ Jump if no parity

- **JS** Instruction - Jump if signed (Jump if negative)

- **JZ/JE** Instruction - Jump if zero/Jump if equal

- **JPE/JP** Instruction - This instruction performs the Jump if parity even / Jump if parity operation according to the condition, if PF=1

Example:

```
IN     AL, 0F8H     ;Read ASCII char from UART
OR     AL, AL       ;Set flags
JPE    ERROR2       ;odd parity is expected, if not
                    ;send error message
```

- **JS** Instruction - This instruction performs the Jump if sign operation according to the condition, if SF=1
- **Example:**

```
ADD    BL, DH       ;Add signed bytes DH to BL
JS     JJS_S1       ;Jump to label if result is
                    ;negative
```

- **LAHF** Instruction - Copy low byte of flag register to AH

- **LDS** Instruction - Load register and Ds with words from memory –

  LDS register, memory address of first word

- **LEA** Instruction - Load effective address-LEA register, source

- **LES** Instruction - Load register and ES with words from memory –LES register, memory address of first word.

- **LAHF** Instruction - LAHF instruction copies the value of SF, ZF, AF, PF, CF, into bits of 7, 6, 4, 2, 0 respectively of AH register. This LAHF instruction was provided to make conversion of assembly language programs written for 8080 and 8085 to 8086 easier.
- **LDS** Instruction - This instruction loads a far pointer from the memory address specified by op2 into the DS segment register and the op1 to the register. **LDS op1, op2**
- **Example:**

LDS BX, [4326] ; copy the contents of the memory at displacement 4326H in DS to BL, contents of the 4327H to BH. Copy contents of 4328H and 4329H in DS to DS register.

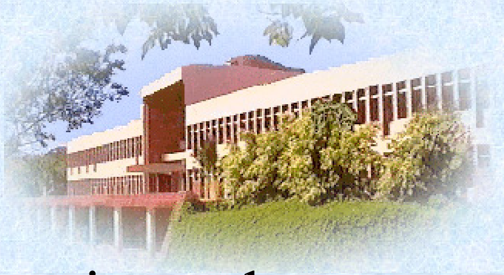
- **LEA** Instruction - This instruction indicates the offset of the variable or memory location named as the source and put this offset in the indicated 16 – bit register.
- **Example:**

```
LEA    BX, PRICE      ;Load BX with offset of PRICE
                      ;in DS
LEA    BP, SS:STAK;Load BP with offset of STACK
                      ;in SS
LEA    CX, [BX][DI] ;Load CX with EA=BX + DI
```

- **LOCK** Instruction - Assert bus lock signal
- **LODS/LODSB/ LODSW** Instruction - Load string byte into AL or Load string word into AX.
- **LOOP** Instruction - Loop to specified label until CX = 0
- **LOOPE / LOOPZ** Instruction - loop while CX ≠ 0 and ZF = 1

- **LODS/LODSB/LODSW** Instruction - This instruction copies a byte from a string location pointed to by SI to AL or a word from a string location pointed to by SI to AX. If DF is cleared to 0,SI will automatically incremented to point to the next element of string.
- **Example:**

```
CLD         ;Clear direction flag so SI is auto incremented

MOV         SI, OFFSET SOURCE_STRING
                      ;point SI at  start of the string

LODS        SOUCE_STRING    ;Copy byte or word from
                            ;string to AL or AX
```



- **LOOP** Instruction - This instruction is used to repeat a series of instruction some number of times
- **Example:**

```
        MOV   BX, OFFSET PRICE
                            ;Point BX at first element in array
        MOV   CX, 40          ;Load CX with number of
                              ;elements in array
NEXT: MOV AL, [BX]            ; Get elements from array
        ADD   AL, 07H         ;Ad correction factor
        DAA                   ; decimal adjust result
        MOV   [BX], AL        ; Put result back in array
        LOOP NEXT             ; Repeat until all elements
                              ;adjusted.
```

- **LOOPE / LOOPZ** Instruction - This instruction is used to repeat a group of instruction some number of times until CX = 0 and ZF = 0
- **Example:**

```
        MOV   BX, OFFSET   ARRAY
                                 ;point BX at start of the array
        DEC   BX
        MOV   CX, 100            ;put number of array elements in
                                 ;CX
NEXT:INC      BX                 ;point to next element in array
        CMP   [BX], 0FFH         ;Compare array elements FFH
        LOOP  NEXT
```

- **LOOPNE/LOOPNZ** Instruction - This instruction is used to repeat a group of instruction some number of times until CX = 0 and ZF = 1
- **Example:**

```
        MOV   BX, OFFSET   ARRAY1
                            ;point BX at start of the array
        DEC   BX
        MOV   CX, 100       ;put number of array elements in
                            ;CX
NEXT:INC    BX              ;point  to next elements in array
        CMP   [BX], 0FFH    ;Compare array elements 0DH
        LOOPNE  NEXT
```

- **MOV** Instruction - MOV destination, source

- **MOVS/MOVSB/ MOVSW** Instruction - Move string byte or string word-MOVS destination, source

- **MUL** Instruction - Multiply unsigned bytes or words-MUL source

- **NEG** Instruction - From 2’s complement – NEG destination

- **NOP** Instruction - Performs no operation.

- **MOV** Instruction - The MOV instruction copies a word or a byte of data from a specified source to a specified destination .

  MOV op1, op2

- **Example:**

```
MOV      CX, 037AH     ; MOV 037AH into the CX.
MOV      AX, BX        ;Copy the contents of register BX
                       ;to AX
MOV      DL,[BX]       ;Copy byte from memory at BX
                       to  DL , BX contains the offset of
                       ;byte  in DS.
```

- **MUL** Instruction - This instruction multiplies an unsigned multiplication of the accumulator by the operand specified by op. The size of op may be a register or memory operand . **MUL op**

Example:

```
                 ;AL = 21h (33 decimal)
                 ;BL = A1h(161 decimal )
MUL       BL     ;AX =14C1h (5313 decimal) since AH≠0,
                 ;CF and OF will set to 1.

MUL       BH     ; AL times BH, result in AX
MUL       CX     ;AX times CX, result high word in DX,
                 ;low word in AX.
```

- **NEG** Instruction - NEG performs the two’s complement subtraction of the operand from zero and sets the flags according to the result.

```
                    ;AX = 2CBh
     NEG    AX      ;after executing NEG result AX =FD35h.
```

Example:

```
NEG         AL      ;Replace number in AL with its 2’s
                            ;complement
NEG         BX      ;Replace word in BX with its 2’s
                    ;complement
NEG         BYTE PTR[BX]; Replace byte at offset BX in
                              ; DS with its 2’s complement
```

- **NOP** Instruction - This instruction simply uses up the three clock cycles and increments the instruction pointer to point to the next instruction. NOP does not change the status of any flag. The NOP instruction is used to increase the delay of a delay loop.

- **NOT** Instruction - Invert each bit of operand – NOT destination.
- **OR** Instruction - Logically OR corresponding of two operands- OR destination, source.
- **OUT** Instruction - Output a byte or word to a port – OUT port, accumulator AL or AX.
- **POP** Instruction - POP destination

- **NOT** Instruction - NOT perform the bitwise complement of op and stores the result back into op.

  NOT op

  Example :

```
NOT     BX      ;Complement contents of BX register.

                ;DX =F038h
NOT     DX      ;after  the instruction DX = 0FC7h
```

- **OR** Instruction - OR instruction perform the bit wise logical OR of two operands .Each bit of the result is cleared to 0 if and only if both corresponding bits in each operand are 0, other wise the bit in the result is set to 1.

OR op1, op2

Examples :

```
OR        AH, CL       ;CL ORed with AH, result in AH.
                       ;CX = 00111110  10100101
OR        CX,FF00h     ;OR  CX with immediate FF00h
                       ;result in CX = 11111111 10100101
                       ;Upper byte are all 1's lower bytes
                       ;are unchanged.
```

- **OUT** Instruction - The OUT instruction copies a byte from AL or a word from AX or a double from the accumulator to I/O port specified by op. Two forms of OUT instruction are available : **(1)** Port number is specified by an immediate byte constant, (0 - 255).It is also called as fixed port form. **(2)** Port number is provided in the DX register (0 – 65535)
- **Example**: **(1)**

```
OUT    3BH, AL    ;Copy the contents of the AL to port 3Bh
OUT   2CH,AX      ;Copy the contents of the AX to port 2Ch
                              (2)
MOV    DX, 0FFF8H         ;Load desired port address in DX
OUT     DX, AL            ; Copy the contents of AL to
                          ;FFF8h
OUT      DX, AX           ;Copy content of AX to port
                          ;FFF8H
```

- **POP** Instruction - POP instruction copies the word at the current top of the stack to the operand specified by op then increments the stack pointer to point to the next stack.
- **Example**:

```
POP         DX        ;Copy a word from top of the stack to
                      ; DX and increments SP by 2.
POP         DS        ; Copy a word from top of the stack to
                      ; DS and increments SP by 2.
POP         TABLE [BX]
    ;Copy a word from top of stack to memory in DS with
    ;EA = TABLE + [BX].
```

- **POPF** Instruction - Pop word from top of stack to flag - register.
- **PUSH** Instruction - PUSH source
- **PUSHF** Instruction- Push flag register on the stack
- **RCL** Instruction - Rotate operand around to the left through CF –RCL destination, source.
- **RCR** Instruction - Rotate operand around to the right through CF- RCR destination, count

- **POPF** Instruction - This instruction copies a word from the two memory location at the top of the stack to flag register and increments the stack pointer by 2.
- **PUSH** Instruction - PUSH instruction decrements the stack pointer by 2 and copies a word from a specified source to the location in the stack segment where the stack pointer pointes.

- **Example:**

```
PUSH      BX      ;Decrement SP by 2 and copy BX to stack
PUSH      DS      ;Decrement  SP by 2 and copy DS to stack
PUSH      TABLE[BX]  ;Decrement SP by 2 and copy word
                     ;from memory in DS at
                     ;EA = TABLE + [BX] to stack .
```

- **PUSHF** Instruction - This instruction decrements the SP by 2 and copies the word in flag register to the memory location pointed to by SP.
- **RCL** Instruction - RCL instruction rotates the bits in the operand specified by op1 towards left by the count specified in op2.The operation is circular, the MSB of operand is rotated into a carry flag and the bit in the CF is rotated around into the LSB of operand. **RCR op1, op2**
- **Example:**

```
CLC               ;put 0 in CF
RCL       AX, 1   ;save higher-order bit of AX in CF
RCL       DX, 1   ;save higher-order bit of DX in CF
ADC       AX, 0   ; set lower order bit if needed.
```

- **Example :**

```
RCL       DX, 1  ;Word in DX of 1 bit is moved to left,
and              ;MSB of word is given to CF and
                 ;CF to LSB.
                 ; CF=0, BH = 10110011
RCL       BH, 1  ;Result : BH =01100110
                 ;CF = 1, OF = 1 because MSB changed

                  ;CF =1,AX =00011111  10101001
MOV       CL, 2   ;Load  CL for rotating 2 bit position
RCL       AX, CL ;Result: CF =0, OF undefined
                   ;AX = 01111110  10100110
```

- **RCR** Instruction - RCR instruction rotates the bits in the operand specified by op1 towards right by the count specified in op2. **RCR op1, op2**
- **Example:(1)**

```
RCR      BX, 1  ;Word in BX is rotated by 1 bit towards
                ;right and CF will contain MSB bit and
                ;LSB contain CF bit .


 ( 2)           ;CF = 1, BL = 00111000
RCR      BL, 1 ;Result: BL = 10011100, CF =0
                ;OF = 1  because MSB is changed to 1.
```

- **REP/REPE/REPZ/**
  REPNE/REPNZ - (Prefix) Repeat String
  instruction until specified
  condition exist

- **RET** Instruction – Return execution from
  procedure to calling
  program.

- **ROL** Instruction - Rotate all bits of operand
  left, MSB to LSB
  ROL destination, count.

- **ROL** Instruction - ROL instruction rotates the bıts ın the operand specified by op1 towards left by the count specified in op2. ROL moves each bit in the operand to next higher bit position. The higher order bit is moved to lower order position. Last bit rotated is copied into carry flag.

  ROL op1, op2

- **Example: (1)**

```
ROL     AX, 1  ;Word in AX is moved to left by 1 bit
               ;and MSB bit is to LSB, and CF


               ;CF =0 ,BH =10101110
ROL     BH, 1  ;Result: CF ,Of =1 , BH = 01011101
```

- **Example : (2)**

```
                              ;BX = 01011100  11010011
                              ;CL = 8 bits to rotate
ROL          BH, CL           ;Rotate BX 8 bits towards left
                              ;CF =0, BX =11010011  01011100
```

- **ROR** Instruction - Rotate all bits of operand right, LSB to MSB – ROR destination, count

- **SAHF** Instruction – Copy AH register to low byte of flag register

- **ROR** Instruction - ROR instruction rotates the bits in the operand op1 to wards right by count specified in op2. The last bit rotated is copied into CF. ROR op1, op2
- **Example: (1)**

ROR BL, 1 ;Rotate all bits in BL towards right by 1 bit position, LSB bit is moved to MSB ;and CF has last rotated bit.

(2) ;CF =0, BX = 00111011 01110101

ROR BX, 1 ;Rotate all bits of BX of 1 bit position ;towards right and CF =1,

BX = 10011101 10111010

- **Example (3)**

```
                                ;CF = 0, AL = 10110011,
MOVE        CL, 04H             ; Load  CL
ROR         AL, CL              ;Rotate all bits of AL towards
right                           ;by 4 bits, CF = 0 ,AL = 00111011
```

- **SAHF** Instruction - SAHF copies the value of bits 7, 6, 4, 2, 0 of the AH register into the SF, ZF, AF, PF, and CF respectively. This instruction was provided to make easier conversion of assembly language program written for 8080 and 8085 to 8086.

- **SAL/SHL** Instruction - Shift operand bits left, put zero in LSB(s)
  SAL/AHL destination, count

- **SAR** Instruction - Shift operand bits right, new MAB = old MSB
  SAR destination, count.

- **SBB** Instruction - Subtract with borrow
  SBB destination, source

- **SAL / SHL** Instruction - SAL instruction shifts the bits in the operand specified by op1 to its left by the count specified in op2. As a bit is shifted out of LSB position a 0 is kept in LSB position. CF will contain MSB bit.

  SAL op1,op2

- **Example:**

```
                    ;CF = 0, BX = 11100101  11010011
SAL        BX, 1    ;Shift BX register contents by 1 bit
                    ;position towards left
                    ;CF = 1, BX = 11001011  1010011
```

- **SAR** Instruction - SAR instruction shifts the bits in the operand specified by op1 towards right by count specified in op2.As bit is shifted out a copy of old MSB is taken in MSB MSB position and LSB is shifted to CF. **SAR op1, op2**
- **Example: (1) ; AL = 00011101 = +29 decimal, CF = 0**

```
SAR       AL, 1  ;Shift signed byte in AL towards right
                 ;( divide by 2 )
                 ;AL = 00001110 = + 14 decimal, CF = 1
          ( 2 )
                 ;BH = 11110011 = - 13 decimal, CF = 1
SAR       BH, 1  ;Shifted signed byte in BH  to right
                 ;BH = 11111001 = - 7 decimal, CF = 1
```

- **SBB** Instruction - SUBB instruction subtracts op2 from op1, then subtracts 1 from op1 is CF flag is set and result is stored in op1 and it is used to set the flag.
- **Example:**

```
SUB        CX, BX         ;CX – BX . Result in CX


SUBB       CH, AL         ; Subtract contents of AL and
                          ;contents CF from contents of CH
.                         ;Result in CH


SUBB       AX, 3427H      ;Subtract immediate number
                          ;from AX
```

- **Example:**

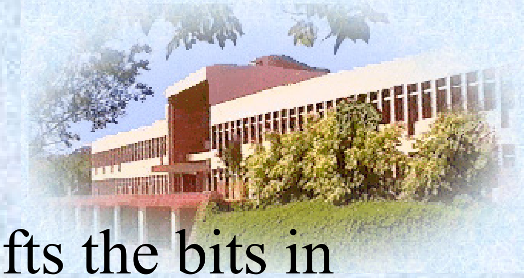
- **Subtracting unsigned number**

```
                        ; CL = 10011100 = 156 decimal
                        ; BH = 00110111 = 55 decimal
SUB        CL, BH       ; CL = 01100101 = 101 decimal
                        ; CF, AF, SF, ZF = 0, OF, PF = 1
```

- **Subtracting signed number**

```
                        ; CL = 00101110 = + 46 decimal
                        ; BH = 01001010= + 74 decimal
SUB        CL, BH       ;CL = 11100100 = -  28 decimal
                        ;CF = 1, AF, ZF =0,
                        ;SF = 1 result negative
```

➢ **STD** Instruction	- Set the direction flag to 1
➢ **STI** Instruction	- Set interrupt flag (IF)
➢ **STOS/STOSB/ STOSW** Instruction	- Store byte or word in string.
➢ **SCAS/SCASB/ SCASW** Instruction	- Scan string byte or a string word.
➢ **SHR** Instruction	- Shift operand bits right, put zero in MSB
➢ **STC** Instruction	- Set the carry flag to 1



- **SHR** Instruction - SHR instruction shifts the bits in op1 to right by the number of times specified by op2 .
- **Example: (1)**

```
SHR        BP, 1   ; Shift word in BP by 1 bit position to
right                        ; and 0 is kept to MSB
           ( 2 )
MOV        CL, 03H ;Load desired number of shifts into
                   ;CL
SHR        BYTE PYR[BX] ;Shift bytes in DS at offset BX
                        ;and rotate 3 bits to right and
                        ;keep 3 0’s in MSB
           ( 3 )   ;SI = 10010011  10101101 , CF = 0
SHR        SI, 1   ; Result: SI = 01001001  11010110
                   ; CF = 1, OF = 1, SF = 0, ZF = 0
```

- **TEST** Instruction – AND operand to update flags
- **WAIT** Instruction - Wait for test signal or interrupt signal
- **XCHG** Instruction - Exchange XCHG destination, source
- **XLAT/**
  XLATB Instruction - Translate a byte in AL
- **XOR** Instruction - Exclusive OR corresponding bits of two operands –
  XOR destination, source

- **TEST** Instruction - This instruction ANDs the contents of a source byte or word with the contents of specified destination word. Flags are updated but neither operand is changed . TEST instruction is often used to set flags before a condition jump instruction
- **Examples:**

```
TEST  AL, BH        ;AND BH with AL. no result is
                    ;stored . Update PF, SF, ZF


TEST  CX, 0001H     ;AND CX with immediate
                    ;number
                    ;no result is stored, Update PF,
                    ;SF
```

- **Example :**

```
                        ;AL = 01010001
TEST       Al, 80H      ;AND immediate 80H with AL to
                        ;test f MSB of AL is 1 or 0
                        ;ZF = 1 if MSB of AL = 0
                        ;AL = 01010001 (unchanged)
                        ;PF = 0 , SF = 0
                        ;ZF = 1 because ANDing produced
                        ;   is 00
```

- **WAIT** Instruction - When this WAIT instruction executes, the 8086 enters an idle condition. This will stay in this state until a signal is asserted on TEST input pin or a valid interrupt signal is received on the INTR or NMI pin.

```
FSTSW     STATUS       ;copy 8087 status word to memory
FWAIT                  ;wait for 8087 to finish before-
                       ; doing next 8086 instruction
MOV       AX,STATUS    ;copy status word to AX to
                       ;check bits
```

- In this code we are adding up of FWAIT instruction so that it will stop the execution of the command until the above instruction is finishes it’s work .so that you are not loosing data and after that you will allow to continue the execution of instructions.

- **XCHG** Instruction - The Exchange instruction exchanges the contents of the register with the contents of another register (or) the contents of the register with the contents of the memory location. Direct memory to memory exchange are not supported.

```
XCHG    op1, op2
```

The both operands must be the same size and one of the operand must always be a register .

Example:

```
XCHG    AX, DX ;Exchange word in AX with word in DX
XCHG    BL, CH  ;Exchange byte in BL with byte  in  CH
XCHG    AL, Money [BX] ;Exchange byte in AL with byte
                                  ;in memory at EA.
```

- **XOR** Instruction - XOR performs a bit wise logical XOR of the operands specified by op1 and op2. The result of the operand is stored in op1 and is used to set the flag.

  XOR op1, op2

  Example : (Numerical)

```
                                ; BX = 00111101 01101001
                                ;CX =  00000000 11111111
XOR          BX, CX             ;Exclusive OR CX with BX
                                ;Result BX = 00111101 10010110
```

Assembler Directives

- **ASSUME**
- **DB** - Defined Byte.
- **DD** - Defined Double Word
- **DQ** - Defined Quad Word
- **DT** - Define Ten Bytes
- **DW** - Define Word

- **ASSUME Directive** - The ASSUME directive is used to tell the assembler that the name of the logical segment should be used for a specified segment. The 8086 works directly with only 4 physical segments: a Code segment, a data segment, a stack segment, and an extra segment.
- **Example:**

  ASUME CS:CODE ;This tells the assembler that the logical segment named CODE contains the instruction statements for the program and should be treated as a code segment.

  ASUME DS:DATA ;This tells the assembler that for any instruction which refers to a data in the data segment, data will found in the logical segment DATA.

- **DB** - DB directive is used to declare a byte-type variable or to store a byte in memory location.
- **Example:**

1. **PRICE DB 49h, 98h, 29h** ;Declare an array of 3 bytes, named as PRICE and initialize.
2. **NAME DB ‘ABCDEF’** ;Declare an array of 6 bytes and initialize with ASCII code for letters
3. **TEMP DB 100 DUP(?)** ;Set 100 bytes of storage in memory and give it the name as TEMP, but leave the 100 bytes uninitialized. Program instructions will load values into these locations.

- **DW** - The DW directive is used to define a variable of type word or to reserve storage location of type word in memory.
- **Example:**

MULTIPLIER DW 437Ah ; this declares a variable of type word and named it as MULTIPLIER. This variable is initialized with the value 437Ah when it is loaded into memory to run.

EXP1 DW 1234h, 3456h, 5678h ; this declares an array of 3 words and initialized with specified values.

STOR1 DW 100 DUP(0); Reserve an array of 100 words of memory and initialize all words with 0000.Array is named as STOR1.

- **END** - END directive is placed after the last statement of a program to tell the assembler that this is the end of the program module. The assembler will ignore any statement after an END directive. Carriage return is required after the END directive.
- **ENDP** - ENDP directive is used along with the name of the procedure to indicate the end of a procedure to the assembler
- **Example:**

  SQUARE_NUM PROCE ; It start the procedure

  ;Some steps to find the square root of a number

  SQUARE_NUM ENDP ;Hear it is the End for the procedure

END	-	End Program
ENDP	-	End Procedure
ENDS	-	End Segment
EQU	-	Equate
EVEN	-	Align on Even Memory Address
EXTRN		

- **ENDS** - This ENDS directive is used with name of the segment to indicate the end of that logic segment.
- **Example:**

```
CODE        SEGMENT   ;Hear it Start the logic
                      ;segment  containing code
; Some instructions statements to perform the logical
;operation
CODE        ENDS      ;End of segment named as
                      ;CODE
```

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

  FACTOR EQU 03H ; you has to write this statement at the starting of your program and later in the program you can use this as follows

  ADD AL, FACTOR ; When it codes this instruction the assembler will code it as ADDAL, 03H

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

- **EVEN** - This **EVEN** directive instructs the assembler to increment the location of the counter to the next even address if it is not already in the even address. If the word is at even address 8086 can read a memory in 1 bus cycle.

  If the word starts at an odd address, the 8086 will take 2 bus cycles to get the data. A series of words can be read much more quickly if they are at even address. When EVEN is used the location counter will simply incremented to next address and NOP instruction is inserted in that incremented location.

- **Example**:

```
DATA1    SEGMENT
; Location counter will point to 0009 after assembler reads
;next statement
SALES  DB  9  DUP(?)    ;declare an array of 9  bytes
EVEN                    ; increment  location counter to 000AH
RECORD  DW  100  DUP( 0 )  ;Array of 100 words will start
         ;from an even address for quicker read
DATA1    ENDS
```

- **GROUP** - Group Related Segments
- **LABLE**
- **NAME**
- **OFFSET**
- **ORG** - Originate

- **GROUP** - The **GROUP** directive is used to group the logical segments named after the directive into one logical group segment.

- **INCLUDE** - This **INCLUDE** directive is used to insert a block of source code from the named file into the current source module.

- **PROC** - Procedure
- **PTR** - Pointer
- **PUBLC**
- **SEGMENT**
- **SHORT**
- **TYPE**

- **PROC** - The **PROC** directive is used to identify the start of a procedure. The term near or far is used to specify the type of the procedure.
- Example:

  SMART **PROC** **FAR** ; This identifies that the start of a procedure named as SMART and instructs the assembler that the procedure is far .

  SMART **ENDP**

  This PROC is used with ENDP to indicate the break of the procedure.

- **PTR** - This **PTR** operator is used to assign a specific type of a variable or to a label.
- Example:

INC [BX] ; This instruction will not know whether to increment the byte pointed to by BX or a word pointed to by BX.

INC BYTE PTR [BX] ;increment the byte
;pointed to by BX

This PTR operator can also be used to override the declared type of variable . If we want to access the a byte in an array **WORDS DW 437Ah, 0B97h,**

MOV AL, BYTE PTR WORDS

- **PUBLIC** - The **PUBLIC** directive is used to instruct the assembler that a specified name or label will be accessed from other modules.
- Example:

  PUBLIC DIVISOR, DIVIDEND ;these two variables are public so these are available to all modules.

  If an instruction in a module refers to a variable in another assembly module, we can access that module by declaring as **EXTRN** directive.

- **TYPE** - **TYPE** operator instructs the assembler to determine the type of a variable and determines the number of bytes specified to that variable.
- **Example:**

Byte type variable – assembler will give a value 1

Word type variable – assembler will give a value 2

Double word type variable – assembler will give a value 4

ADD BX, TYPE WORD_ ARRAY ; hear we want to increment BX to point to next word in an array of words.

DOS Function Calls

➢ **AH 00H**	: Terminate a Program
➢ **AH 01H**	: Read the Keyboard
➢ **AH 02H**	: Write to a Standard Output Device
➢ **AH 08H**	: Read a Standard Input without Echo
➢ **AH 09H**	: Display a Character String
➢ **AH 0AH**	: Buffered keyboard Input
➢ **INT 21H**	: Call DOS Function

Contents

❖ Description of Instructions

❖ Assembly directives

❖ Algorithms with assembly software programs

DATA TRANSFER INSTRUCTIONS

❖GENERAL – PURPOSE BYTE OR WORD TRANSFER INSTRUCTIONS:

- MOV
- PUSH
- POP
- XCHG
- XLAT

❖SIMPLE INPUT AND OUTPUT PORT TRANSFER INSTRUCTION:

- IN
- OUT

❖SPECIAL ADDRESS TRANSFER INSTRUCTION

- LEA
- LDS
- LES

❖FLAG TRANSFER INSTRUCTIONS:

- LAHF
- SAHF
- PUSHF
- POPF

ARITHMETIC INSTRUCTIONS

❖ADITION INSTRUCTIONS:

- ADD
- ADC
- INC
- AAA
- DAA

❖SUBTRACTION INSTRUCTIONS:

➢SUB
➢SBB
➢DEC
➢NEG
➢CMP
➢AAS
➢DAS

❖MULTIPLICATION INSTRUCTIONS:

➢MUL
➢IMUL
➢AAM

❖DIVISION INSTRUCTIONS:

➢DIV
➢IDIV
➢AAD
➢CBW
➢CWD

BIT MANIPULATION INSTRUCTIONS

❖LOGICAL INSTRUCTIONS:

➢NOT
➢AND
➢OR
➢XOR
➢TEST

❖SHIFT INSTRUCTIONS:

➢SHL / SAL
➢SHR
➢SAR

❖RPTATE INSTRUCTIONS:

- ROL
- ROR
- RCL
- RCR

STRING INSTRUCTIONS

- REP
- REPE / REPZ
- REPNE / REPNZ
- MOVS / MOVSB / MOVSW
- COMPS / COMPSB / COMPSW
- SCAS / SCASB / SCASW
- LODS / LODSB / LODSW
- STOS / STOSB / STOSW

PROGRAM EXECUTION TRANSFER INSTRUCTIONS

❖UNCONDITIONAL TRANSFER INSTRUCTIONS:

- CALL
- RET
- JMP

❖CONDITIONAL TRANSFER INSTRUCTIONS:

- JA / JNBE
- JAE / JNB
- JB / JNAE
- JBE / JNA
- JC
- JE / JZ
- JG / JNLE
- JGE / JNL
- JL / JNGE
- JLE / JNG
- JNC
- JNE / JNZ
- JNO
- JNP / JPO
- JNS
- JO

➢JP / JPE
➢JS

❖ITERATION CONTROL INSTRUCTIONS:

➢LOOP
➢LOOPE / LOOPZ
➢LOOPNE / LOOPNZ
➢JCXZ

❖INTERRUPT INSTRUCTIONS:

➢INT
➢INTO
➢IRET

PROCESS CONTROL INSTRUCTIONS

❖FLAG SET / CLEAR INSTRUCTIONS:

➢STC
➢CLC
➢CMC
➢STD
➢CLD
➢STI
➢CLI

❖EXTERNAL HARDWARE SYNCHRONIZATION INSTRUCTIONS:

➢HLT
➢WAIT
➢ESC
➢LOCK
➢NOP

Instruction Description

➢**AAA** Instruction - ASCII Adjust after Addition

➢**AAD** Instruction - ASCII adjust before Division

➢**AAM** Instruction - ASCII adjust after Multiplication

➢**AAS** Instruction - ASCII Adjust for Subtraction

➢**ADC** Instruction - Add with carry.

➢**ADD** Instruction - ADD destination, source

➢**AND** Instruction - AND corresponding bits of two operands

Example

➢**AAA** Instruction - AAA converts the result of the addition of two valid unpacked BCD digits to a valid 2-digit BCD number and takes the AL register as its implicit operand.

Two operands of the addition must have its lower 4 bits contain a number in the range from 0-9.The AAA instruction then adjust AL so that it contains a correct BCD digit. If the addition produce carry (AF=1), the AH register is incremented and the carry CF and auxiliary carry AF flags are set to 1. If the addition did not produce a decimal carry, CF and AF are cleared to 0 and AH is not altered. In both cases the higher 4 bits of AL are cleared to 0.

AAA will adjust the result of the two ASCII characters that were in the range from 30h ("0") to 39h("9").This is because the lower 4 bits of those character fall in the range of 0-9.The result of addition is not a ASCII character but it is a BCD digit.

-

➢**Example:**

```
        MOV  AH,0   ;Clear AH for MSD
        MOV  AL,6   ;BCD 6 in AL
        ADD  AL,5   ;Add BCD 5 to digit in AL
        AAA         ;AH=1, AL=1 representing BCD 11.
```

➢**AAD Instruction** - ADD converts unpacked BCD digits in the AH and AL register into a single binary number in the AX register in preparation for a division operation.

Before executing AAD, place the Most significant BCD digit in the AH register and Last significant in the AL register. When AAD is executed, the two BCD digits are combined into a single binary number by setting AL=(AH*10)+AL and clearing AH to 0.

➢**Example:**

```
        MOV  AX,0205h      ;The unpacked BCD number 25
        AAD                ;After AAD , AH=0 and
                           ;AL=19h (25)
```

After the division AL will then contain the unpacked BCD quotient and AH will contain the unpacked BCD remainder.

➢**Example:**

```
                           ;AX=0607 unpacked BCD for 67 decimal
                           ;CH=09H
```

```
AAD                     ;Adjust to binary before division
                        ;AX=0043 = 43H =67 decimal
DIV  CH                 ;Divide AX by unpacked BCD in CH
                        ;AL = quotient = 07 unpacked BCD
                        ;AH = remainder = 04 unpacked BCD
```

➢**AAM** Instruction - AAM converts the result of the multiplication of two valid unpacked BCD digits into a valid 2-digit unpacked BCD number and takes AX as an implicit operand.

To give a valid result the digits that have been multiplied must be in the range of 0 – 9 and the result should have been placed in the AX register. Because both operands of multiply are required to be 9 or less, the result must be less than 81 and thus is completely contained in AL.

AAM unpacks the result by dividing AX by 10, placing the quotient (MSD) in AH and the remainder (LSD) in AL.

➢**Example:**

```
MOV        AL, 5
MOV        BL, 7
MUL        BL      ;Multiply AL by BL , result in AX
AAM                ;After AAM, AX =0305h (BCD 35)
```

➢**AAS** Instruction - AAS converts the result of the subtraction of two valid unpacked BCD digits to a single valid BCD number and takes the AL register as an implicit operand. The two operands of the subtraction must have its lower 4 bit contain number in the range from 0 to 9 .The AAS instruction then adjust AL so that it contain a correct BCD digit.

```
MOV  AX,0901H      ;BCD 91
SUB  AL, 9         ;Minus 9
AAS                ; Give AX =0802 h (BCD  82)


                   ( a )

                   ;AL =0011 1001 =ASCII   9
                   ;BL=0011 0101 =ASCII   5
SUB  AL, BL        ;(9 - 5) Result :
                   ;AL = 00000100 = BCD 04,CF = 0
AAS                ;Result :
                   ;AL=00000100 =BCD  04
                   ;CF = 0 NO Borrow required
```

(b)

```
                              ;AL = 0011 0101 =ASCII    5
                              ;BL = 0011 1001 = ASCII   9
SUB   AL, BL                  ;( 5 - 9 )  Result :
                              ;AL = 1111 1100 =  - 4
                              ; in 2's complement CF = 1
AAS                           ;Results :
                              ;AL = 0000 0100 =BCD  04
                              ;CF = 1 borrow  needed .
```

➢**ADD** Instruction - These instructions add a number from source to a number from some destination and put the result in the specified destination. The add with carry instruction ADC, also add the status of the carry flag into the result. The source and destination must be of same type , means they must be a byte location or a word location. If you want to add a byte to a word, you must copy the byte to a word location and fill the upper byte of the word with zeroes before adding.

➢**EXAMPLE:**

```
     ADD   AL,74H             ;Add immediate number 74H to
                              ; content of AL

     ADC   CL,BL              ;Add contents of BL plus
                              ;carry  status to contents of CL.
                              ;Results in CL

     ADD   DX, BX             ;Add  contents of BX to contents
                              ;of DX

  ADD      DX, [SI]           ;Add word from memory at
                              ;offset [SI] in DS to contents of DX

                               ; Addition of Un Signed numbers

     ADD    CL,  BL  ;CL = 01110011 =115  decimal
                              ;+ BL = 01001111 = 79 decimal
                              ;Result in CL = 11000010 = 194 decimal

                              ; Addition of Signed numbers

     ADD    CL,  BL  ;CL = 01110011 = + 115  decimal
```

```
                        ;+ BL = 01001111 = +79 decimal
                        ;Result in CL = 11000010 = - 62 decimal
      ; Incorrect because result is too large to fit in 7 bits.
```

➢**AND** Instruction - This Performs a bitwise Logical AND of two operands. The result of the operation is stored in the op1 and used to set the flags.

AND op1, op2

To perform a bitwise AND of the two operands, each bit of the result is set to 1 if and only if the corresponding bit in both of the operands is 1, otherwise the bit in the result I cleared to 0 .

```
AND   BH, CL        ;AND byte in CL with byte in BH
                    ;result in BH
AND   BX,00FFh      ;AND word in BX with immediate
                    ;00FFH. Mask upper byte, leave
                    ;lower unchanged


AND   CX,[SI]       ; AND word at offset [SI] in data
                    ;segment with word in CX
                    ;register . Result in CX register .

                    ;BX = 10110011 01011110
AND   BX,00FFh      ;Mask out upper 8 bits of BX
                    ;Result BX = 00000000  01011110
                    ;CF =0 , OF = 0, PF = 0, SF = 0 ,
                    ;ZF = 0
```

➢**CALL** Instruction

- Direct within-segment (near or intrasegment)
- Indirect within-segment (near or intrasegment)
- Direct to another segment (far or intersegment)
- Indirect to another segment (far or intersegment)

➢**CBW** Instruction - Convert signed Byte to signed word

➢**CLC** Instruction - Clear the carry flag

➢**CLD** Instruction - Clear direction flag
➢**CLI** Instruction - Clear interrupt flag

➢**CMC** Instruction - Complement the carry flag

➢**CMP** Instruction - Compare byte or word -CMP destination, source.

➢**CMPS/CMPSB/**
CMPSW Instruction - Compare string bytes or string words

➢**CWD** Instruction - Convert Signed Word to - Signed Double word

Example

➢**CALL** Instruction - This Instruction is used to transfer execution to a subprogram or procedure. There are two basic types of CALL 's : Near and Far.

A Near CALL is a call to a procedure which is in the same code segment as the CALL instruction .

When 8086 executes the near CALL instruction it decrements the stack pointer by two and copies the offset of the next instruction after the CALL on the stack. This offset saved on the stack is referred as the return address, because this is the address that execution will returns to after the procedure executes. A near CALL instruction will also load the instruction pointer with the offset of the first instruction in the procedure.

A RET instruction at the end of the procedure will return execution to the instruction after the CALL by coping the offset saved on the stack back to IP.

A Far CALL is a call to a procedure which is in a different from that which contains the CALL instruction . When 8086 executes the Far CALL instruction it decrements the stack pointer by two again and copies the content of CS register to the stack. It then decrements the stack pointer by two again and copies the offset contents offset of the instruction after the CALL to the stack. Finally it loads CS with segment base of the segment which contains the procedure and IP with the offset of the first instruction of the procedure in segment. A RET instruction at end of procedure will return to the next instruction after the CALL by restoring the saved CS and IP from the stack.

;Direct within-segment (near or intrasegment)

CALL MULTO ;MULTO is the name of the procedure. The assembler determines displacement of MULTO from the instruction after the CALL and codes this displacement in as part of the instruction .

;Indirect within-segment (near or intrasegment)

CALL BX ; BX contains the offset of the first instruction of the procedure .Replaces contents of word of IP with contents o register BX.

CALL WORD PTR[BX] ;Offset of first instruction of procedure is in two memory addresses in DS .Replaces contents of IP with contents of word memory location in DS pointed to by BX.

;Direct to another segment- far or intersegment.

CALL SMART ;SMART is the name of the Procedure

SMART PROC FAR ; Procedure must be declare as an far

➢**CBW** Instruction - CBW converts the signed value in the AL register into an equivalent 16 bit signed value in the AX register by duplicating the sign bit to the left.

This instruction copies the sign of a byte in AL to all the bits in AH. AH is then said to be the sign extension of AL.

Example:

```
        ; AX =   00000000  10011011 = - 155 decimal
CBW     ; Convert signed byte in AL to signed word in AX.
        ; Result in AX = 11111111  10011011
        ; = - 155 decimal
```

➢**CLC** Instruction - CLC clear the carry flag (CF) to 0 This instruction has no affect on the processor, registers, or other flags. It is often used to clear the CF before returning from a procedure to indicate a successful termination. It is also use to clear the CF during rotate operation involving the CF such as ADC, RCL, RCR .

Example:
CLC ;Clear carry flag.

➢**CLD** Instruction - This instruction reset the designation flag to zero. This instruction has no effect on the registers or other flags. When the direction flag is cleared / reset SI and DI will automatically be incremented when one of the string instruction such as MOVS, CMPS, SCAS,MOVSB and STOSB executes.

Example :

CLD ;Clear direction flag so that string pointers auto increment

➢**CLI** Instruction - This instruction resets the interrupt flag to zero. No other flags are affected. If the interrupt flag is reset , the 8086 will not respond to an interrupt signal on its INTR input. This CLI instruction has no effect on the nonmaskable interrupt input, NMI

➢**CMC** Instruction - If the carry flag CF is a zero before this instruction, it will be set to a one after the instruction. If the carry flag is one before this instruction, it will be reset to a zero after the instruction executes. CMC has no effect on other flags.

Example:

CMC ;Invert the carry flag.

➢**CWD** Instruction - CWD converts the 16 bit signed value in the AX register into an equivalent 32 bit signed value in DX: AX register pair by duplicating the sign bit to the left.

The CWD instruction sets all the bits in the DX register to the same sign bit of the AX register. The effect is to create a 32- bit signed result that has same integer value as the original 16 bit operand.

Example:

Assume AX contains C435h. If the CWD instruction is executed, DX will contain FFFFh since bit 15 (MSB) of AX was 1. Both the original value of AX (C435h) and resulting value of DX : AX (FFFFC435h) represents the same signed number.

Example:

```
                ;DX = 00000000 00000000
                ;AX = 11110000 11000111 = - 3897 decimal
CWD             ;Convert signed word in AX to signed double
                ;word in DX:AX
                ;Result  DX = 11111111 11111111
                ;AX = 11110000 11000111 = -3897 decimal .
```

➢**DAA** Instruction - Decimal Adjust Accumulator

➢**DAS** Instruction - Decimal Adjust after Subtraction

➢**DEC** Instruction - Decrement destination register or memory DEC destination.

➢**DIV** Instruction - Unsigned divide-Div source

➢**ESC** Instruction

When a double word is divided by a word, the most significant word of the double word must be in DX and the least significant word of the double word must be in AX. After the division AX will contain the 16 –bit result (quotient) and DX will contain a 16 bit remainder. Again , if an attempt is made to divide by zero or quotient is too large to fit in AX (greater than FFFFH) the 8086 will do a type of 0 interrupt .

Example:

```
        DIV    CX      ; (Quotient) AX= (DX:AX)/CX
                       : (Reminder) DX=(DX:AX)%CX
```

For DIV the dividend must always be in AX or DX and AX, but the source of the divisor can be a register or a memory location specified by one of the 24 addressing modes.

If you want to divide a byte by a byte, you must first put the dividend byte in AL and fill AH with all 0's . The SUB AH,AH instruction is a quick way to do.

If you want to divide a word by a word, put the dividend word in AX and fill DX with all 0's. The SUB DX,DX instruction does this quickly.

➢**Example:**

```
                       ; AX = 37D7H = 14, 295 decimal
                       ; BH = 97H = 151 decimal
        DIV  BH        ;AX / BH
                       ; AX = Quotient = 5EH = 94 decimal
                       ; AH = Remainder = 65H = 101 decimal
```

➢**ESC** Instruction - Escape instruction is used to pass instruction to a coprocessor such as the 8087 math coprocessor which shares the address and data bus with an 8086. Instruction for the coprocessor are represented by a 6 bit code embedded in the escape instruction. As the 8086 fetches instruction byte, the coprocessor also catches these bytes from data bus and puts them in its queue. The coprocessor treats all of the 8086 instruction as an NOP. When 8086 fetches an ESC instruction , the coprocessor decodes the instruction and carries out the action specified by the 6 bit code. In most of the case 8086 treats ESC instruction as an NOP.

➢**HLT** Instruction - HALT processing

➢**IDIV** Instruction - Divide by signed byte or word IDIV source

➢**IMUL** Instruction - Multiply signed number-IMUL source

➢**IN** Instruction - Copy data from a port

IN accumulator, port

➢**INC** Instruction - Increment - INC destination

➢**HALT** Instruction - 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 .

➢**IDIV** Instruction - This instruction is used to divide a signed word by a signed byte or to divide a signed double word by a signed word.

➢**Example:**

```
IDIV   BL      ;Signed word in AX is divided by signed   byte in BL
```

➢**IMUL** Instruction - This instruction performs a signed multiplication.

IMUL op ;In this form the accumulator is the multiplicand and op is the multiplier. op may be a register or a memory operand.

IMUL op1, op2 ;In this form op1 is always be a register operand and op2 may be a register or a memory operand.

➢**Example:**

```
IMUL BH      ;Signed byte in AL times multiplied by
             ;signed byte in BH and result in AX .
```

➢**Example:**

```
                         ; 69 * 14
                         ; AL = 01000101 = 69 decimal
                         ; BL = 00001110 = 14 decimal
IMUL BL                  ;AX = 03C6H =  + 966 decimal
                         ;MSB = 0 because positive result

                         ; - 28 * 59
                         ; AL = 11100100 = - 28 decimal
                         ;BL = 00001110 =  14 decimal
IMUL BL                  ;AX = F98Ch = - 1652 decimal
                         ; MSB = 1 because negative result
```

➢**IN** Instruction - This IN instruction will copy data from a port to the AL or AX register.

For the Fixed port IN instruction type the 8 – bit port address of a port is specified directly in the instruction.

➢**Example:**

```
IN     AL,0C8H     ;Input a byte from port 0C8H to AL
IN     AX, 34H     ;Input a word from port 34H to AX

A_TO_D      EQU   4AH
```

```
        IN      AX, A_TO_D ;Input a word from port 4AH to AX
```

For a variable port IN instruction, the port address is loaded in DX register before IN instruction. DX is 16 bit. Port address range from 0000H – FFFFH.

➢**Example:**

```
        MOV  DX, 0FF78H           ;Initialize DX point to port
        IN      AL, DX                ;Input a byte from a 8 bit port
                                      ;0FF78H to AL

        IN      AX, DX                ;Input a word from 16 bit port to
                                      ;0FF78H  to AX.
```

➢**INC** Instruction - INC instruction adds one to the operand and sets the flag according to the result. INC instruction is treated as an unsigned binary number.

➢**Example:**

```
                          ; AX = 7FFFh
                INC  AX   ;After this instruction AX = 8000h

                INC   BL  ; Add 1 to the contents of BL register
                INC   CL  ; Add 1 to the contents of CX register.
```

➢**INT** Instruction - Interrupt program

➢**INTO** Instruction - Interrupt on overflow.

➢**IRET** Instruction - Interrupt return

➢**JA/JNBE** Instruction - Jump if above/Jump if not below nor equal.

➢**JAE/JNB/**
JNC Instructions - Jump if above or equal/ Jump if not below/ Jump if no carry.

➢**JA / JNBE** - This instruction performs the Jump if above (or) Jump if not below or equal operations according to the condition, if CF and ZF = 0 .

➢**Example:** (1)

```
                CMP   AX, 4371H     ;Compare by subtracting 4371H
        ;from AX
```

```
          JA      RUN_PRESS ;Jump to label RUN_PRESS if
     ;AX  above 4371H
                  ( 2 )
          CMP     AX, 4371H     ;Compare ( AX – 4371H)
          JNBE    RUN_PRESS ;Jump to label RUN_PRESS if
     ;AX not below or equal to 4371H
```

➢**JAE / JNB / JNC** - This instructions performs the Jump if above or equal, Jump if not below, Jump if no carry operations according to the condition, if CF = 0.

➢**Examples**:

```
1.   CMP   AX, 4371H    ;Compare ( AX – 4371H)
     JAE   RUN          ;Jump to the label RUN if AX is
                        ;above or equal to 4371H .
2.   CMP   AX, 4371H    ;Compare ( AX – 4371H)
     JNB   RUN_1        ;Jump to the label RUN_1 if AX
                        ;is not below than 4371H
3.   ADD   AL, BL       ; Add AL, BL. If result is with in JNC OK
                        ;acceptable range, continue
```

➢**JB/JC/JNAE** Instruction - Jump if below/Jump if carry/ Jump if not above nor equal

➢**JBE/JNA** Instructions - Jump if below or equal / Jump if not above

➢**JCXZ** Instruction - Jump if the CX register is zero

➢**JE/JZ** Instruction - Jump if equal/Jump if zero

➢**JG/JNLE** Instruction - Jump if greater/Jump if not less than nor equal

➢**JB/JC/JNAE** Instruction - This instruction performs the Jump if below (or) Jump if carry (or) Jump if not below/ equal operations according to the condition, if CF = 1

➢**Example:**

```
1.CMP      AX, 4371H    ;Compare ( AX – 4371H )
     JB    RUN_P        ;Jump to label RUN_P if AX is
                        ;below 4371H

2.ADD BX, CX            ;Add two words and Jump to
```

```
JC      ERROR         ; label ERROR if CF = 1
```

➢**JBE/JNA** Instruction - This instruction performs the Jump if below or equal (or) Jump if not above operations according to the condition, if CF and ZF = 1

➢**Example:**

```
CMP   AX, 4371H     ;Compare ( AX – 4371H )
JBA   RUN           ;Jump to label RUN if AX is
                    ;below or equal to 4371H

CMP   AX, 4371H     ;Compare ( AX – 4371H )
JNA   RUN_R         ;Jump to label RUN_R if AX is
                    ;not above  than 4371H
```

➢**JCXZ** Instruction - This instruction performs the Jump if CX register is zero. If CX does not contain all zeros, execution will simply proceed to the next instruction.

➢**Example:**

```
    JCXZ      SKIP_LOOP;If CX = 0, skip the process
 NXT: SUB     [BX], 07H    ;Subtract 7 from data value
      INC     BX           ; BX point to next value
    LOOP      NXT          ; Loop until CX = 0
      SKIP_LOOP            ;Next instruction
```

➢**JE/JZ** Instruction Instruction - This instruction performs the Jump if equal (or) Jump if zero operations according to the condition if ZF = 1

➢**Example:**

```
NXT:CMP          BX, DX ;Compare ( BX – DX )
      JE         DONE  ;Jump to DONE if BX = DX,
      SUB        BX, AX ;Else subtract Ax
      INC        CX      ;Increment counter
      JUMP       NXT     ;Check again
    DONE: MOVAX, CX  ;Copy count to AX
```

➢**Example:**

```
IN     AL, 8FH       ;read data from port 8FH
SUB    AL, 30H       ;Subtract minimum value
```

```
JZ      STATR          ; Jump to label if result of
                       ;subtraction  was 0
```

➢**JG/JNLE** Instruction - This instruction performs the Jump if greater (or) Jump if not less than or equal operations according to the condition if ZF =0 and SF = OF

➢**Example:**

```
CMP        BL, 39H     ;Compare by subtracting
                       ;39H from BL
JG         NEXT1       ;Jump to label if BL is
                       ;more positive than 39H

CMP        BL, 39H     ;Compare by subtracting
                               ;39H from BL
JNLE       NEXT2       ;Jump to label if BL is not
                       ;less than or equal 39H
```

➢**JGE/JNL** Instruction - Jump if greater than or equal/ Jump if not less than

➢**JL/JNGE** Instruction - Jump if less than/Jump if not greater than or equal

➢**JLE/JNG** Instruction - Jump if less than or equal/ Jump if not greater

➢**JMP** Instruction - Unconditional jump to - specified destination

➢**JGE/JNL** Instruction - This instruction performs the Jump if greater than or equal / Jump if not less than operation according to the condition if SF = OF

➢**Example:**

```
CMP        BL, 39H     ;Compare by the
                       ;subtracting 39H from BL
JGE        NEXT11      ;Jump to label if BL is
                       ;more positive than 39H
                       ; or equal to 39H
```

```
CMP      BL, 39H      ;Compare by subtracting
                      ;39H from BL
JNL      NEXT22       ;Jump to label if BL is not
                      ;less than 39H
```

➢**JL/JNGE** Instruction - This instruction performs the Jump if less than / Jump if not greater than or equal operation according to the condition, if SF ≠ OF

➢**Example:**

```
CMP   BL, 39H      ;Compare by subtracting 39H
                   ;from BL
JL    AGAIN        ;Jump to the label if BL is more
                      ;negative than 39H

CMP   BL, 39H      ;Compare by subtracting 39H
                   ;from BL
JNGE  AGAIN1       ; Jump to the label if BL is  not
                   ;more positive than 39H or
                   ;not equal to 39H
```

➢**JLE/JNG** Instruction - This instruction performs the Jump if less than or equal / Jump if not greater operation according to the condition, if ZF=1 and SF ≠ OF

➢**Example:**

```
CMP   BL, 39h      ; Compare by subtracting 39h
                   ;from BL
JLE          NXT1  ;Jump to the label if BL is more
                   ;negative than 39h or equal to 39h

CMP   BL, 39h      ;Compare by subtracting 39h
                   ;from BL
JNG   AGAIN2       ; Jump to the label if BL is  not
                   ;more positive than 39h
```

➢**JNA/JBE** Instruction - Jump if not above/Jump if below or equal

➢**JNAE/JB** Instruction - Jump if not above or equal/ Jump if below

➢**JNB/JNC/JAE** Instruction - Jump if not below/Jump if no carry/Jump if above or equal

➢**JNE/JNZ** Instruction - Jump if not equal/Jump if not zero

➢**JNE/JNZ** Instruction - This instruction performs the Jump if not equal / Jump if not zero operation according to the condition, if ZF=0

➢**Example:**

```
NXT:  IN    AL, 0F8H         ;Read data value from port
            CMP  AL, 72      ;Compare ( AL – 72 )
            JNE  NXT         ;Jump to NXT if AL ≠ 72
            IN   AL, 0F9H    ;Read next port when AL = 72
            MOV BX, 2734H    ; Load BX as counter
     NXT_1:ADD  AX, 0002H    ;Add count factor to AX
              DEC      BX         ;Decrement BX
              JNZ      NXT_1      Repeat until BX = 0
```

➢**JNG/JLE** Instruction - Jump if not greater/ Jump if less than or equal

➢**JNGE/JL** Instruction - Jump if not greater than nor equal/Jump if less than

➢**JNL/JGE** Instruction - Jump if not less than/ Jump if greater than or equal

➢**JNLE/JG** Instruction - Jump if not less than nor equal to /Jump if greater than

➢**JNO** Instruction – Jump if no overflow

➢**JNP/JPO** Instruction – Jump if no parity/ Jump if parity odd

➢**JNS** Instruction - Jump if not signed (Jump if positive)

➢**JNZ/JNE** Instruction - Jump if not zero / jump if not equal

➢**JO** Instruction - Jump if overflow

➢**JNO** Instruction – This instruction performs the Jump if no overflow operation according to the condition, if OF=0

➢**Example:**

```
            ADD   AL, BL        ; Add signed bytes in AL and BL
            JNO   DONE          ;Process done if no overflow -
            MOV   AL, 00H       ;Else load error code in AL
     DONE: OUT   24H, AL        ; Send result to display
```

➢**JNP/JPO** Instruction – This instruction performs the Jump if not parity / Jump if parity odd operation according to the condition, if PF=0

➢**Example:**

```
            IN    AL, 0F8H      ;Read ASCII char from UART
            OR    AL, AL        ;Set flags
            JPO   ERROR1        ;If even parity executed, if not
                                ;send error message
```

➢**JNS** Instruction - This instruction performs the Jump if not signed (Jump if positive) operation according to the condition, if SF=0

➢**Example:**

```
     DEC   AL      ;Decrement  counter
     JNS   REDO; Jump to label REDO if counter has not
                ;decremented to FFH
```

➢**JO** Instruction - This instruction performs Jump if overflow operation according to the condition OF = 0

➢**Example:**

```
     ADD   AL, BL        ;Add signed bits in AL and BL
     JO    ERROR         ; Jump to label if overflow occur
                         ;in addition
     MOV   SUM, AL       ; else put the result in memory
                         ;location named SUM
```

➢**JPE/JP** Instruction - Jump if parity even/ Jump if parity

➢**JPO/JNP** Instruction - Jump if parity odd/ Jump if no parity

➢**JS** Instruction - Jump if signed (Jump if negative)

➢**JZ/JE** Instruction - Jump if zero/Jump if equal

➢**JPE/JP** Instruction - This instruction performs the Jump if parity even / Jump if parity operation according to the condition, if PF=1

Example:

```
IN      AL, 0F8H    ;Read ASCII char from UART
OR      AL, AL      ;Set flags
JPE     ERROR2      ;odd parity is expected, if not
                    ;send error message
```

➢**JS** Instruction **-** This instruction performs the Jump if sign operation according to the condition, if SF=1

➢**Example:**

```
ADD     BL, DH      ;Add signed bytes DH to BL
JS      JJS_S1      ;Jump to label if result is
                    ;negative
```

➢**LAHF** Instruction - Copy low byte of flag register to AH

➢**LDS** Instruction - Load register and Ds with words from memory – LDS register, memory address of first word

➢**LEA** Instruction - Load effective address-LEA register, source

➢**LES** Instruction - Load register and ES with words from memory –LES register, memory address of first word.

➢**LAHF** Instruction - LAHF instruction copies the value of SF, ZF, AF, PF, CF, into bits of 7, 6, 4, 2, 0 respectively of AH register. This LAHF instruction was provided to make conversion of assembly language programs written for 8080 and 8085 to 8086 easier.

➢**LDS** Instruction - This instruction loads a far pointer from the memory address specified by op2 into the DS segment register and the op1 to the register.

LDS op1, op2

➢**Example:**

LDS BX, [4326] ; copy the contents of the memory at displacement 4326H in DS to BL, contents of the 4327H to BH. Copy contents of 4328H and 4329H in DS to DS register.

➢**LEA** Instruction - This instruction indicates the offset of the variable or memory location named as the source and put this offset in the indicated 16 – bit register.

➢**Example:**

```
LEA   BX, PRICE   ;Load BX with offset of PRICE
                  ;in DS
LEA   BP, SS:STAK;Load BP with offset of STACK
                  ;in SS
LEA   CX, [BX][DI] ;Load CX with EA=BX + DI
```

➢**LOCK** Instruction - Assert bus lock signal

➢**LODS/LODSB/**
LODSW Instruction - Load string byte into AL or
Load string word into AX.

➢**LOOP** Instruction - Loop to specified
label until CX = 0

➢**LOOPE /**
LOOPZ Instruction - loop while CX ≠ 0 and
ZF = 1

➢**LODS/LODSB/LODSW** Instruction - This instruction copies a byte from a string location pointed to by SI to AL or a word from a string location pointed to by SI to AX. If DF is cleared to 0,SI will automatically incremented to point to the next element of string.

➢**Example:**

```
CLD   ;Clear direction flag so SI is auto incremented

MOV  SI, OFFSET SOURCE_STRING    ;point SI at   start of the string

LODS SOUCE_STRING    ;Copy byte or word from
                     ;string to AL or AX
```

➢**LOOP** Instruction - This instruction is used to repeat a series of instruction some number of times

➢**Example:**

```
        MOV  BX, OFFSET PRICE
                             ;Point BX at first element in array
        MOV  CX, 40           ;Load CX with number of
                              ;elements in array
    NEXT: MOV AL, [BX]        ; Get elements from array
        ADD   AL, 07H         ;Ad correction factor
        DAA                   ; decimal adjust result
        MOV  [BX], AL         ; Put result back in array
        LOOP NEXT             ; Repeat until all elements
                              ;adjusted.
```

➢**LOOPE / LOOPZ** Instruction - This instruction is used to repeat a group of instruction some number of times until CX = 0 and ZF = 0

➢**Example:**

```
        MOV  BX, OFFSET   ARRAY
                              ;point BX at start of the array
        DEC   BX
        MOV  CX, 100          ;put number of array elements in
                              ;CX
        NEXT:INC    BX        ;point to next element in array
        CMP   [BX], 0FFH      ;Compare array elements FFH
        LOOP  NEXT
```

➢**LOOPNE/LOOPNZ** Instruction - This instruction is used to repeat a group of instruction some number of times until CX = 0 and ZF = 1

➢**Example:**

```
        MOV  BX, OFFSET   ARRAY1
                              ;point BX at start of the array
        DEC   BX
        MOV  CX, 100          ;put number of array elements in
                              ;CX
    NEXT:INC   BX             ;point  to next elements in array
        CMP   [BX], 0FFH      ;Compare array elements 0DH
        LOOPNE  NEXT
```

➢**MOV** Instruction - MOV destination, source

➢**MOVS/MOVSB/**
MOVSW Instruction - Move string byte or string word-MOVS destination, source

➢**MUL** Instruction - Multiply unsigned bytes or words-MUL source

➢**NEG** Instruction - From 2's complement – NEG destination

➢**NOP** Instruction - Performs no operation.

➢**MOV** Instruction - The MOV instruction copies a word or a byte of data from a specified source to a specified destination .

```
          MOV   op1, op2
```

➢**Example:**

```
MOV  CX, 037AH    ; MOV 037AH into the CX.
MOV  AX, BX       ;Copy the contents of register BX
                  ;to AX
MOV  DL,[BX]      ;Copy byte from memory at BX
                  ;to  DL , BX contains the offset of byte  in DS.
```

➢**MUL** Instruction - This instruction multiplies an unsigned multiplication of the accumulator by the operand specified by op. The size of op may be a register or memory operand .

```
               MUL  op
```

```
Example:          ;AL = 21h (33 decimal)
                  ;BL = A1h(161 decimal )
     MUL  BL      ;AX =14C1h (5313 decimal) since AH≠0,
                  ;CF and OF will set to 1.
     MUL  BH      ; AL times BH, result in AX
     MUL  CX      ;AX times CX, result high word in DX,
                  ;low word in AX.
```

➢**NEG** Instruction - NEG performs the two's complement subtraction of the operand from zero and sets the flags according to the result.

```
               ;AX = 2CBh
     NEG  AX      ;after executing NEG result AX =FD35h.
```

Example:

```
NEG   AL     ;Replace number in AL with its 2's complement
NEG   BX     ;Replace word in BX with its 2's    complement
NEG   BYTE PTR[BX]; Replace byte at offset BX in
                  ; DS with its 2's complement
```

➢**NOP** Instruction - This instruction simply uses up the three clock cycles and increments the instruction pointer to point to the next instruction. NOP does not change the status of any flag. The NOP instruction is used to increase the delay of a delay loop.

➢**NOT** Instruction - Invert each bit of operand –NOT destination.

➢**OR** Instruction - Logically OR corresponding of two operands- OR destination, source.

➢**OUT** Instruction - Output a byte or word to a port – OUT port, accumulator AL or AX.

➢**POP** Instruction - POP destination

➢**NOT** Instruction - NOT perform the bitwise complement of op and stores the result back into op.

NOT op

Example :

```
NOT   BX    ;Complement contents of BX register.

            ;DX =F038h
NOT   DX    ;after  the instruction DX = 0FC7h
```

➢**OR** Instruction - OR instruction perform the bit wise logical OR of two operands .Each bit of the result is cleared to 0 if and only if both corresponding bits in each operand are 0, other wise the bit in the result is set to 1.

OR op1, op2

Examples :

```
OR          AH, CL      ;CL ORed with AH, result in AH.
                        ;CX = 00111110  10100101
OR          CX,FF00h    ;OR  CX with immediate FF00h
                        ;result in CX = 11111111 10100101
                        ;Upper byte are all 1's lower bytes
                        ;are unchanged.
```

➢**OUT** Instruction - The OUT instruction copies a byte from AL or a word from AX or a double from the accumulator to I/O port specified by op. Two forms of OUT instruction are available : **(1)** Port number is specified by an immediate byte constant, (0

- 255).It is also called as fixed port form. **(2)** Port number is provided in the DX register (0 – 65535)

➢**Example**: **(1)**

```
OUT    3BH, AL      ;Copy the contents of the AL to port 3Bh
OUT   2CH,AX        ;Copy the contents of the AX to port 2Ch

                            (2)
MOV    DX, 0FFF8H;Load desired port address in DX
OUT     DX, AL              ; Copy the contents of AL to
                            ;FFF8h
OUT      DX, AX             ;Copy content of AX to port
                            ;FFF8H
```

➢**POP** Instruction - POP instruction copies the word at the current top of the stack to the operand specified by op then increments the stack pointer to point to the next stack.

➢**Example**:

```
POP    DX     ;Copy a word from top of the stack to
              ; DX and increments SP by 2.
POP    DS     ; Copy a word from top of the stack to
              ; DS and increments SP by 2.

POP    TABLE [BX]
              ;Copy a word from top of stack to memory in DS with
              ;EA = TABLE + [BX].
```

➢**POPF** Instruction - Pop word from top of stack to flag - register.

➢**PUSH** Instruction - PUSH source

➢**PUSHF** Instruction - Push flag register on the stack

➢**RCL** Instruction - Rotate operand around to the left through CF – RCL destination, source.

➢ **RCR** Instruction - Rotate operand around to the right through CF- RCR destination, count

➢**POPF** Instruction - This instruction copies a word from the two memory location at the top of the stack to flag register and increments the stack pointer by 2.

➢**PUSH** Instruction - PUSH instruction decrements the stack pointer by 2 and copies a word from a specified source to the location in the stack segment where the stack pointer pointes.

➢**Example:**

```
PUSH BX      ;Decrement SP by 2 and copy BX to stack
PUSH DS      ;Decrement  SP by 2 and copy DS to stack
PUSH TABLE[BX]  ;Decrement SP by 2 and copy word
             ;from memory in DS at
             ;EA = TABLE + [BX] to stack .
```

➢**PUSHF** Instruction - This instruction decrements the SP by 2 and copies the word in flag register to the memory location pointed to by SP.

➢**RCL** Instruction - RCL instruction rotates the bits in the operand specified by op1 towards left by the count specified in op2.The operation is circular, the MSB of operand is rotated into a carry flag and the bit in the CF is rotated around into the LSB of operand. **RCR op1, op2**

➢**Example:**

```
CLC          ;put 0 in CF
RCL   AX, 1  ;save higher-order bit of AX in CF
RCL   DX, 1  ;save higher-order bit of DX in CF
ADC   AX, 0  ; set lower order bit if needed.
```

➢**Example :**

```
RCL   DX, 1          ;Word in DX of 1 bit is moved to left, and
                     ;MSB of word is given to CF and
                     ;CF to LSB.
                     ; CF=0, BH = 10110011
RCL   BH, 1          ;Result : BH =01100110
                     ;CF = 1, OF = 1 because MSB changed

                      ;CF =1,AX =00011111  10101001
MOV  CL, 2           ;Load  CL for rotating 2 bit position
RCL   AX, CL         ;Result: CF =0, OF undefined
                       ;AX = 01111110  10100110
```

➢**RCR** Instruction - RCR instruction rotates the bits in the operand specified by op1 towards right by the count specified in op2. **RCR op1, op2**

➢**Example:(1)**

```
        RCR   BX, 1  ;Word in BX is rotated by 1 bit towards
                     ;right and CF will contain MSB bit and
                     ;LSB contain CF bit .
```

(2)

```
                     ;CF = 1, BL = 00111000
        RCR   BL, 1  ;Result: BL = 10011100, CF =0
                     ;OF = 1  because MSB is changed to 1.
```

➢**REP/REPE/REPZ/**
REPNE/REPNZ **-** (Prefix) Repeat String instruction until specified condition exist

➢**RET** Instruction – Return execution from procedure to calling program.

➢

➢**ROL** Instruction - Rotate all bits of operand left, MSB to LSB
ROL destination, count.

➢**ROL** Instruction - ROL instruction rotates the bits in the operand specified by op1 towards left by the count specified in op2. ROL moves each bit in the operand to next higher bit position. The higher order bit is moved to lower order position. Last bit rotated is copied into carry flag.

```
              ROL   op1, op2
```

➢**Example: (1)**

```
        ROL   AX, 1  ;Word in AX is moved to left by 1 bit
                     ;and MSB bit is to LSB, and CF

                     ;CF =0 ,BH =10101110
        ROL   BH, 1  ;Result: CF ,Of =1 , BH = 01011101
```

➢**Example : (2)**

```
                                ;BX = 01011100  11010011
                                ;CL = 8 bits to rotate
        ROL   BH, CL            ;Rotate BX 8 bits towards left
                                ;CF =0, BX =11010011  01011100
```

➢**ROR** Instruction - Rotate all bits of operand right, LSB to MSB – ROR destination, count

➢**SAHF** Instruction – Copy AH register to low byte of flag register

➢**ROR** Instruction - ROR instruction rotates the bits in the operand op1 to wards right by count specified in op2. The last bit rotated is copied into CF.

ROR op1, op2

➢**Example:**

(1)

```
ROR   BL, 1           ;Rotate all bits in BL towards right by 1
                      ;bit position, LSB bit is moved to MSB
                      ;and CF has last rotated bit.
```

(2)

```
                      ;CF =0, BX = 00111011  01110101
ROR   BX, 1           ;Rotate all bits of BX of 1 bit position
                      ;towards right and CF =1,
BX = 10011101  10111010
```

➢**Example** **(3)**

```
                           ;CF = 0, AL = 10110011,
MOVE       CL, 04H         ; Load  CL
ROR        AL, CL          ;Rotate all bits of AL towards right
                           ;by 4 bits, CF = 0 ,AL = 00111011
```

➢**SAHF** Instruction - SAHF copies the value of bits 7, 6, 4, 2, 0 of the AH register into the SF, ZF, AF, PF, and CF respectively. This instruction was provided to make easier conversion of assembly language program written for 8080 and 8085 to 8086.

➢**SAL/SHL** Instruction - Shift operand bits left, put zero in LSB(s)
SAL/AHL destination, count

➢**SAR** Instruction - Shift operand bits right, new MAB = old MSB
SAR destination, count.

➢**SBB** Instruction - Subtract with borrow SBB destination, source

➢**SAL / SHL** Instruction - SAL instruction shifts the bits in the operand specified by op1 to its left by the count specified in op2. As a bit is shifted out of LSB position a 0 is kept in LSB position. CF will contain MSB bit.

SAL op1,op2

➢**Example:**

```
                    ;CF = 0, BX = 11100101  11010011
SAL   BX, 1         ;Shift BX register contents by 1 bit
                    ;position towards left
                    ;CF = 1, BX = 11001011  1010011
```

➢**SAR** Instruction - SAR instruction shifts the bits in the operand specified by op1 towards right by count specified in op2.As bit is shifted out a copy of old MSB is taken in MSB

MSB position and LSB is shifted to CF.

SAR op1, op2

➢**Example: (1)**

```
                ; AL = 00011101 = +29 decimal, CF = 0
SAR   AL, 1     ;Shift signed byte in AL towards right
                ;( divide by 2 )
                ;AL = 00001110 = + 14 decimal, CF = 1
```

(2)

```
                    ;BH = 11110011 = - 13 decimal, CF = 1
SAR   BH, 1     ;Shifted signed byte in BH  to right
                    ;BH = 11111001 = - 7 decimal, CF = 1
```

➢**SBB** Instruction - SUBB instruction subtracts op2 from op1, then subtracts 1 from op1 is CF flag is set and result is stored in op1 and it is used to set the flag.

➢**Example:**

```
SUB   CX, BX        ;CX – BX . Result in CX

SUBB  CH, AL        ; Subtract contents of AL and
                    ;contents CF from contents of CH .
                    ;Result in CH

SUBB  AX, 3427H     ;Subtract immediate number
                    ;from AX
```

➢**Example:**

•Subtracting unsigned number

```
                                  ; CL = 10011100 = 156 decimal
                                  ; BH = 00110111 = 55 decimal
        SUB    CL, BH             ; CL = 01100101 = 101 decimal
                                  ; CF, AF, SF, ZF = 0, OF, PF = 1
```

•Subtracting signed number

```
                                  ; CL = 00101110 = + 46 decimal
                                  ; BH = 01001010= + 74 decimal
        SUB    CL, BH             ;CL = 11100100 = -  28 decimal
                                  ;CF = 1, AF, ZF =0,
                                  ;SF = 1 result negative
```

➢**STD** Instruction - Set the direction flag to 1

➢**STI** Instruction - Set interrupt flag (IF)

➢**STOS/STOSB/**
STOSW Instruction - Store byte or word in string.

➢**SCAS/SCASB/**
SCASW Instruction - Scan string byte or a string word.

➢**SHR** Instruction - Shift operand bits right, put zero in MSB

➢**STC** Instruction - Set the carry flag to 1

➢**SHR** Instruction - SHR instruction shifts the bits in op1 to right by the number of times specified by op2 .

➢**Example:**

(1)

```
        SHR    BP, 1   ; Shift word in BP by 1 bit position to right
                       ; and 0 is kept to MSB
```

(2)

```
        MOV  CL, 03H              ;Load desired number of shifts into CL

        SHR    BYTE PYR[BX]       ;Shift bytes in DS at offset BX and
                                  ;rotate 3 bits to right and keep 3 0's in MSB
```

(3)

```
                             ;SI = 10010011  10101101 , CF = 0
        SHR    SI, 1         ; Result: SI = 01001001  11010110
```

; CF = 1, OF = 1, SF = 0, ZF = 0

- **TEST** Instruction – AND operand to update flags

- **WAIT** Instruction - Wait for test signal or interrupt signal

- **XCHG** Instruction - Exchange XCHG destination, source

- **XLAT/**
  XLATB Instruction - Translate a byte in AL

- **XOR** Instruction - Exclusive OR corresponding bits of two operands –
  XOR destination, source

- **TEST** Instruction - This instruction ANDs the contents of a source byte or word with the contents of specified destination word. Flags are updated but neither operand is changed . TEST instruction is often used to set flags before a condition jump instruction

- **Examples:**

```
TEST  AL, BH        ;AND BH with AL. no result is
                    ;stored . Update PF, SF, ZF

TEST  CX, 0001H     ;AND CX with immediate
                    ;number
                    ;no result is stored, Update PF,
                    ;SF
```

- **Example :**

```
                    ;AL = 01010001
TEST  Al, 80H       ;AND immediate 80H with AL to
                    ;test f MSB of AL is 1 or 0
                    ;ZF = 1 if MSB of AL = 0
                    ;AL = 01010001 (unchanged)
                    ;PF = 0 , SF = 0
                    ;ZF = 1 because ANDing produced   is 00
```

- **WAIT** Instruction - When this WAIT instruction executes, the 8086 enters an idle condition. This will stay in this state until a signal is asserted on TEST input pin or a valid interrupt signal is received on the INTR or NMI pin.

```
FSTSW      STATUS     ; copy 8087 status word to memory
FWAIT                 ; wait for 8087 to finish before-
                      ; doing next 8086 instruction
```

```
        MOV  AX, STATUS          ;copy status word to AX to
                                 ;check bits
```

➢In this code we are adding up of FWAIT instruction so that it will stop the execution of the command until the above instruction is finishes it's work .so that you are not loosing data and after that you will allow to continue the execution of instructions.

➢**XCHG** Instruction - The Exchange instruction exchanges the contents of the register with the contents of another register (or) the contents of the register with the contents of the memory location. Direct memory to memory exchange are not supported.

```
        XCHG        op1, op2
```

The both operands must be the same size and one of the operand must always be a register .

Example:

```
        XCHG        AX, DX                ;Exchange word in AX with word in DX
        XCHG        BL, CH                ;Exchange byte in BL with byte  in  CH
        XCHG        AL, Money [BX]        ;Exchange byte in AL with byte
                                          ;in memory at EA.
```

➢**XOR** Instruction - XOR performs a bit wise logical XOR of the operands specified by op1 and op2. The result of the operand is stored in op1 and is used to set the flag.

```
                    XOR   op1, op2
```

Example : (Numerical)

```
                                 ; BX = 00111101 01101001
                                 ;CX =  00000000 11111111
        XOR   BX, CX             ;Exclusive OR CX with BX
                                 ;Result BX = 00111101 10010110
```

Assembler Directives

➢**ASSUME**

➢**DB** - Defined Byte.

➢**DD** - Defined Double Word

➢**DQ** - Defined Quad Word

➢**DT** - Define Ten Bytes

➢**DW** - Define Word

➢**ASSUME Directive** - The ASSUME directive is used to tell the assembler that the name of the logical segment should be used for a specified segment. The 8086 works directly with only 4 physical segments: a Code segment, a data segment, a stack segment, and an extra segment.

➢**Example:**

ASUME CS:CODE ;This tells the assembler that the logical segment named CODE contains the instruction statements for the program and should be treated as a code segment.

ASUME DS:DATA ;This tells the assembler that for any instruction which refers to a data in the data segment, data will found in the logical segment DATA.

➢**DB** - DB directive is used to declare a byte-type variable or to store a byte in memory location.

➢**Example:**

1. **PRICE DB 49h, 98h, 29h** ;Declare an array of 3 bytes,
; named as PRICE and initialize.

2. **NAME DB 'ABCDEF'** ;Declare an array of 6 bytes and
; initialize with ASCII code for letters

3. **TEMPDB 100 DUP(?)** ;Set 100 bytes of storage in memory and give it the name as TEMP, but leave the 100 bytes uninitialized. Program instructions will load values into these locations.

➢**DW** - The DW directive is used to define a variable of type word or to reserve storage location of type word in memory.

➢**Example:**

MULTIPLIER DW 437Ah ; this declares a variable of type word and named it as MULTIPLIER. This variable is initialized with the value 437Ah when it is loaded into memory to run.

EXP1 DW 1234h, 3456h, 5678h ; this declares an array of 3 words and initialized with specified values.

STOR1 DW 100 DUP(0); Reserve an array of 100 words of memory and initialize all words with 0000.Array is named as STOR1.

➢**END** - END directive is placed after the last statement of a program to tell the assembler that this is the end of the program module. The assembler will ignore any statement after an END directive. Carriage return is required after the END directive.

➢**ENDP** - ENDP directive is used along with the name of the procedure to indicate the end of a procedure to the assembler

➢**Example:**

```
        SQUARE_NUM      PROCE ; It start the procedure
                                ;Some steps to find the square root of a number

        SQUARE_NUM      ENDP ;Hear it is the End for the procedure
```

➢**END** - End Program

➢**ENDP** - End Procedure

➢**ENDS** - End Segment

➢**EQU** - Equate

➢**EVEN** - Align on Even Memory Address

➢**EXTRN**

➢**ENDS** - This ENDS directive is used with name of the segment to indicate the end of that logic segment.

➢**Example:**

```
                CODE        SEGMENT    ;Hear it Start the logic
                                       ;segment  containing code
                                       ; Some instructions statements to perform
                                       ;the logical operation

                CODE        ENDS       ;End of segment named as
                                               ;CODE
```

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

➢**Example:**

FACTOR **EQU 03H** ; you has to write this statement at the starting of your program and later in the program you can use this as follows

ADD AL, FACTOR ; When it codes this instruction the assembler will code it as ADDAL, 03H

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

➢**EVEN** - This **EVEN** directive instructs the assembler to increment the location of the counter to the next even address if it is not already in the even address. If the word is at even address 8086 can read a memory in 1 bus cycle.

If the word starts at an odd address, the 8086 will take 2 bus cycles to get the data. A series of words can be read much more quickly if they are at even address. When EVEN is used the location counter will simply incremented to next address and NOP instruction is inserted in that incremented location.

➢**Example**:

```
        DATA1       SEGMENT
                ; Location counter will point to 0009 after assembler reads
                 ;next statement

        SALES  DB  9  DUP(?)        ;declare an array of 9  bytes
        EVEN                        ; increment  location counter to 000AH
        RECORD  DW  100  DUP( 0 )  ;Array of 100 words will start from an even
address for quicker read
        DATA1       ENDS
```

➢**GROUP** - Group Related Segments

➢**LABLE**

➢**NAME**

➢**OFFSET**

➢**ORG** - Originate

➢**GROUP** - The **GROUP** directive is used to group the logical segments named after the directive into one logical group segment.

➢**INCLUDE** - This **INCLUDE** directive is used to insert a block of source code from the named file into the current source module.

➢**PROC** - Procedure

➢**PTR** - Pointer

➢**PUBLC**

➢**SEGMENT**

➢**SHORT**

➢**TYPE**

➢**PROC** - The **PROC** directive is used to identify the start of a procedure. The term near or far is used to specify the type of the procedure.

➢**Example:**

SMART PROC FAR ; This identifies that the start of a procedure named as SMART and instructs the assembler that the procedure is far .

SMART ENDP

This PROC is used with ENDP to indicate the break of the procedure.

➢**PTR** - This **PTR** operator is used to assign a specific type of a variable or to a label.

➢**Example**:

INC [BX] ; This instruction will not know whether to increment the byte pointed to by BX or a word pointed to by BX.

INC BYTE PTR [BX] ;increment the byte ;pointed to by BX

This PTR operator can also be used to override the declared type of variable . If we want to access the a byte in an array **WORDS DW 437Ah, 0B97h,**

MOV AL, BYTE PTR WORDS

➢**PUBLIC** - The **PUBLIC** directive is used to instruct the assembler that a specified name or label will be accessed from other modules.

➢**Example:**

PUBLIC DIVISOR, DIVIDEND ;these two variables are public so these are available to all modules.

If an instruction in a module refers to a variable in another assembly module, we can access that module by declaring as **EXTRN** directive.

➢**TYPE** - **TYPE** operator instructs the assembler to determine the type of a variable and determines the number of bytes specified to that variable.

➢**Example:**

Byte type variable – assembler will give a value 1
Word type variable – assembler will give a value 2
Double word type variable – assembler will give a value 4

ADD BX, TYPE WORD_ ARRAY ; hear we want to increment BX to point to next word in an array of words.

DOS Function Calls

- **AH 00H** : Terminate a Program
- **AH 01H** : Read the Keyboard
- **AH 02H** : Write to a Standard Output Device
- **AH 08H** : Read a Standard Input without Echo
- **AH 09H** : Display a Character String
- **AH 0AH** : Buffered keyboard Input
- **INT 21H** : Call DOS Function

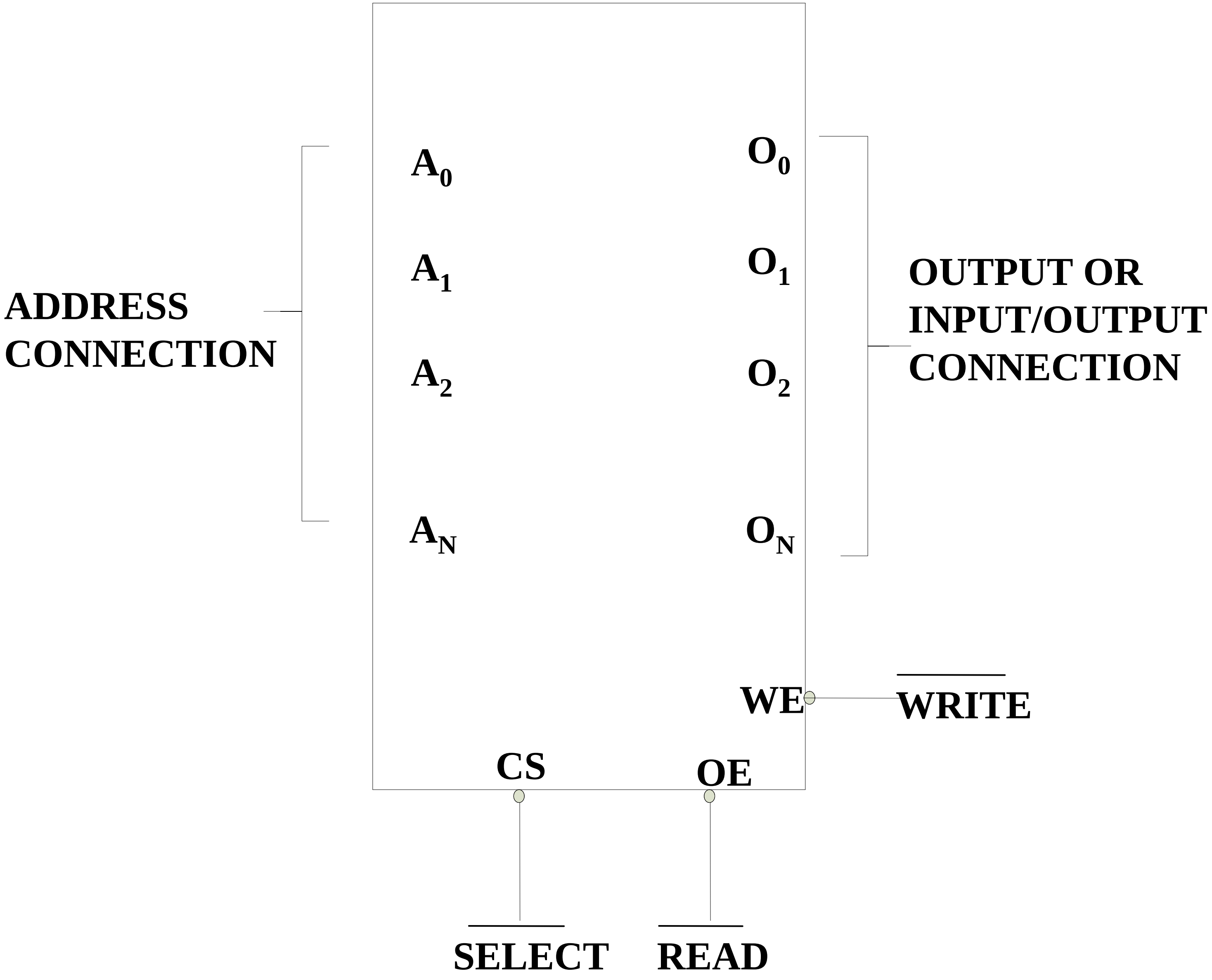
Interface

- 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.

Next Page

- ***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.

Next Page



ADDRESS CONNECTION

A_0

A_1

A_2

A_N

O_0

O_1

O_2

O_N

OUTPUT OR INPUT/OUTPUT CONNECTION

WE

$\overline{\text{WRITE}}$

CS

OE

$\overline{\text{SELECT}}$

$\overline{\text{READ}}$

MEMORY COMPONENT ILLUSTRATING THE ADDRESS, DATA AND CONTROL CONNEC'TIONS

Next Page

- RAM memory generally has at least one $\overline{\text{CS}}$ or $\overline{\text{S}}$ input and ROM at least one $\overline{\text{CE}}$.
- If the $\overline{\text{CE}}$, $\overline{\text{CS}}$, $\overline{\text{S}}$ 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 $\overline{\text{CS}}$ 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.

Next Page

- 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.

Next Page

- 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.

- 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.

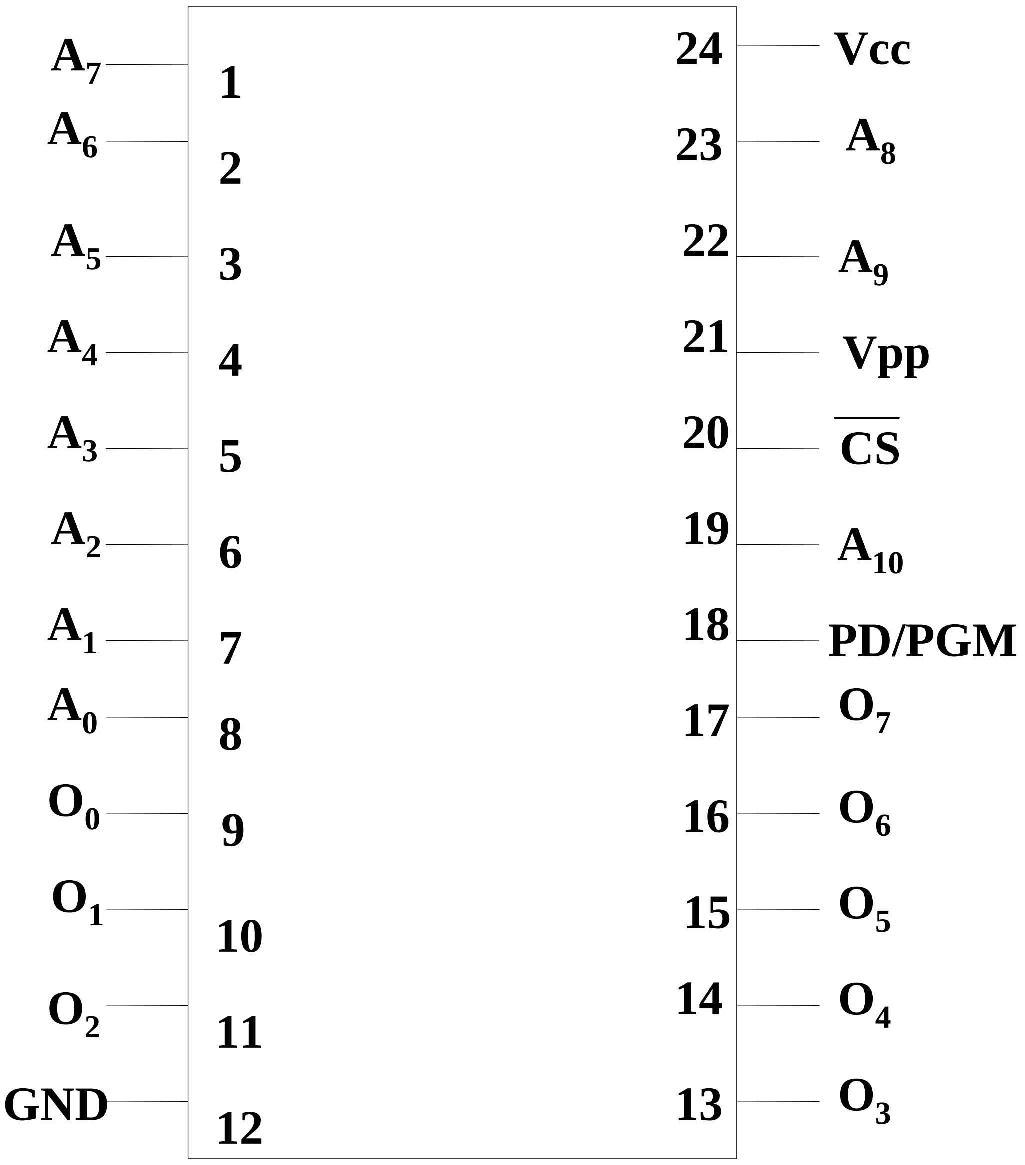
Next Page

- 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..

Next Page

- Each of these parts contains address pins, eight data connections, one or more chip selection inputs ($\overline{\text{CE}}$) and an output enable pin ($\overline{\text{OE}}$).
- This device contains **11** address inputs and **8** data outputs.
- If both the pin connection $\overline{\text{CE}}$ and $\overline{\text{OE}}$ 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.

Next Page



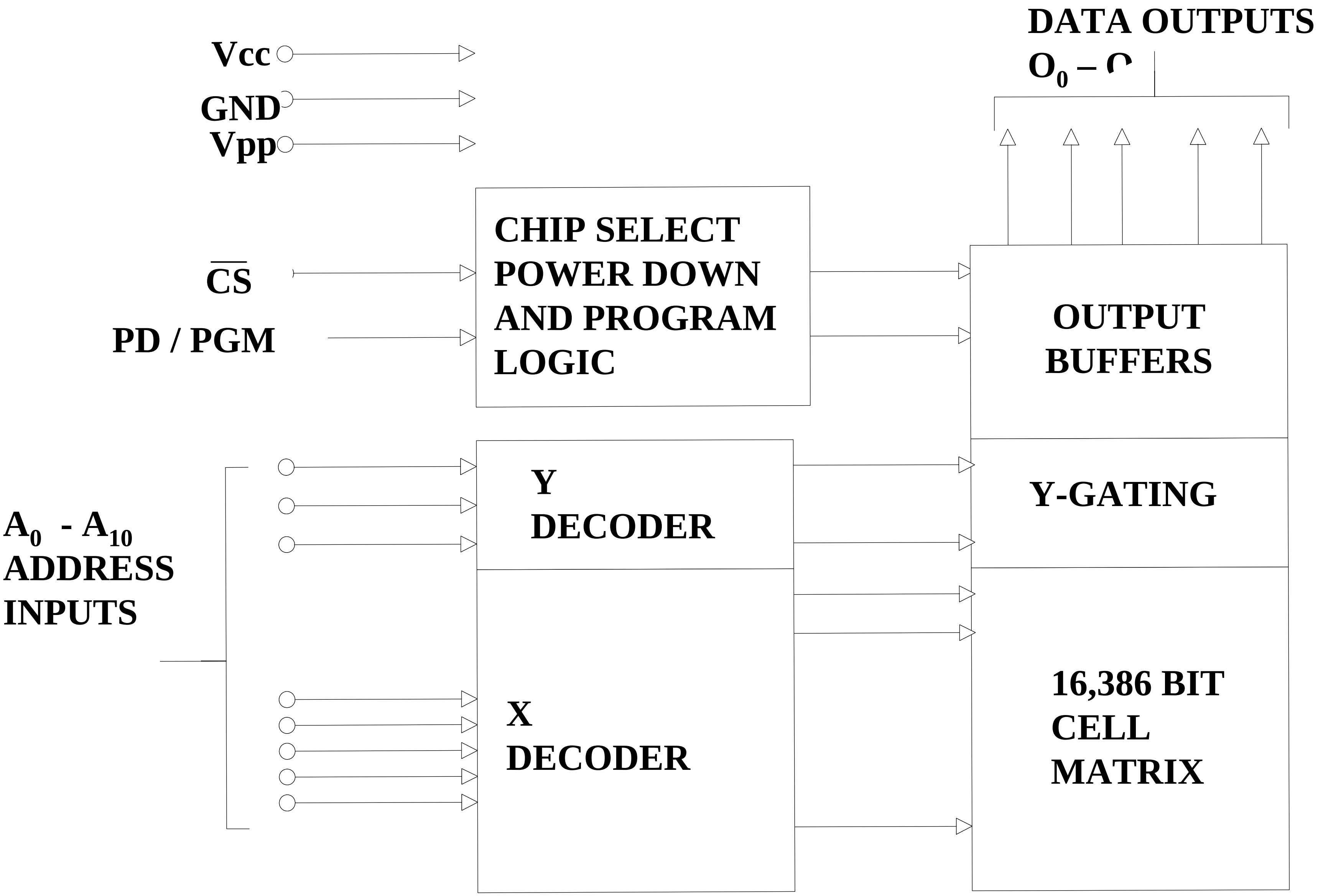
PIN CONFIGURATION OF 2716 EPROM

Next Page

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

PIN NAMES

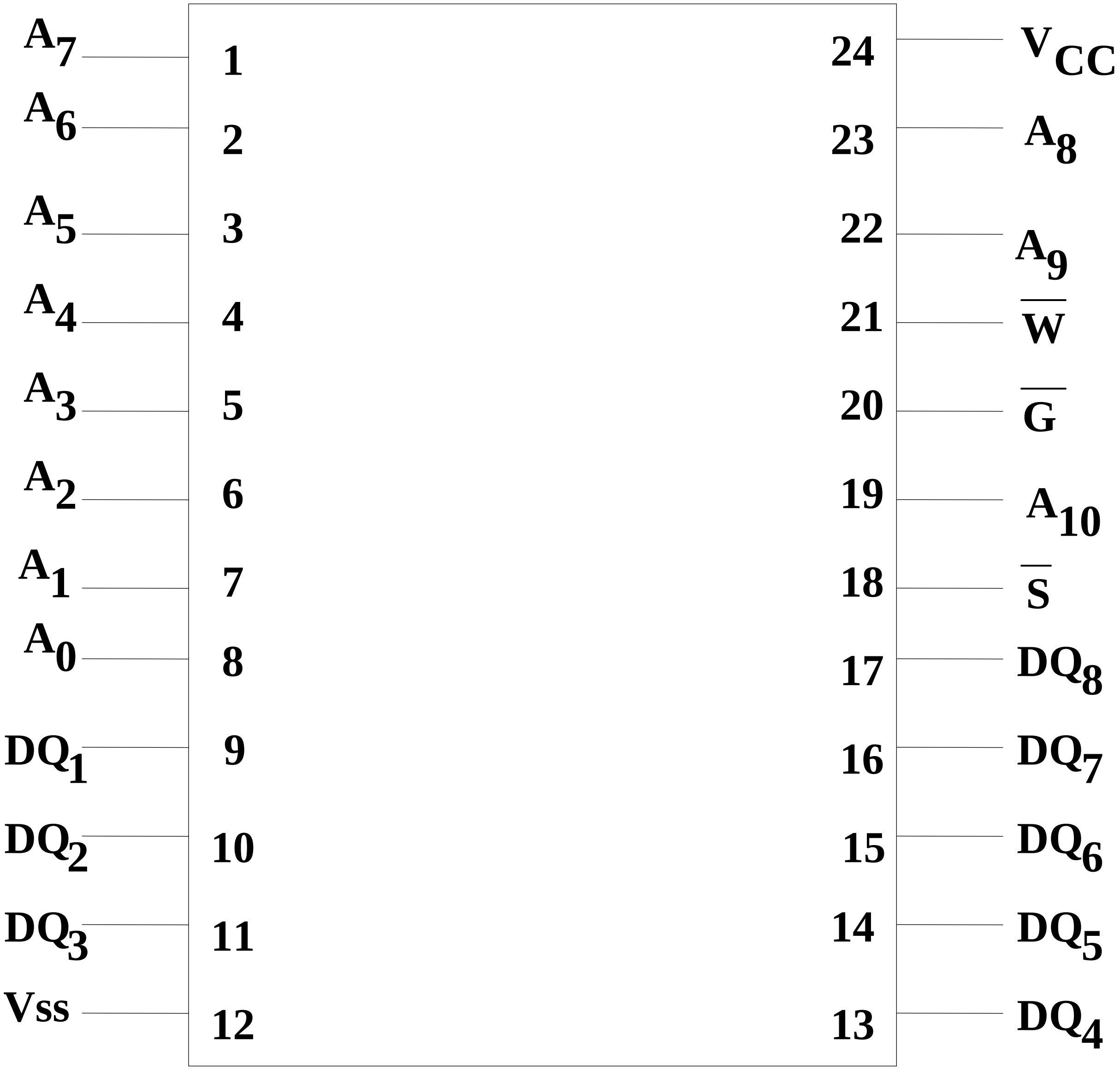
Next Page



BLOCK DIAGRAM

- 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.

Next Page



PIN CONFIGURATION OF TMS 4016 SRAM

Next Page

$A_0 – A_{10}$	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

Next Page

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

Static RAM Interfacing

- 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.

- 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 = 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.

- 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.

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.

- 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

- 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.

Next Page

- 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***.

Next Page

- 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.

Next Page

- 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.

Next Page

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.

Next Page

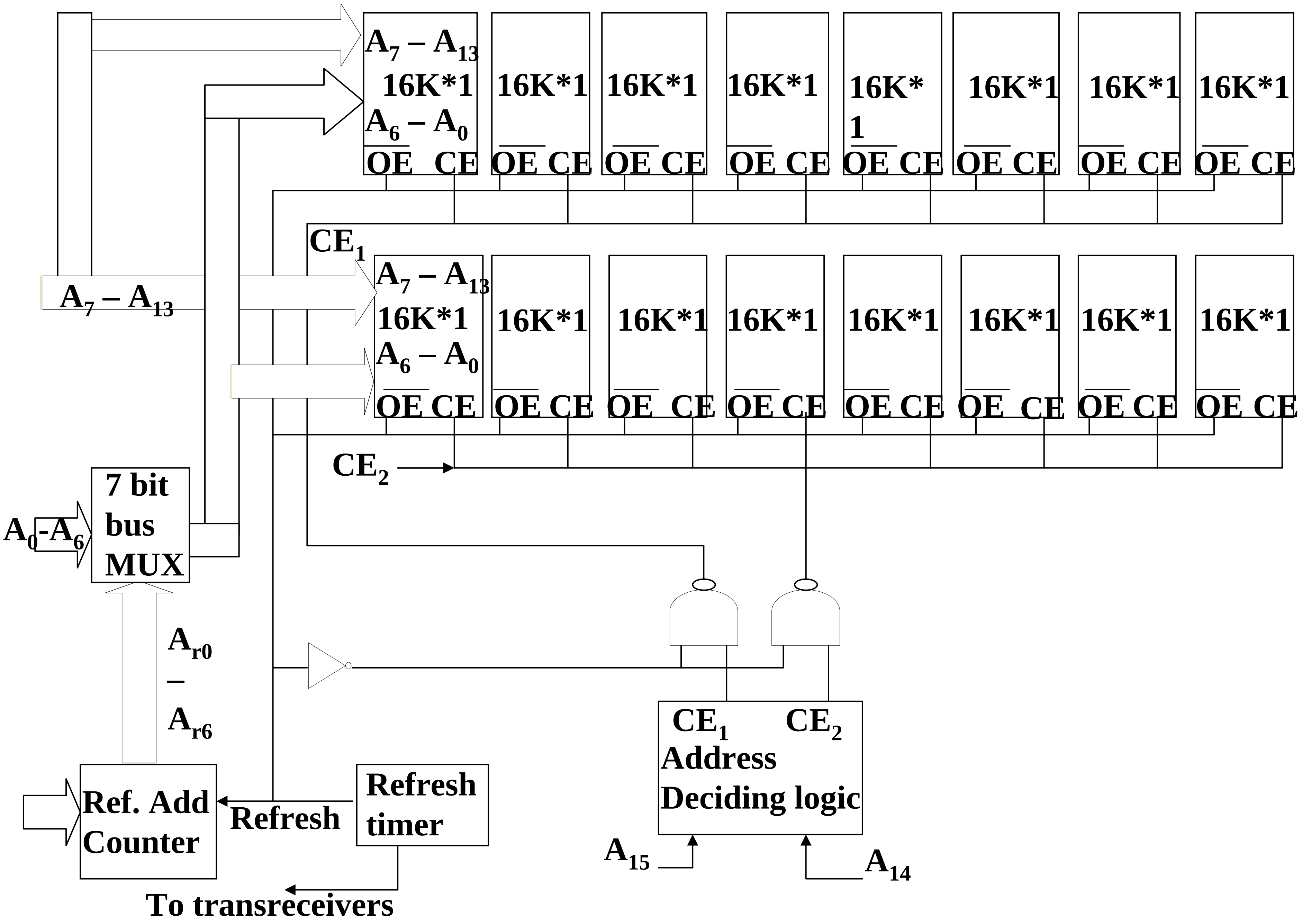
- Dynamic RAM is available in units of several kilobits to megabits of memory. This memory is arranged internally in a two dimensional matrix array so that it will have n rows and m columns. The row address n and column address m are important for the refreshing operation.
- For example, a typical 4K 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.

- 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.

- 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.

- 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.

- The $\overline{\text{OE}}$ 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, $\overline{\text{OE}}$ and $\overline{\text{CE}}$ 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

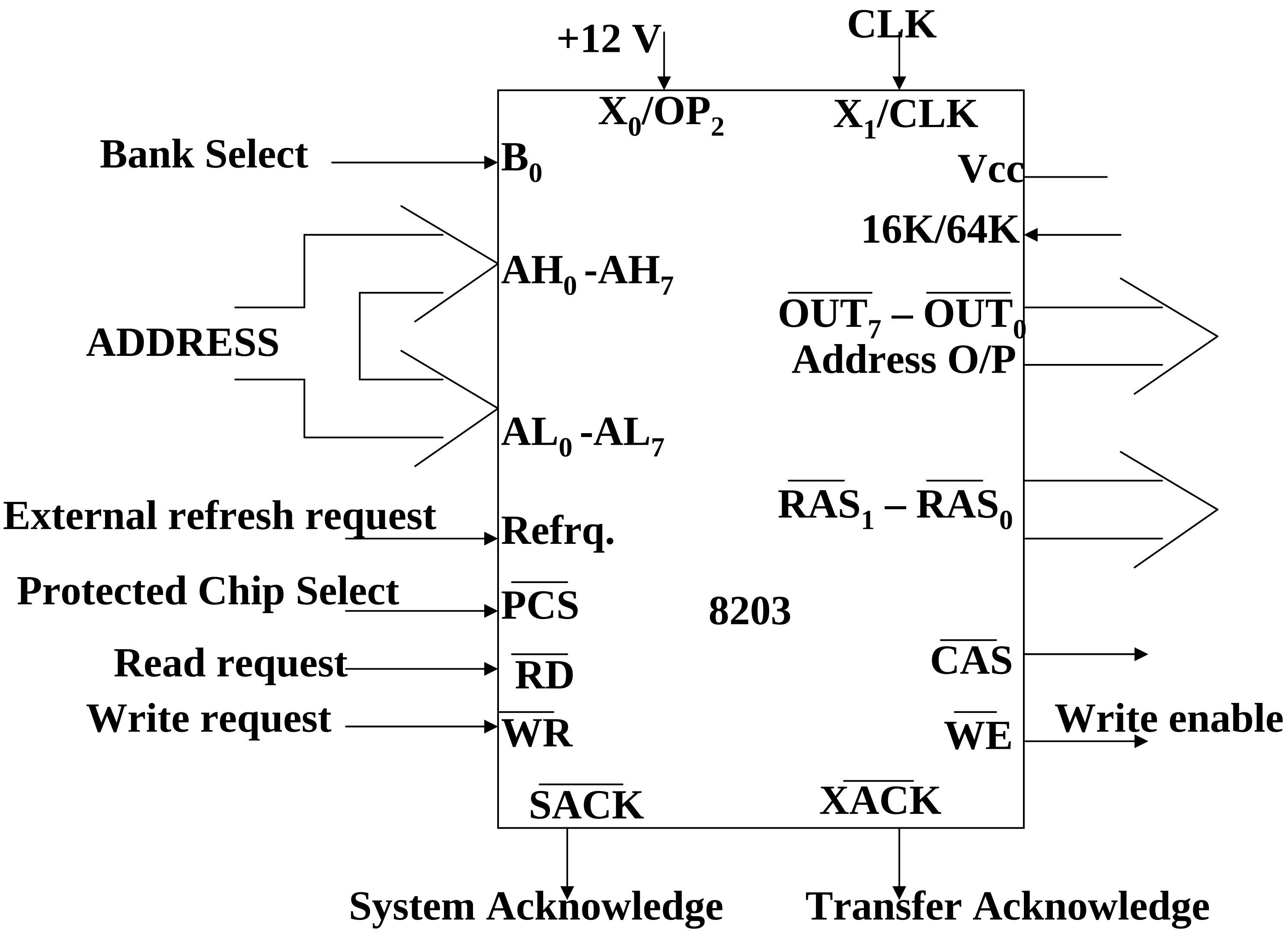


Fig : Dynamic RAM controller

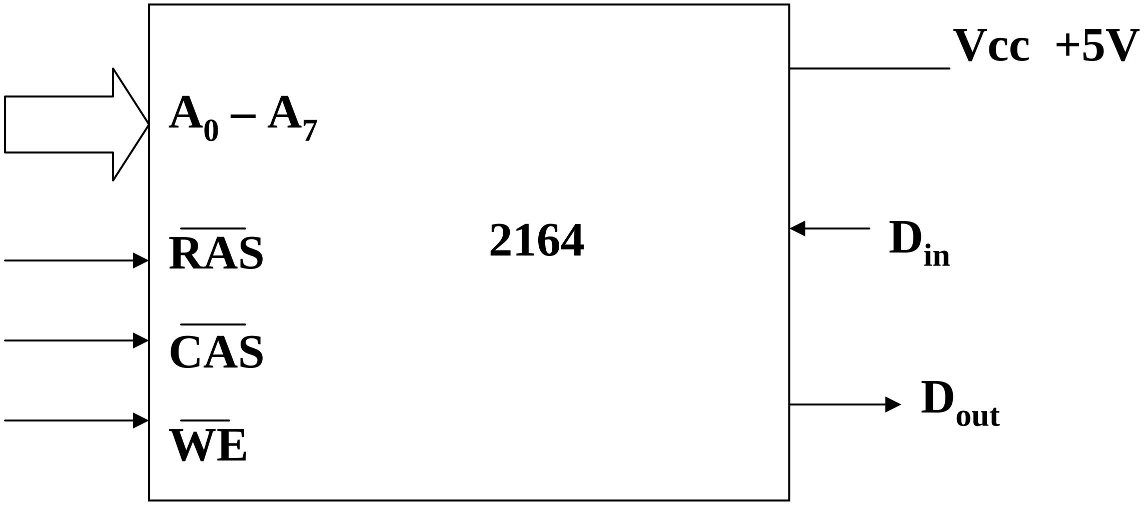


Fig : 1- bit Dynamic RAM

- The pin assignment for 2164 dynamic RAM is as in above fig.
- The $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$ 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 $\overline{\text{WE}}$ 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.

- 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.

- The $\overline{\mathrm{WE}}$ signal specifies the memory write cycle and is not output from 8203 that drives the WE input of dynamic RAM memory chip. The $\overline{\mathrm{OUT}}_0$ – $\overline{\mathrm{OUT}}_7$ 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.

- 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.

- 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).

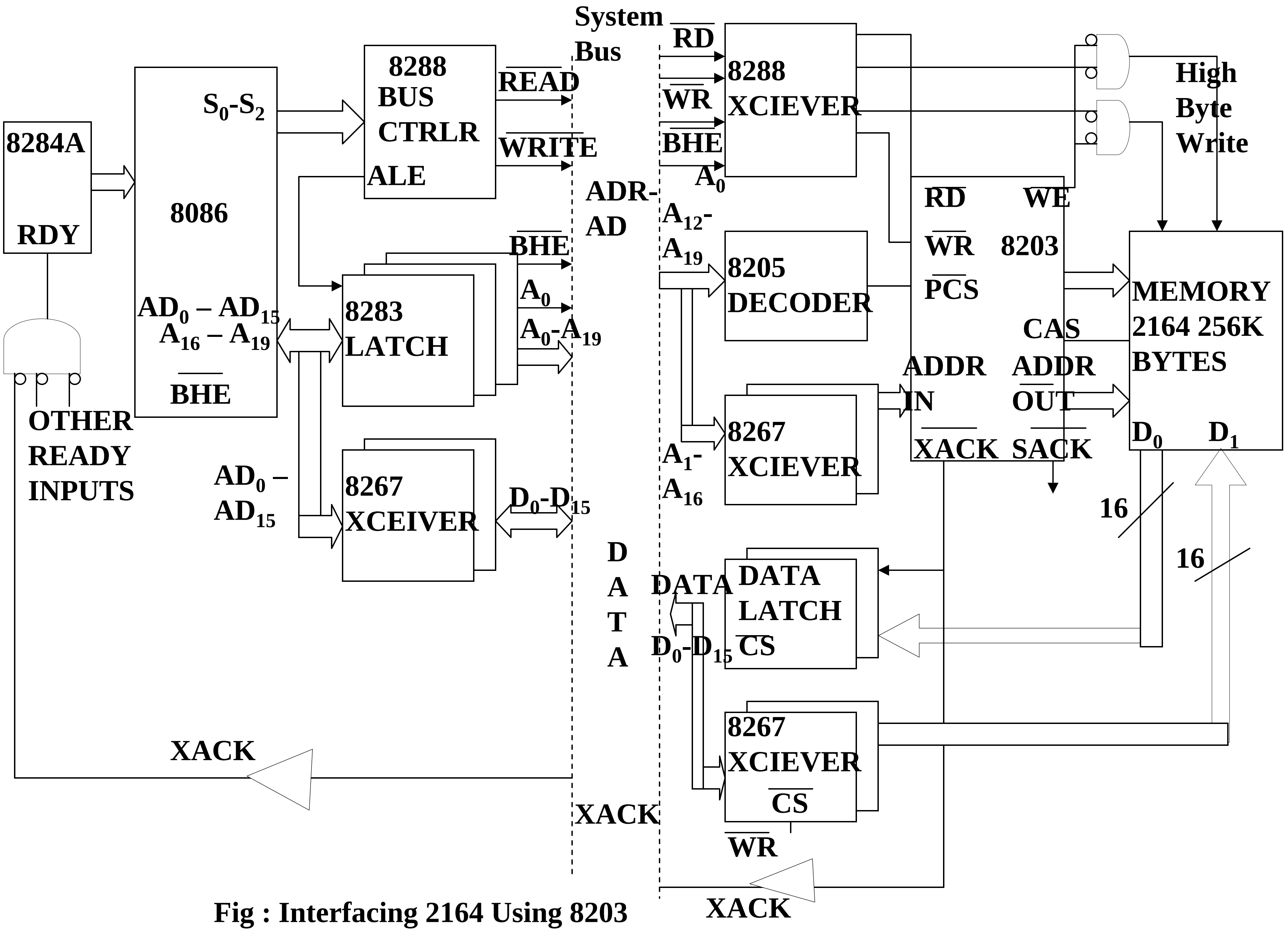


Fig : Interfacing 2164 Using 8203

- 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.

8254

- Compatible with All Intel and Most other Microprocessors
- Handles Inputs from DC to 10 MHz
- 8 MHz 8254
- 10 MHz 8254-2
- Status Read-Back Command
- Six Programmable Counter Modes
- Three Independent 16-Bit Counters
- Binary or BCD Counting
- Single a 5V Supply
- Standard Temperature Range

- The Intel 8254 is a counter/timer device designed to solve the common timing control problems in microcomputer system design.
- It provides three independent 16-bit counters, each capable of handling clock inputs up to 10 MHz.
- All modes are software programmable. The 8254 is a superset of the 8253.
- The 8254 uses HMOS technology and comes in a 24-pin plastic or CERDIP package.

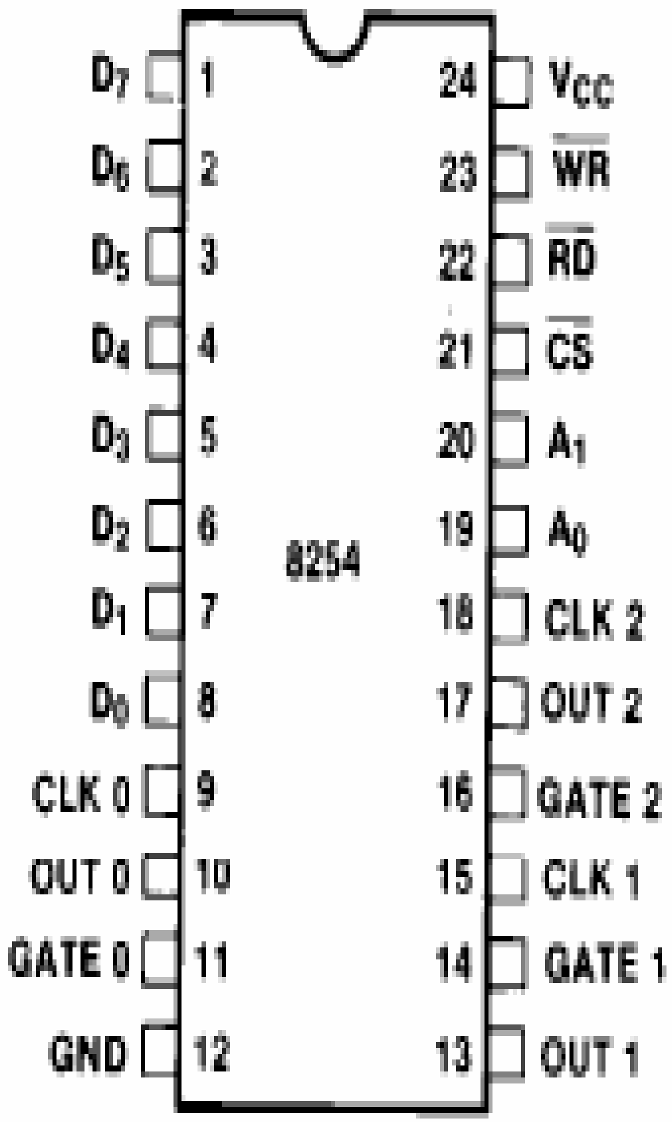


Figure 1. Pin Configuration

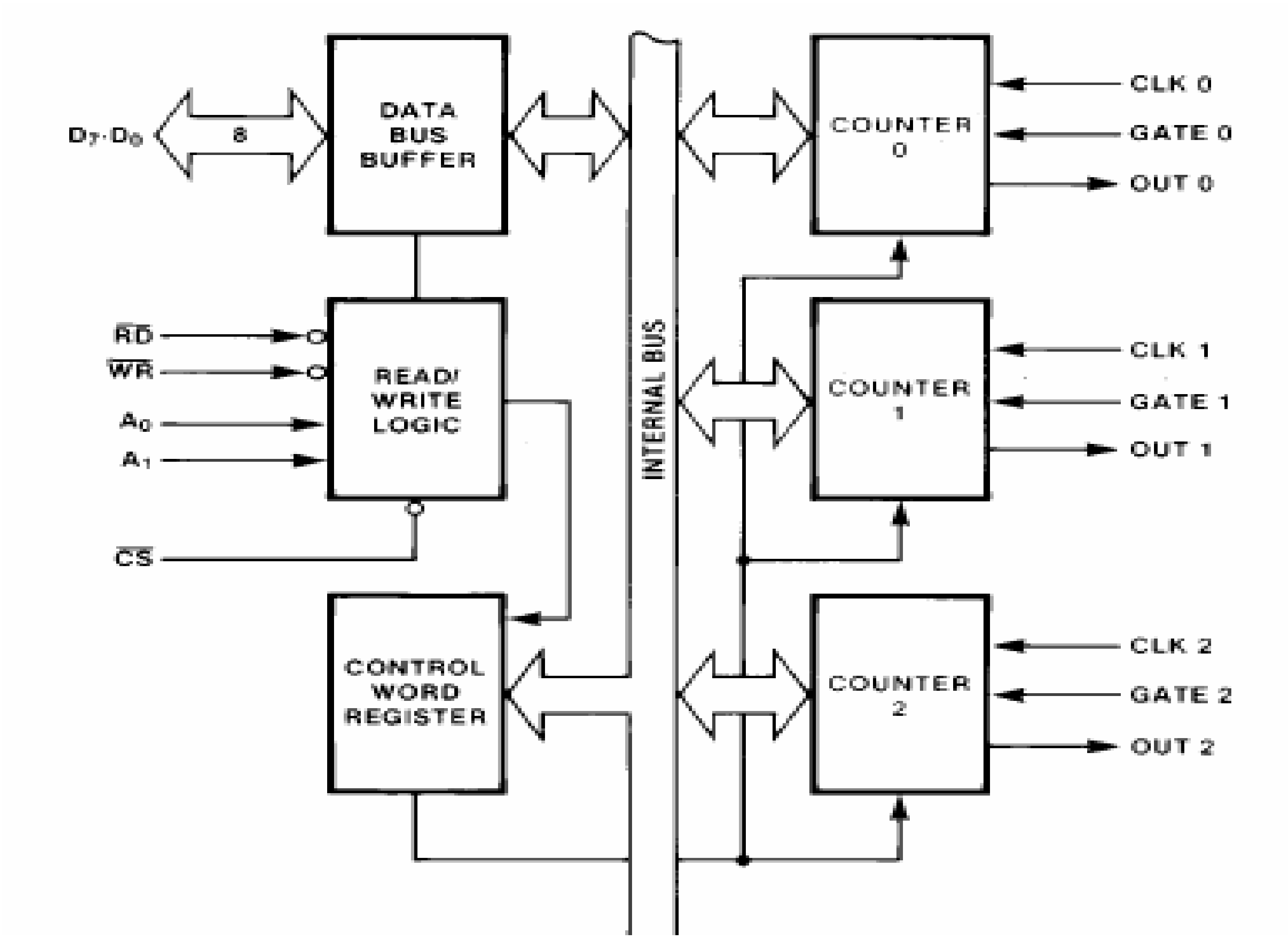


Figure 2. 8254 Block Diagram

Pin Description

Symbol	Pin No.	Type	Name and Function
D7-D0	1 - 8	I/O	DATA: Bi-directional three state data bus lines, connected to system data bus.
CLK 0	9	I	CLOCK 0: Clock input of Counter 0.
OUT 0	10	O	OUTPUT 0: Output of Counter 0.
GATE 0	11	I	GATE 0: Gate input of Counter 0.
GND	12		GROUND: Power supply connection.
VCC	24		POWER: A 5V power supply connection.
$\overline{WR}$	23	I	WRITE CONTROL: This input is low during CPU write operations.
RD	22	I	READ CONTROL: This input is low during CPU read operations.

<table>
<tr><td>CS</td><td>21</td><td>I</td><td colspan="3">CHIP SELECT: A low on this input enables the 8254 to respond to RD and WR signals. RD and WR are ignored otherwise.</td></tr>
<tr><td rowspan="6">A1, A0</td><td rowspan="6">20 – 9</td><td rowspan="6">I</td><td colspan="3">ADDRESS: Used to select one of the three Counters or the Control Word Register for read or write operations. Normally connected to the system address bus.</td></tr>
<tr><td>A1</td><td>A0</td><td>Selects</td></tr>
<tr><td>0</td><td>0</td><td>Counter 0</td></tr>
<tr><td>0</td><td>1</td><td>Counter 1</td></tr>
<tr><td>1</td><td>0</td><td>Counter 2</td></tr>
<tr><td>1</td><td>1</td><td>Control Word Register</td></tr>
<tr><td>CLK 2</td><td>18</td><td>I</td><td colspan="3">CLOCK 2: Clock input of Counter 2.</td></tr>
<tr><td>OUT 2</td><td>17</td><td>O</td><td colspan="3">OUT 2: Output of Counter 2.</td></tr>
</table>

GATE 2	16	I	GATE 2: Gate input of Counter 2.
CLK 1	15	I	CLOCK 1: Clock input of Counter 1.
GATE 1	14	I	GATE 1: Gate input of Counter 1.
OUT 1	OUT 1	O	OUT 1: Output of Counter 1.

Functional Description

- The 8254 is a programmable interval timer/counter designed for use with Intel microcomputer systems.
- It is a general purpose, multi-timing element that can be treated as an array of I/O ports in the system software.
- The 8254 solves one of the most common problems in any microcomputer system, the generation of accurate time delays under software control. Instead of setting up timing loops in software, the programmer configures the 8254 to match his requirements and programs one of the counters for the desired delay.

- After the desired delay, the 8254 will interrupt the CPU. Software overhead is minimal and variable length delays can easily be accommodated.
- Some of the other counter/timer functions common to microcomputers which can be implemented with the 8254 are:
- Real time clock
- Event-counter
- Digital one-shot

- Programmable rate generator
- Square wave generator
- Binary rate multiplier
- Complex waveform generator
- Complex motor controller

Block Diagram

- **DATA BUS BUFFER**: This 3-state, bi-directional, 8-bit buffer is used to interface the 8254 to the system bus, see the figure : Block Diagram Showing Data Bus Buffer and Read/Write Logic Functions.
- **READ/WRITE LOGIC** : The Read/Write Logic accepts inputs from the system bus and generates control signals for the other functional blocks of the 8254. A_1 and A_0 select one of the three counters or the Control Word Register to be read from/written into.
- A ``low" on the $\overline{\text{RD}}$ input tells the 8254 that the CPU is reading one of the counters.

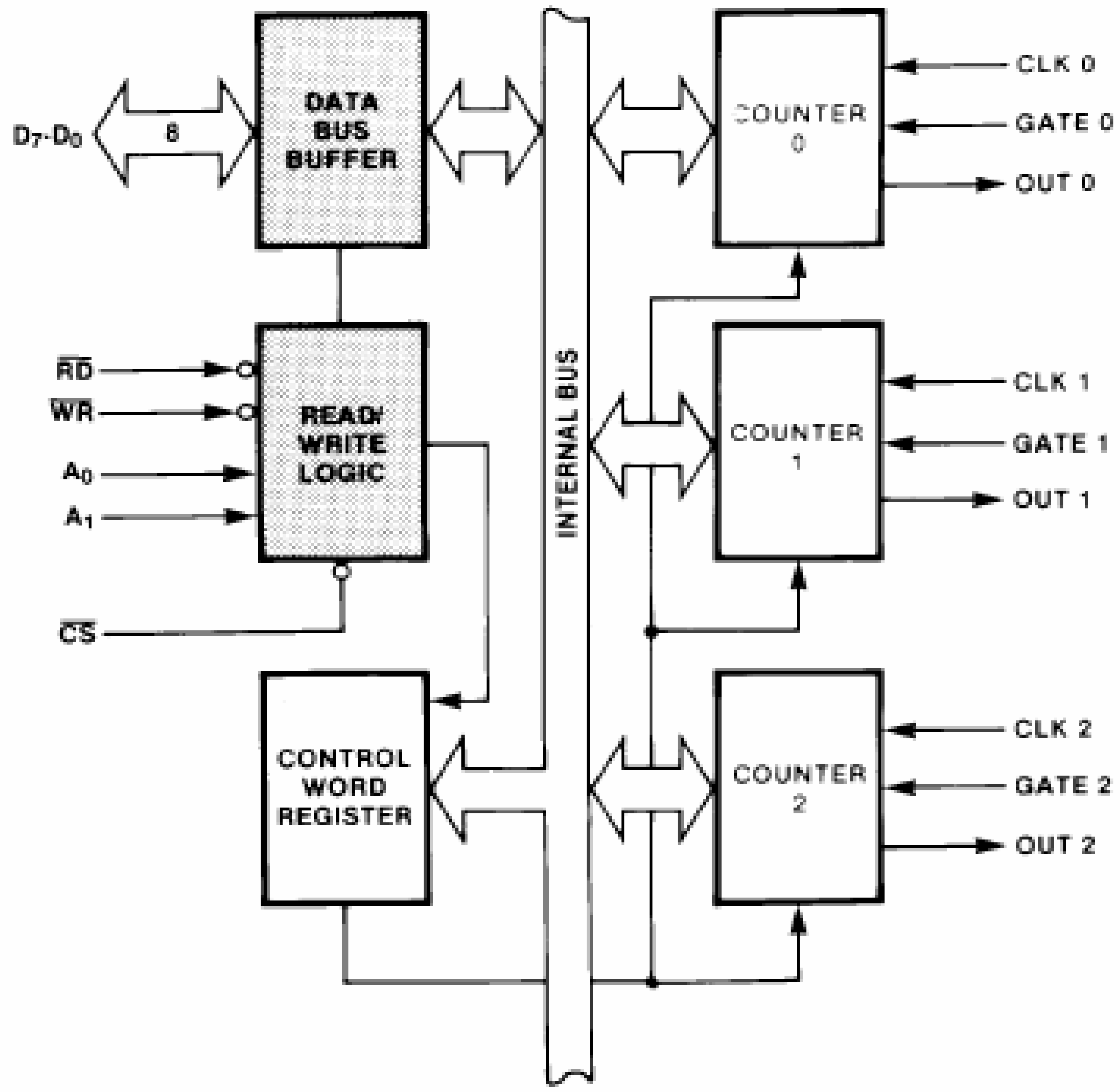


Figure 3. Block Diagram Showing Data Bus Buffer and Read/Write Logic Functions

- A ``low'' on the $\overline{\text{WR}}$ input tells the 8254 that the CPU is writing either a Control Word or an initial count. Both $\overline{\text{RD}}$ and WR are qualified by CS; RD and WR are ignored unless the 8254 has been selected by holding CS low.
- **CONTROL WORD REGISTER :**The Control Word Register (see Figure 4) is selected by the Read/Write Logic when A_1,A_0 = 11. If the CPU then does a write operation to the 8254, the data is stored in the Control Word Register and is interpreted as a Control Word used to define the operation of the Counters.

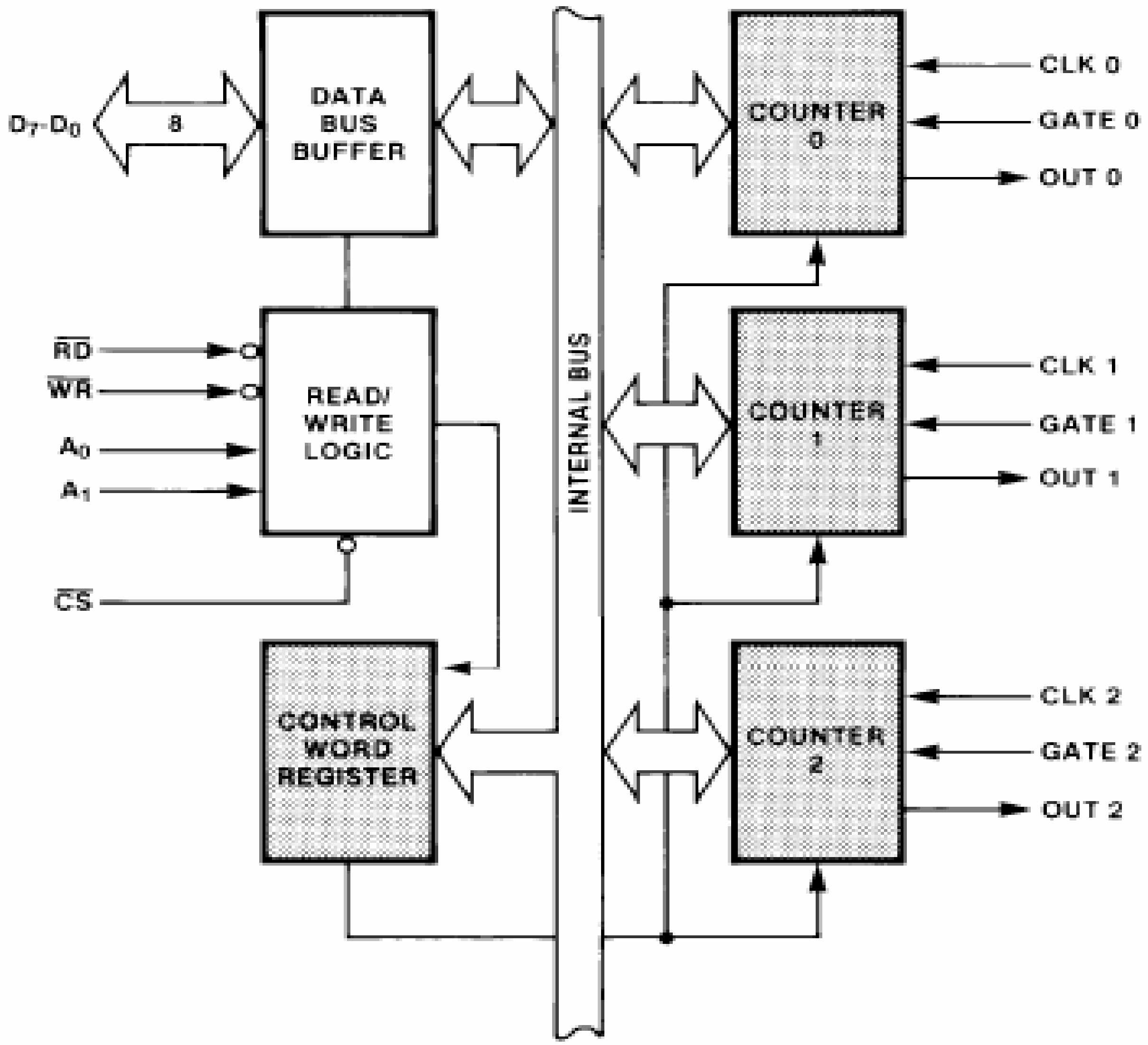


Figure 4. Block Diagram Showing Control Word Register and Counter Functions

- The Control Word Register can only be written to; status information is available with the Read-Back Command.
- **COUNTER 0, COUNTER 1, COUNTER 2** :These three functional blocks are identical in operation, so only a single Counter will be described. The internal block diagram of a single counter is shown in Figure 5.
- The Counters are fully independent. Each Counter may operate in a different Mode.
- The Control Word Register is shown in the figure; it is not part of the Counter itself, but its contents determine how the Counter operates.

- The status register, shown in Figure 5, when latched, contains the current contents of the Control Word Register and status of the output and null count flag. (See detailed explanation of the Read-Back command.)
- The actual counter is labelled CE (for ``Counting Element''). It is a 16-bit presettable synchronous down counter. OLM and OLL are two 8-bit latches. OL stands for ``Output Latch''; the subscripts M and L stand for ``Most significant byte'' and ``Least significant byte'‘ respectively.

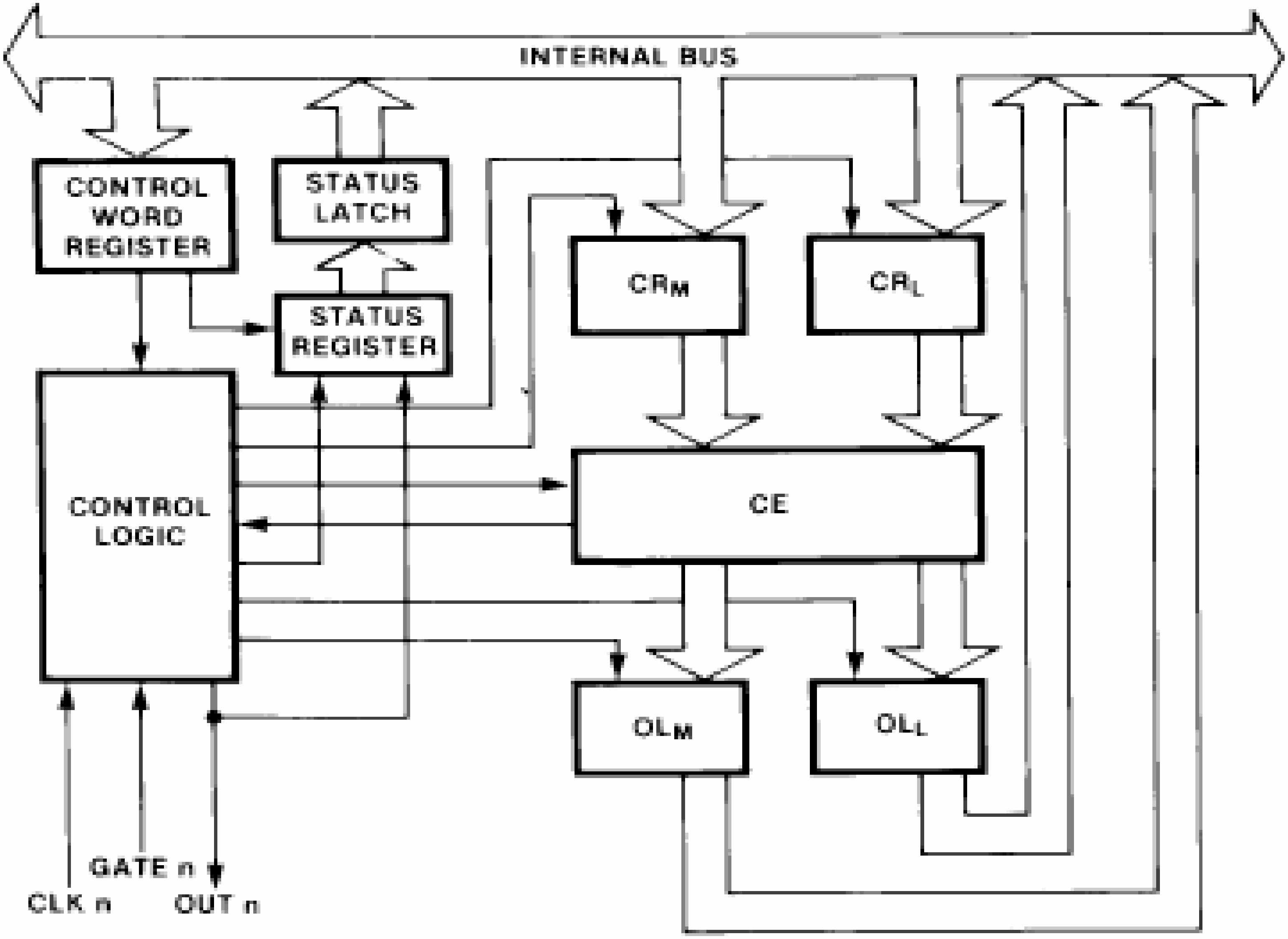


Figure 5. Internal Block Diagram of a Counter

- Both are normally referred to as one unit and called just OL. These latches normally ``follow'‘ the CE, but if a suitable Counter Latch Command is sent to the 8254, the latches ``latch" the present count until read by the CPU and then return to ``following" the CE.
- One latch at a time is enabled by the counter's Control Logic to drive the internal bus. This is how the 16-bit Counter communicates over the 8-bit internal bus. Note that the CE itself cannot be read; whenever you read the count, it is the OL that is being read.

- Similarly, there are two 8-bit registers called CRM and CRL (for ``Count Register"). Both are normally referred to as one unit and called just CR.
- When a new count is written to the Counter, the count is stored in the CR and later transferred to the CE. The Control Logic allows one register at a time to be loaded from the internal bus. Both bytes are transferred to the CE simultaneously.
- CRM and CRL are cleared when the Counter is programmed. In this way, if the Counter has been programmed for one byte counts (either most significant byte only or least significant byte only) the other byte will be zero.

- Note that the CE cannot be written into, whenever a count is written, it is written into the CR.
- The Control Logic is also shown in the diagram.
- CLK n, GATE n, and OUT n are all connected to the outside world through the Control Logic.
- **8254 SYSTEM INTERFACE :**The 8254 is a component of the Intel Microcomputer Systems and interfaces in the same manner as all other peripherals of the family.
- It is treated by the system's software as an array of peripheral I/O ports; three are counters and the fourth is a control register for MODE programming.

- Basically, the select inputs A_0,A_1 connect to the A_0,A_1 address bus signals of the CPU. The CS can be derived directly from the address bus using a linear select method. Or it can be connected to the output of a decoder, such as an Intel 8205 for larger systems.
- **Programming the 8254 :**Counters are programmed by writing a Control Word and then an initial count.
- The Control Words are written into the Control Word Register, which is selected when A_1,A_0 = 11. The Control Word itself specifies which Counter is being programmed.

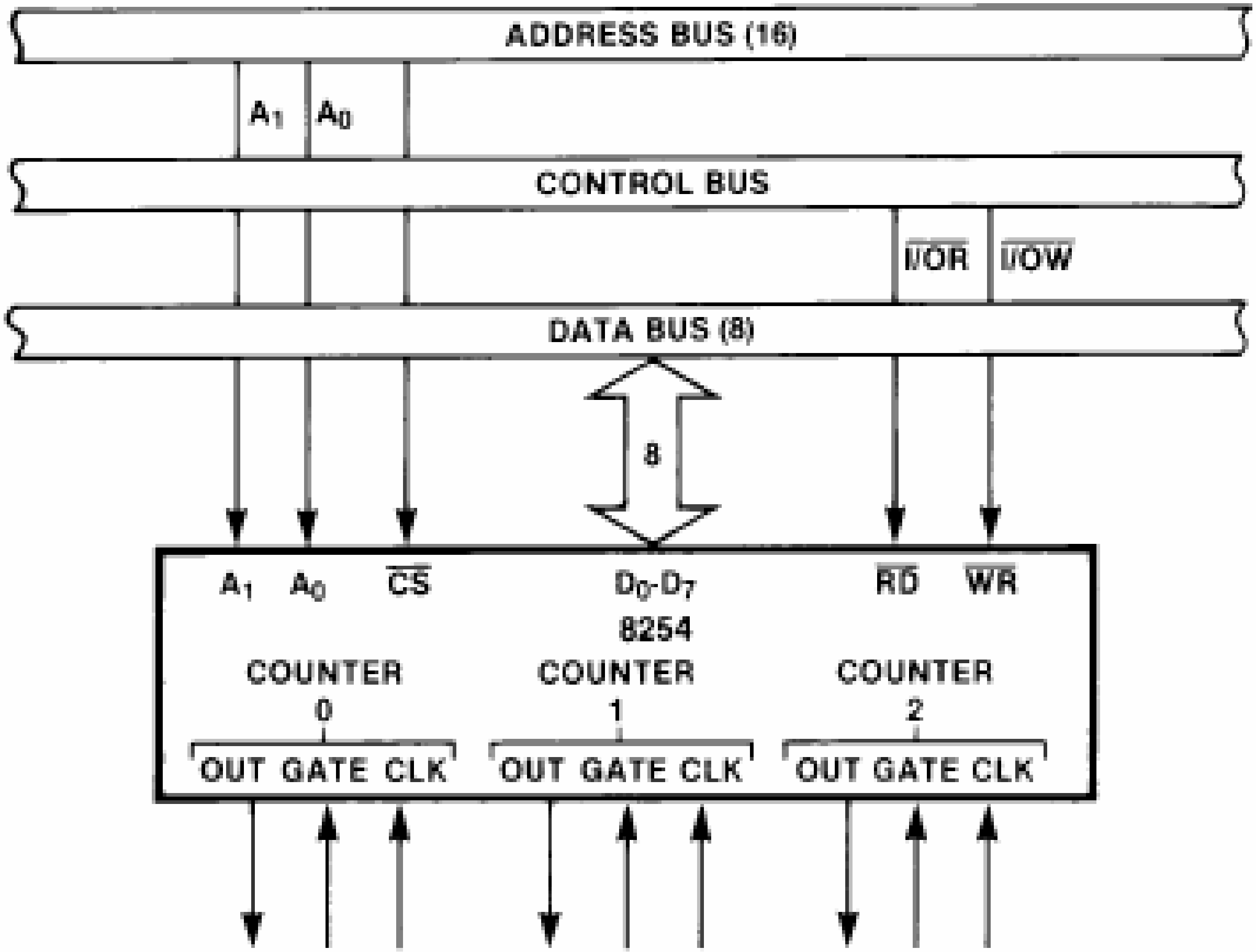


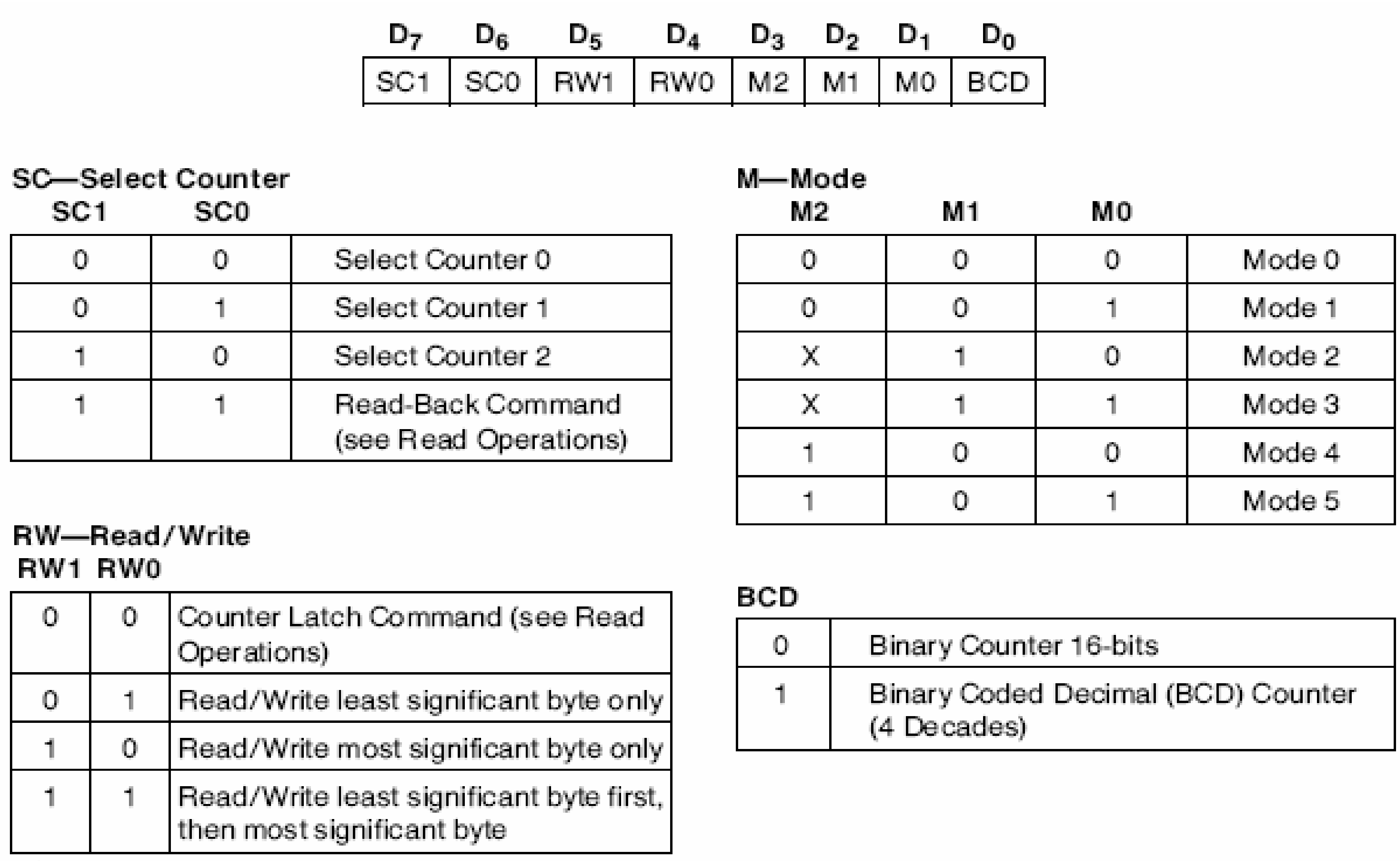
Figure 6. 8254 System Interface

- **Control Word Format:** A1,A0 = 11, CS = 0, $\overline{\text{RD}}$ = 1, $\overline{\text{WR}}$ = 0.
- By contrast, initial counts are written into the Counters, not the Control Word Register. The A1,A0 inputs are used to select the Counter to be written into. The format of the initial count is determined by the Control Word used.
- **Write Operations**: The programming procedure for the 8254 is very flexible. Only two conventions need to be remembered:

1) For each Counter, the Control Word must be written before the initial count is written.

2) The initial count must follow the count format specified in the Control Word (least significant byte only, most significant byte only, or least significant byte and then most significant byte).

- Since the Control Word Register and the three Counters have separate addresses (selected by the A_1,A_0 inputs), and each Control Word specifies the Counter it applies to (SC_0,SC_1 bits), no special instruction sequence is required.
- Any programming sequence that follows the conventions in Figure 7 is acceptable.



D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

SC—Select Counter

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Read-Back Command (see Read Operations)

M—Mode

M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

RW—Read/Write

RW1	RW0	
0	0	Counter Latch Command (see Read Operations)
0	1	Read/Write least significant byte only
1	0	Read/Write most significant byte only
1	1	Read/Write least significant byte first, then most significant byte

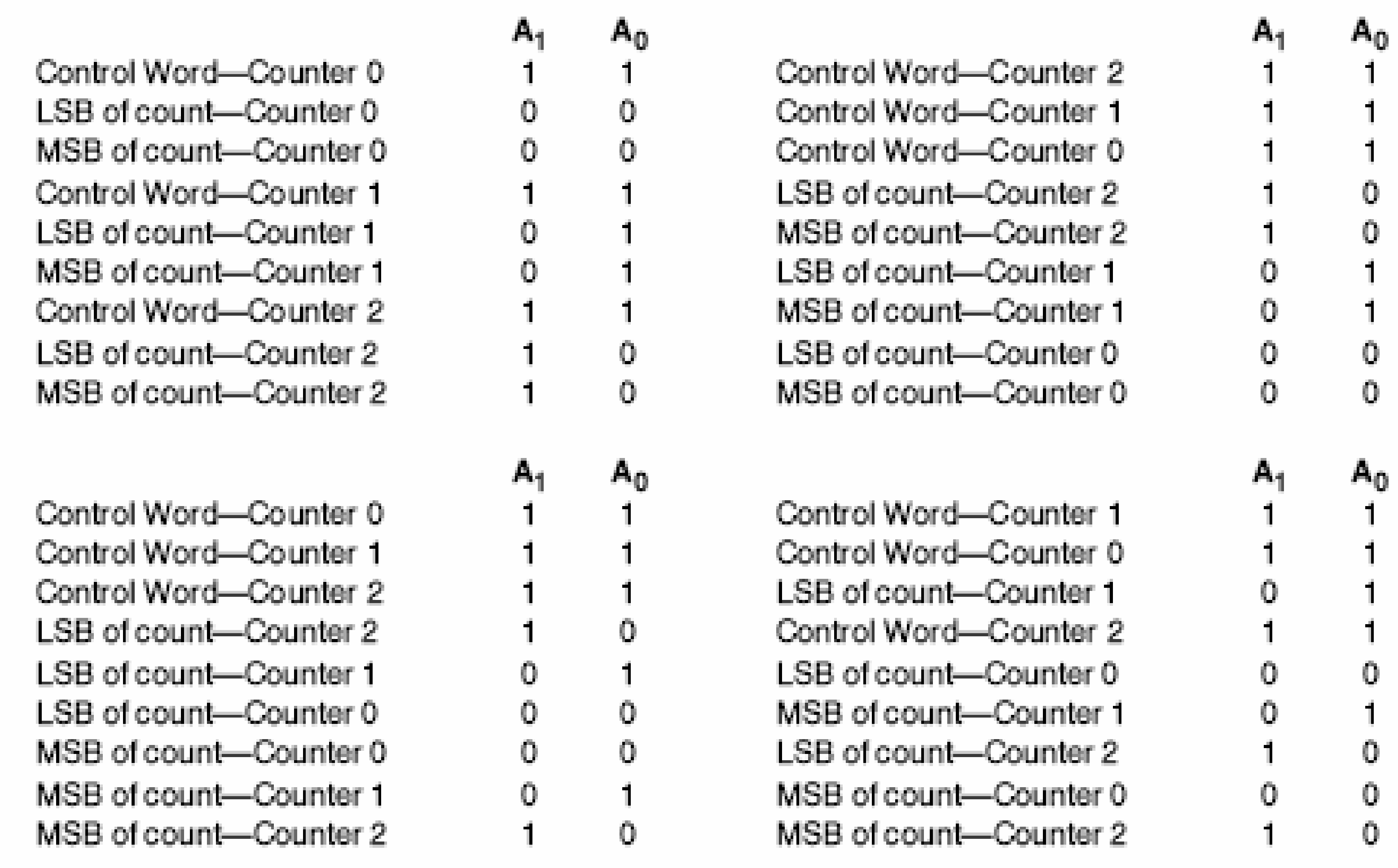
BCD

0	Binary Counter 16-bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

NOTE: Don't care bits (X) should be 0 to insure compatibility with future Intel products.

Figure 7. Control Word Format

- A new initial count may be written to a Counter at any time without affecting the Counter's programmed Mode in any way. Counting will be affected as described in the Mode definitions. The new count must follow the programmed count format.
- If a Counter is programmed to read/write two-byte counts, the following precaution applies: A program must not transfer control between writing the first and second byte to another routine which also writes into that same Counter. Otherwise, the Counter will be loaded with an incorrect count.



	A_1	A_0
Control Word—Counter 0	1	1
LSB of count—Counter 0	0	0
MSB of count—Counter 0	0	0
Control Word—Counter 1	1	1
LSB of count—Counter 1	0	1
MSB of count—Counter 1	0	1
Control Word—Counter 2	1	1
LSB of count—Counter 2	1	0
MSB of count—Counter 2	1	0

	A_1	A_0
Control Word—Counter 2	1	1
Control Word—Counter 1	1	1
Control Word—Counter 0	1	1
LSB of count—Counter 2	1	0
MSB of count—Counter 2	1	0
LSB of count—Counter 1	0	1
MSB of count—Counter 1	0	1
LSB of count—Counter 0	0	0
MSB of count—Counter 0	0	0

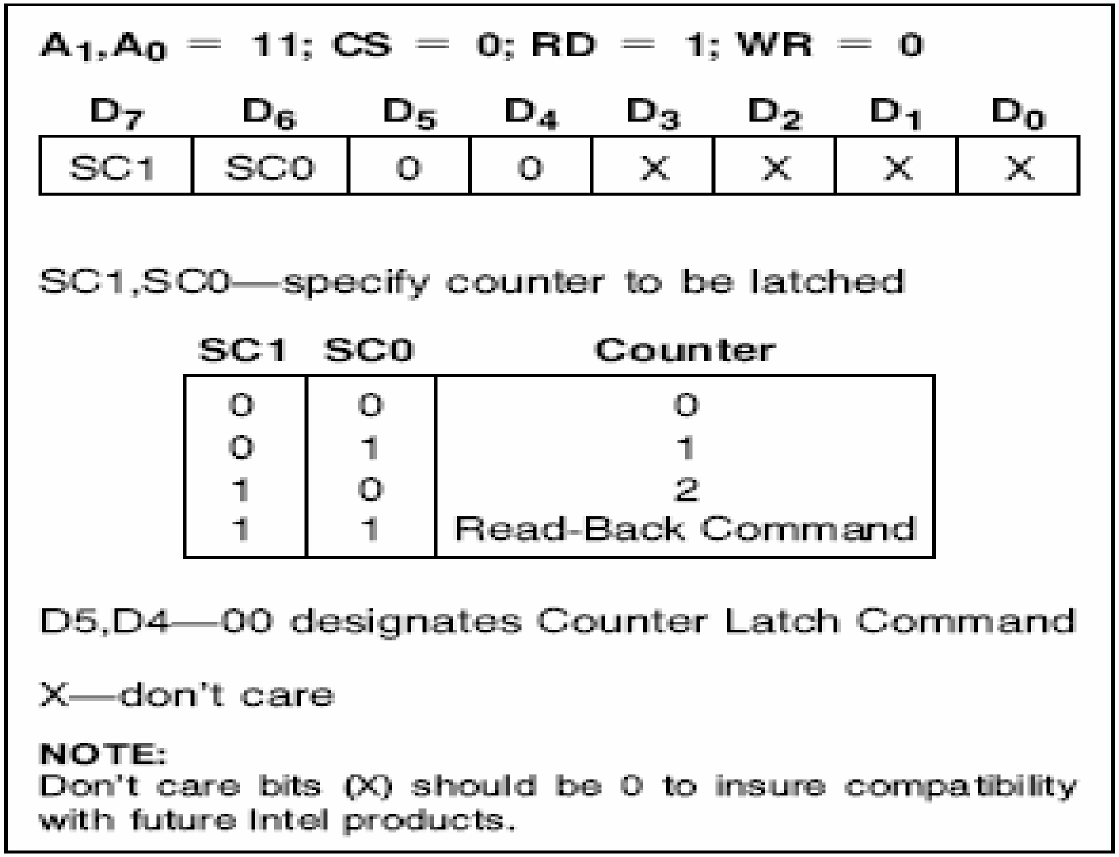
	A_1	A_0
Control Word—Counter 0	1	1
Control Word—Counter 1	1	1
Control Word—Counter 2	1	1
LSB of count—Counter 2	1	0
LSB of count—Counter 1	0	1
LSB of count—Counter 0	0	0
MSB of count—Counter 0	0	0
MSB of count—Counter 1	0	1
MSB of count—Counter 2	1	0

	A_1	A_0
Control Word—Counter 1	1	1
Control Word—Counter 0	1	1
LSB of count—Counter 1	0	1
Control Word—Counter 2	1	1
LSB of count—Counter 0	0	0
MSB of count—Counter 1	0	1
LSB of count—Counter 2	1	0
MSB of count—Counter 0	0	0
MSB of count—Counter 2	1	0

Figure 8. A Few Possible Programming Sequences

- **Read Operations**: It is often desirable to read the value of a Counter without disturbing the count in progress. This is easily done in the 8254.
- There are three possible methods for reading the counters: a simple read operation, the Counter Latch Command, and the Read-Back Command.
- Each is explained below. The first method is to perform a simple read operation. To read the Counter, which is selected with the A_1, A_0 inputs, the CLK input of the selected Counter must be inhibited by using either the GATE input or external logic.
- Otherwise, the count may be in the process of changing when it is read, giving an undefined result.

- **COUNTER LATCH COMMAND**: The second method uses the ``Counter Latch Command".
- Like a Control Word, this command is written to the Control Word Register, which is selected when A_1,A_0 = 11. Also like a Control Word, the SC0, SC1 bits select one of the three Counters, but two other bits, D5 and D4, distinguish this command from a Control Word.
- The selected Counter's output latch (OL) latches the count at the time the Counter Latch Command is received. This count is held in the latch until it is read by the CPU (or until the Counter is reprogrammed).



A_1,A_0 = 11; CS = 0; RD = 1; WR = 0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	0	0	X	X	X	X

SC1,SC0—specify counter to be latched

SC1	SC0	Counter
0	0	0
0	1	1
1	0	2
1	1	Read-Back Command

D5,D4—00 designates Counter Latch Command

X—don't care

NOTE:
Don't care bits (X) should be 0 to insure compatibility with future Intel products.

Figure 9. Counter Latching Command Format

- The count is then unlatched automatically and the OL returns to ``following" the counting element (CE).
- This allows reading the contents of the Counters ``on the fly" without affecting counting in progress.
- Multiple Counter Latch Commands may be used to latch more than one Counter. Each latched Counter's OL holds its count until it is read.
- Counter Latch Commands do not affect the programmed Mode of the Counter in any way.

- If a Counter is latched and then, some time later, latched again before the count is read, the second Counter Latch Command is ignored. The count read will be the count at the time the first Counter Latch Command was issued.
- With either method, the count must be read according to the programmed format; specifically, if the Counter is programmed for two byte counts, two bytes must be read. The two bytes do not have to be read one right after the other, read or write or programming operations of other Counters may be inserted between them.

- Another feature of the 8254 is that reads and writes of the same Counter may be interleaved.
- **Example**: If the Counter is programmed for two byte counts, the following sequence is valid.

1) Read least significant byte.

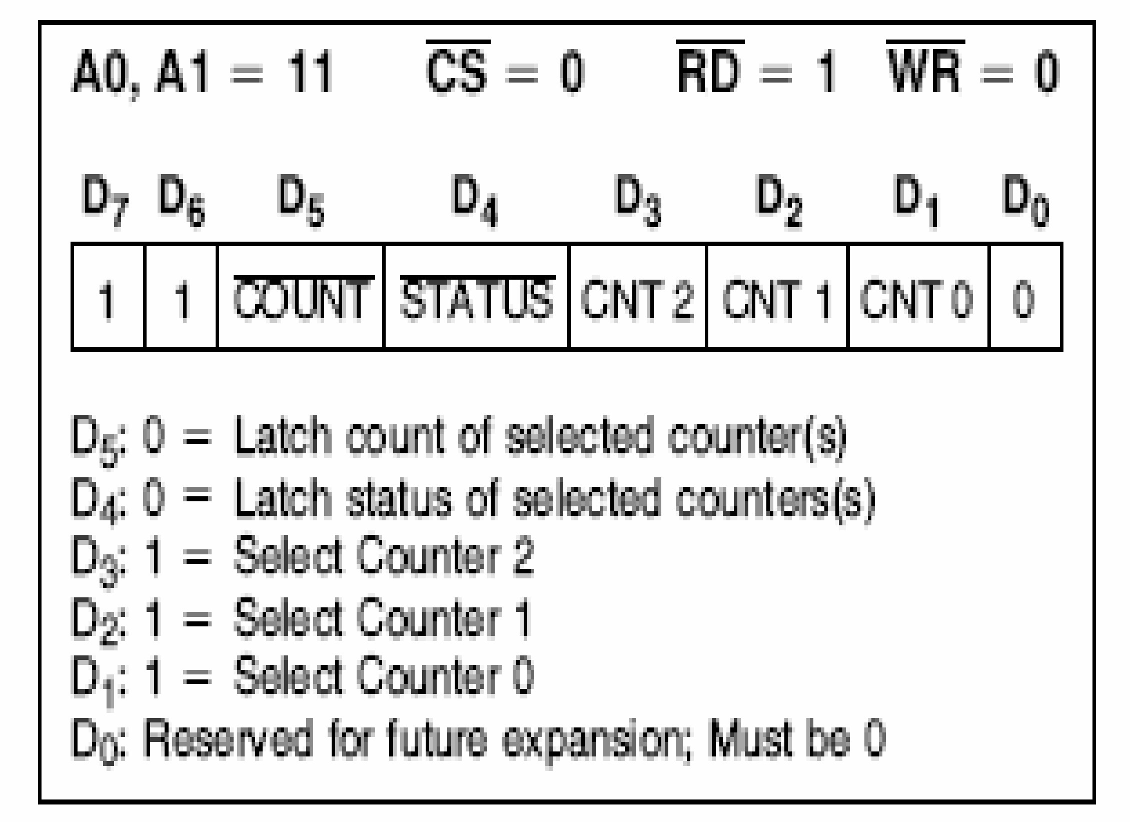
2) Write new least significant byte.

3) Read most significant byte.

4) Write new most significant byte.

- If a Counter is programmed to read/write two-byte counts, the following precaution applies: A program must not transfer control between reading the first and second byte to another routine which also reads from that same Counter. Otherwise, an incorrect count will be read.
- **READ-BACK COMMAND:** The third method uses the Read-Back Command. This command allows the user to check the count value, programmed Mode, and current states of the OUT pin and Null Count flag of the selected counter (s).

- The command is written into the Control Word Register and has the format shown in Figure 10. The command applies to the counters selected by setting their corresponding bits D_3, D_2, D_1 = 1.
- The read-back command may be used to latch multiple counter output latches (OL) by setting the COUNT bit D_5 = 0 and selecting the desired counter (s). This single command is functionally equivalent to several counter latch commands, one for each counter latched.



A0, A1 = 11 $\overline{CS}$ = 0 $\overline{RD}$ = 1 $\overline{WR}$ = 0

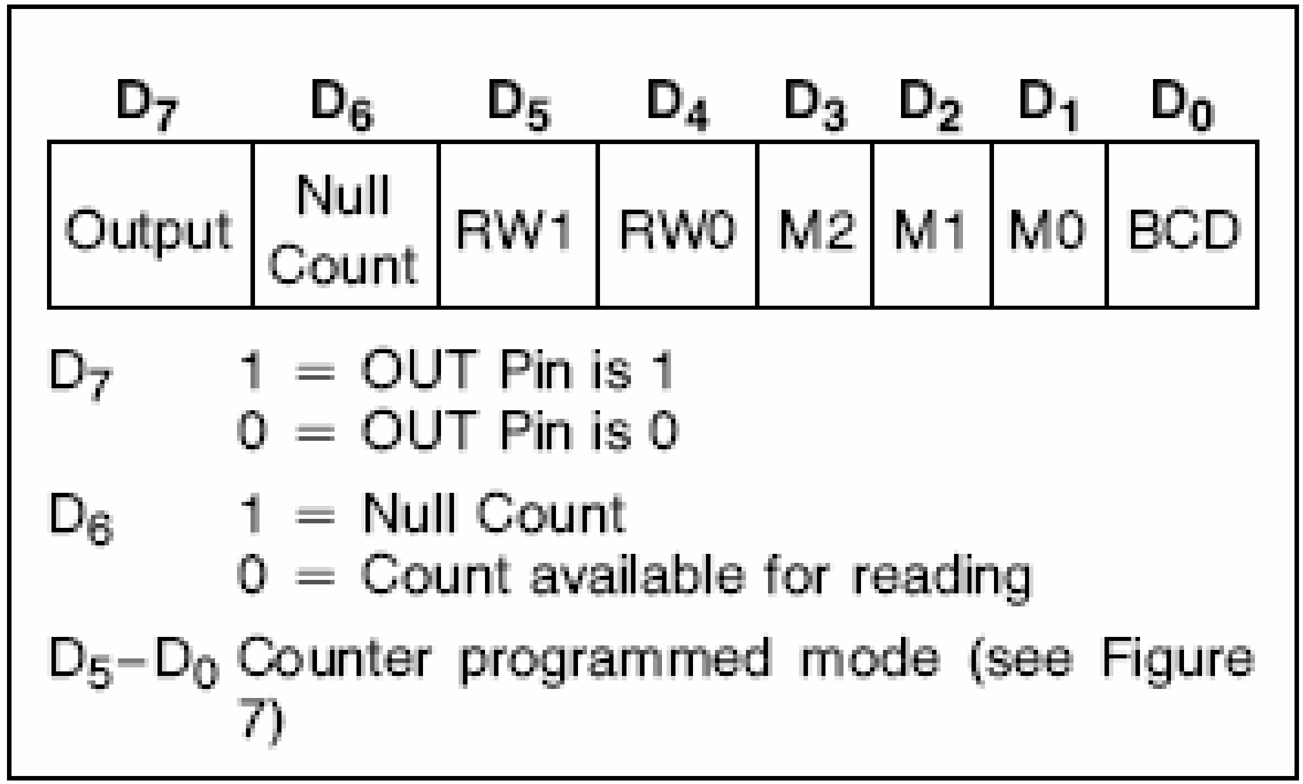
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	$\overline{\text{COUNT}}$	$\overline{\text{STATUS}}$	CNT 2	CNT 1	CNT 0	0

D_5: 0 = Latch count of selected counter(s)
D_4: 0 = Latch status of selected counters(s)
D_3: 1 = Select Counter 2
D_2: 1 = Select Counter 1
D_1: 1 = Select Counter 0
D_0: Reserved for future expansion; Must be 0

Figure 10. Read-Back Command Format

- Each counter's latched count is held until it is read (or the counter is reprogrammed).
- The counter is automatically unlatched when read, but other counters remain latched until they are read. If multiple count read-back commands are issued to the same counter without reading the count, all but the first are ignored; i.e., the count which will be read is the count at the time the first read-back command was issued.
- The read-back command may also be used to latch status information of selected counter (s) by setting STATUS bit D_4 = 0. Status must be latched to be read; status of a counter is accessed by a read from that counter.

- The counter status format is shown in Figure 11.
- Bits D_5 through D_0 contain the counter's programmed Mode exactly as written in the last Mode Control Word. OUTPUT bit D_7 contains the current state of the OUT pin.
- This allows the user to monitor the counter's output via software, possibly eliminating some hardware from a system. NULL COUNT bit D_6 indicates when the last count written to the counter register (CR) has been loaded into the counting element (CE).
- The exact time this happens depends on the Mode of the counter and is described in the Mode Definitions, but until the count is loaded into the counting element (CE), it can't be read from the counter.



D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Output	Null Count	RW1	RW0	M2	M1	M0	BCD

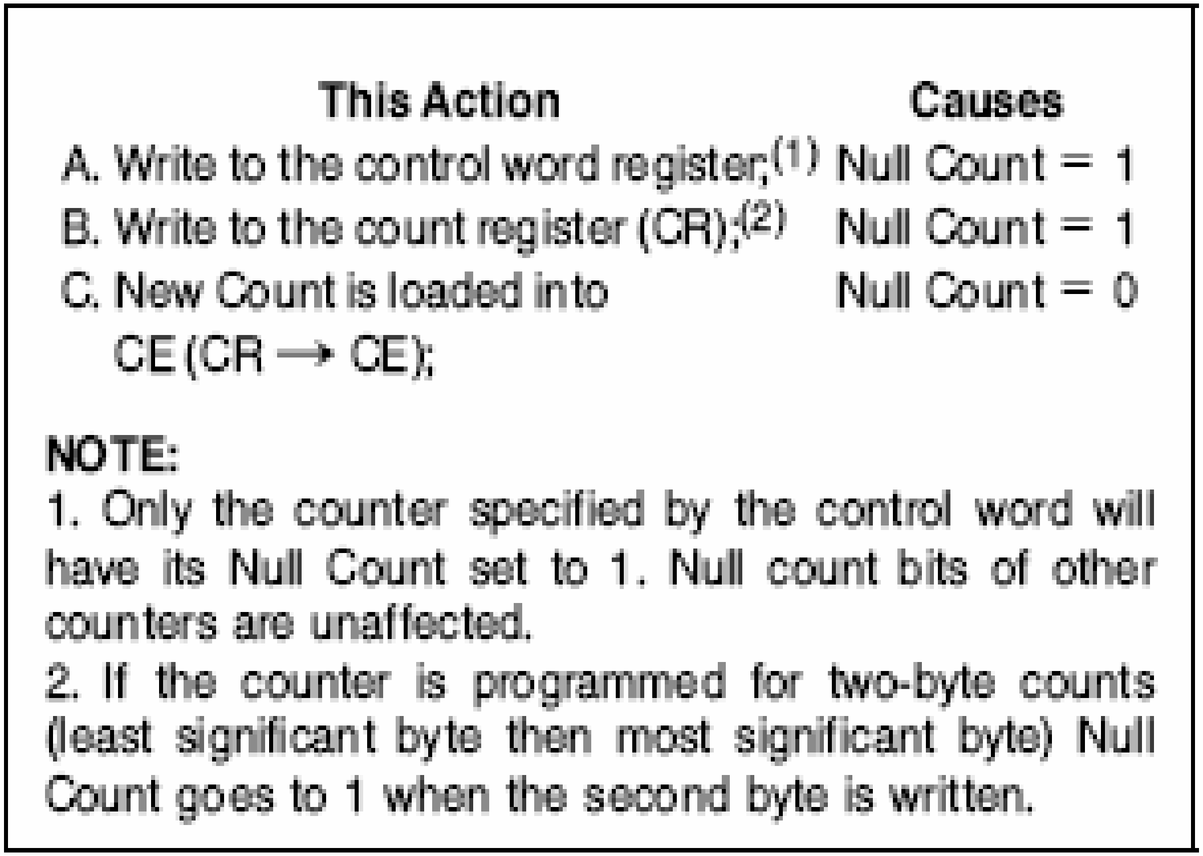
D_7 1 = OUT Pin is 1
0 = OUT Pin is 0

D_6 1 = Null Count
0 = Count available for reading

D_5–D_0 Counter programmed mode (see Figure 7)

Figure 11. Status Byte

- If the count is latched or read before this time, the count value will not reflect the new count just written. The operation of Null Count is shown in Figure 12.
- If multiple status latch operations of the counter (s) are performed without reading the status, all but the first are ignored; i.e., the status that will be read is the status of the counter at the time the first status read-back command was issued.
- Both count and status of the selected counter (s) may be latched simultaneously by setting both COUNT and STATUS bits D_5,D_4 = 0. This is functionally the same as issuing two separate read-back commands at once, and the above discussions apply here also.



This Action	Causes
A. Write to the control word register;[1]	Null Count = 1
B. Write to the count register (CR);[2]	Null Count = 1
C. New Count is loaded into CE (CR → CE);	Null Count = 0

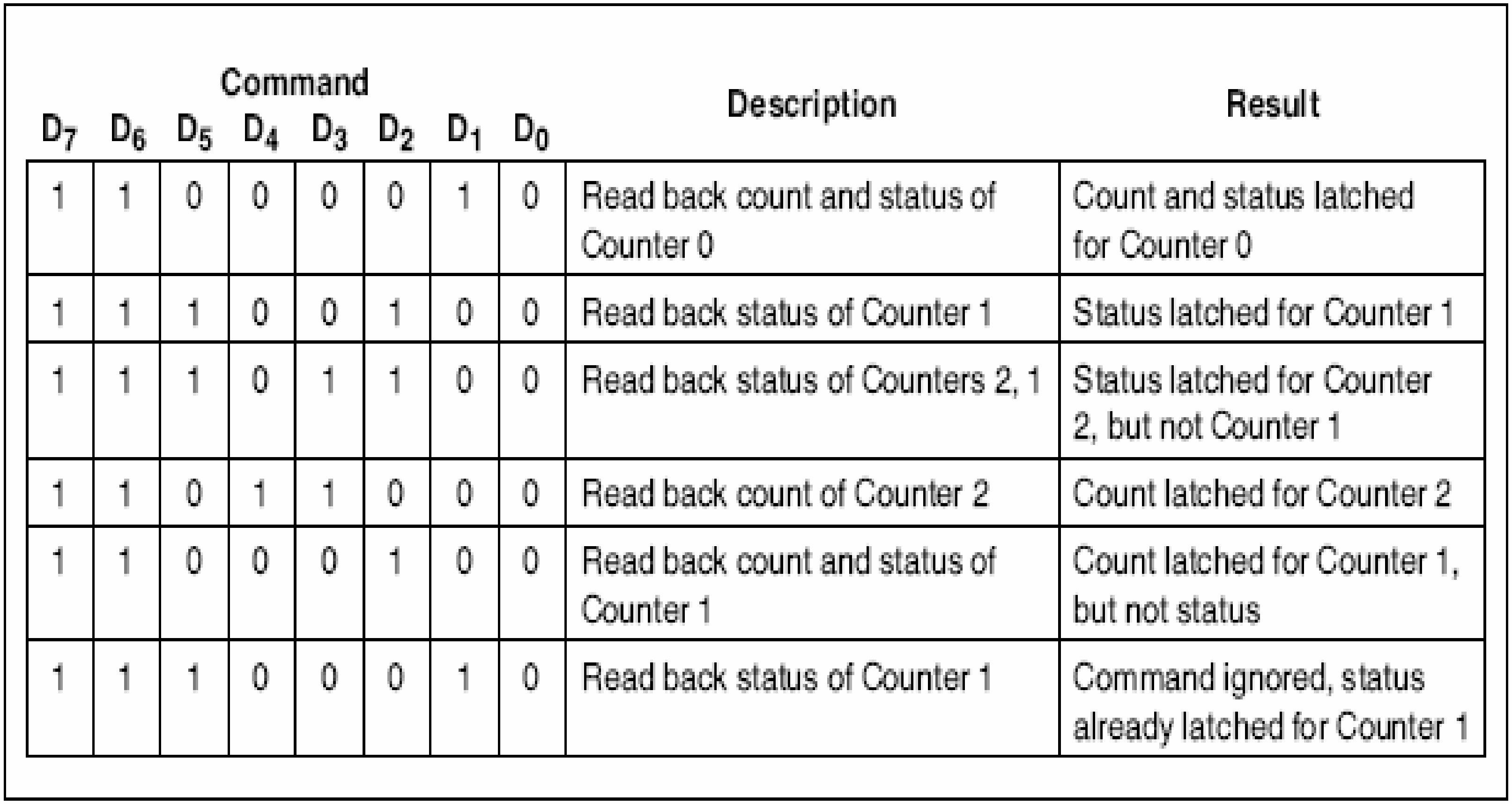
NOTE:

1. Only the counter specified by the control word will have its Null Count set to 1. Null count bits of other counters are unaffected.

2. If the counter is programmed for two-byte counts (least significant byte then most significant byte) Null Count goes to 1 when the second byte is written.

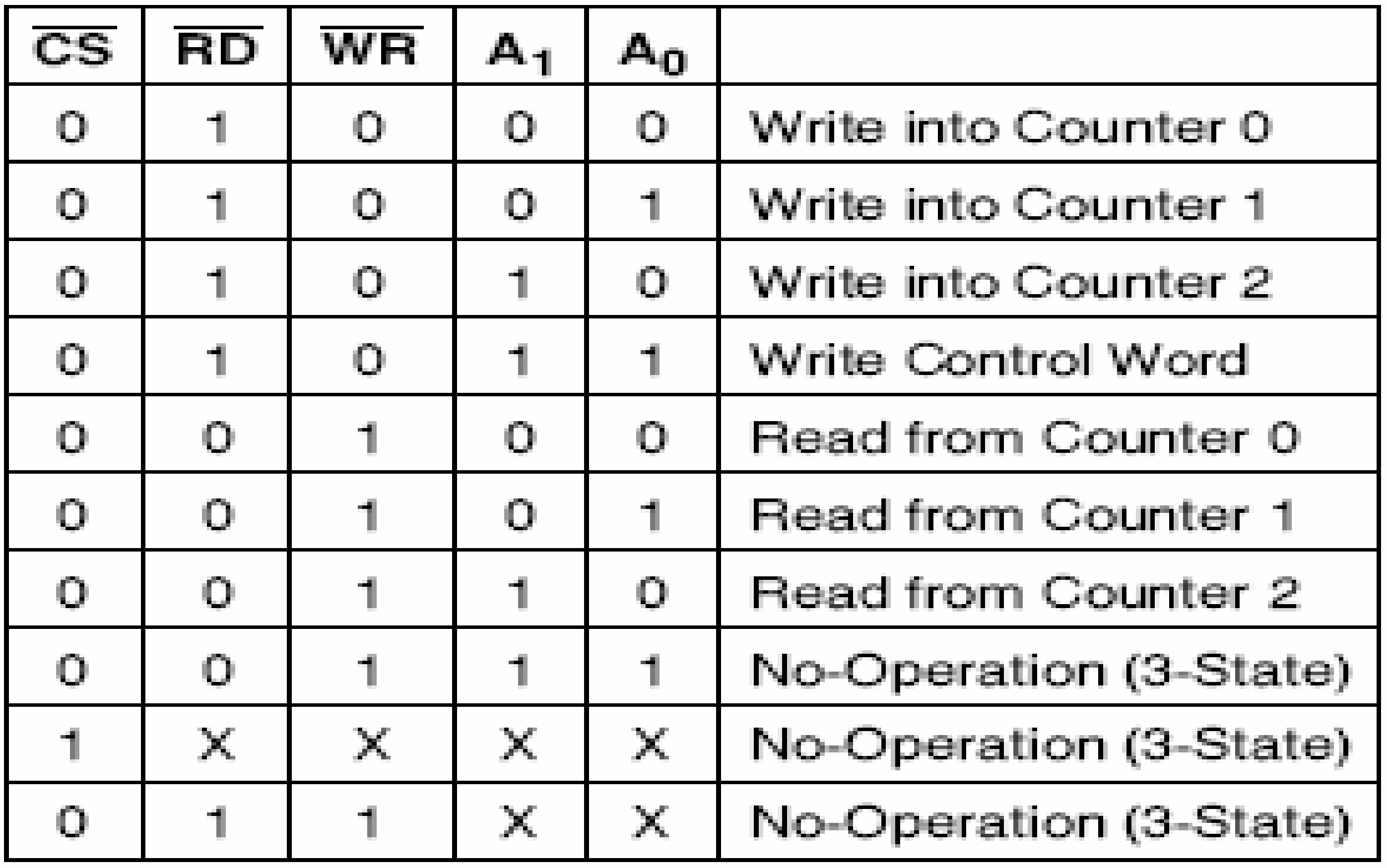
Figure 12. Null Count Operation

- Specifically, if multiple count and/or status read-back commands are issued to the same counter (s) without any intervening reads, all but the first are ignored. This is illustrated in Figure 13.
- If both count and status of a counter are latched, the first read operation of that counter will return latched status, regardless of which was latched first. The next one or two reads (depending on whether the counter is programmed for one or two type counts) return latched count. Subsequent reads return unlatched count.



Command								Description	Result
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0		
1	1	0	0	0	0	1	0	Read back count and status of Counter 0	Count and status latched for Counter 0
1	1	1	0	0	1	0	0	Read back status of Counter 1	Status latched for Counter 1
1	1	1	0	1	1	0	0	Read back status of Counters 2, 1	Status latched for Counter 2, but not Counter 1
1	1	0	1	1	0	0	0	Read back count of Counter 2	Count latched for Counter 2
1	1	0	0	0	1	0	0	Read back count and status of Counter 1	Count latched for Counter 1, but not status
1	1	1	0	0	0	1	0	Read back status of Counter 1	Command ignored, status already latched for Counter 1

Figure 13. Read-Back Command Example



$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_1	A_0	
0	1	0	0	0	Write into Counter 0
0	1	0	0	1	Write into Counter 1
0	1	0	1	0	Write into Counter 2
0	1	0	1	1	Write Control Word
0	0	1	0	0	Read from Counter 0
0	0	1	0	1	Read from Counter 1
0	0	1	1	0	Read from Counter 2
0	0	1	1	1	No-Operation (3-State)
1	X	X	X	X	No-Operation (3-State)
0	1	1	X	X	No-Operation (3-State)

Figure 14. Read/Write Operations Summary

- **Mode Definitions** :The following are defined for use in describing the operation of the 8254.
- **CLK Pulse**: A rising edge, then a falling edge, in that order, of a Counter's CLK input.
- **Trigger**: A rising edge of a Counter's GATE input.
- **Counter loading**: The transfer of a count from the CR to the CE (refer to the ``Functional Description'').
- **MODE 0**: ***INTERRUPT ON TERMINAL COUNT*** : Mode 0 is typically used for event counting. After the Control Word is written, OUT is initially low, and will remain low until the Counter reaches zero.

- OUT then goes high and remains high until a new count or a new Mode 0 Control Word is written into the Counter.
- GATE = 1 enables counting; GATE = 0 disables counting. GATE has no effect on OUT.
- After the Control Word and initial count are written to a Counter, the initial count will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not go high until N a 1 CLK pulses after the initial count is written.
- If a new count is written to the Counter, it will be loaded on the next CLK pulse and counting will continue from the new count. If a two-byte count is written, the following happens:

1) Writing the first byte disables counting. OUT is set low immediately (no clock pulse required).

2) Writing the second byte allows the new count to be loaded on the next CLK pulse.

- This allows the counting sequence to be synchronized by software. Again, OUT does not go high until Na1 CLK pulses after the new count of N is written.
- If an initial count is written while GATE e 0, it will still be loaded on the next CLK pulse. When GATE goes high, OUT will go high N CLK pulses later; no CLK pulse is needed to load the Counter as this has already been done.

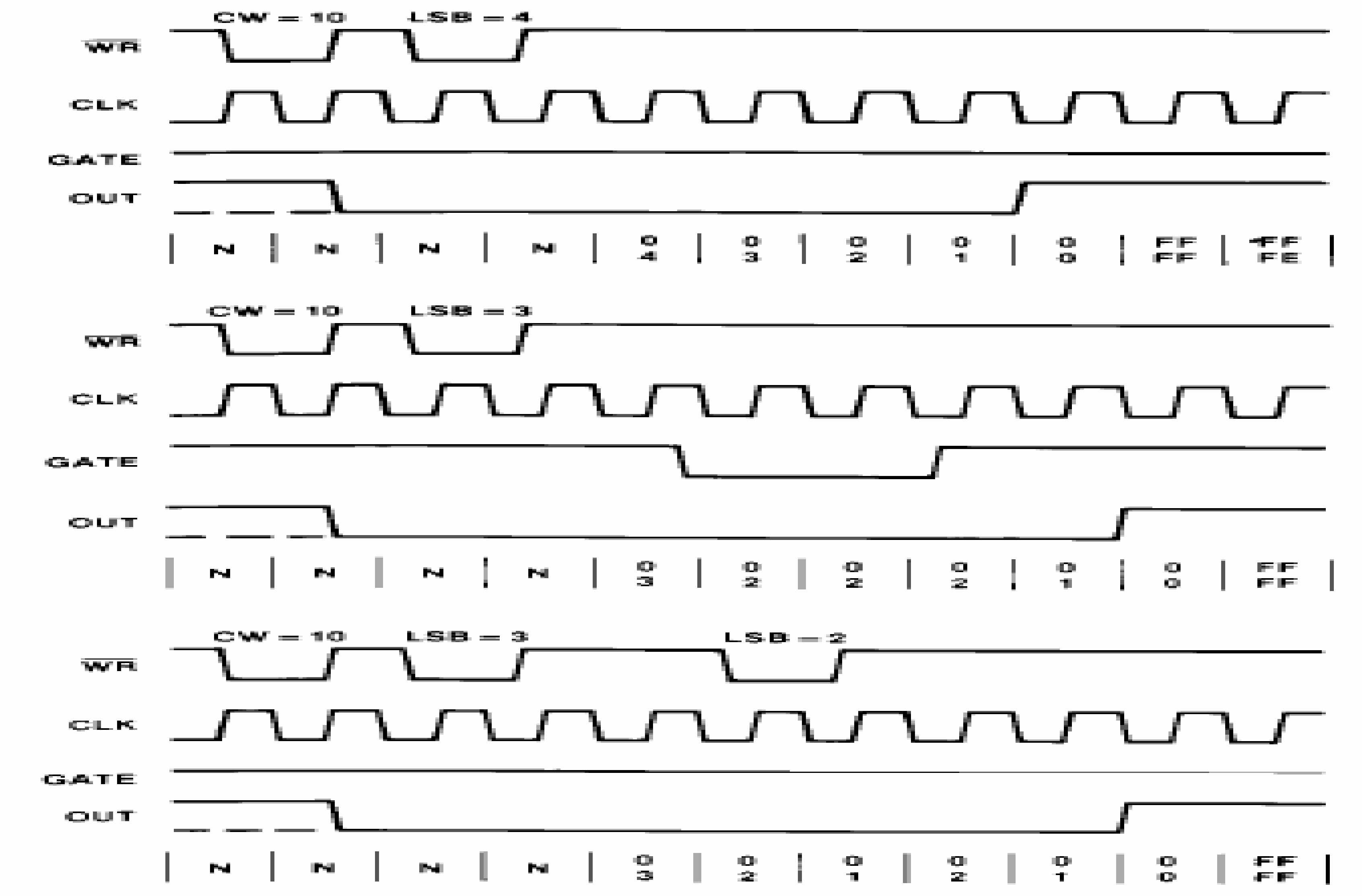


Figure 15. Mode 0

Note:

1. Counters are programmed for binary (not BCD) counting and for reading/writing least significant byte (LSB) only.
2. The counter is always selected (CS always low).
3. CW stands for ``Control Word''; CW = 10 means a control word of 10 HEX is written to the counter.
4. LSB stands for ``Least Significant Byte'' of count.
5. Numbers below diagrams are count values. The lower number is the least significant byte. The upper number is the most significant byte. Since the counter is programmed to read/write LSB only, the most significant byte cannot be read. N stands for an undefined count. Vertical lines show transitions between count values.

- **MODE 1**: ***HARDWARE RETRIGGERABLE ONE-SHOT***: OUT will be initially high.
- OUT will go low on the CLK pulse following a trigger to begin the one-shot pulse, and will remain low until the Counter reaches zero.
- OUT will then go high and remain high until the CLK pulse after the next trigger.
- After writing the Control Word and initial count, the Counter is armed. A trigger results in loading the Counter and setting OUT low on the next CLK pulse, thus starting the one-shot pulse. An initial count of N will result in a one-shot pulse N CLK cycles in duration.

- The one-shot is retriggerable, hence OUT will remain low for N CLK pulses after any trigger. The one-shot pulse can be repeated without rewriting the same count into the counter. GATE has no effect on OUT.
- If a new count is written to the Counter during a oneshot pulse, the current one-shot is not affected unless the counter is retriggered. In that case, the Counter is loaded with the new count and the oneshot pulse continues until the new count expires.

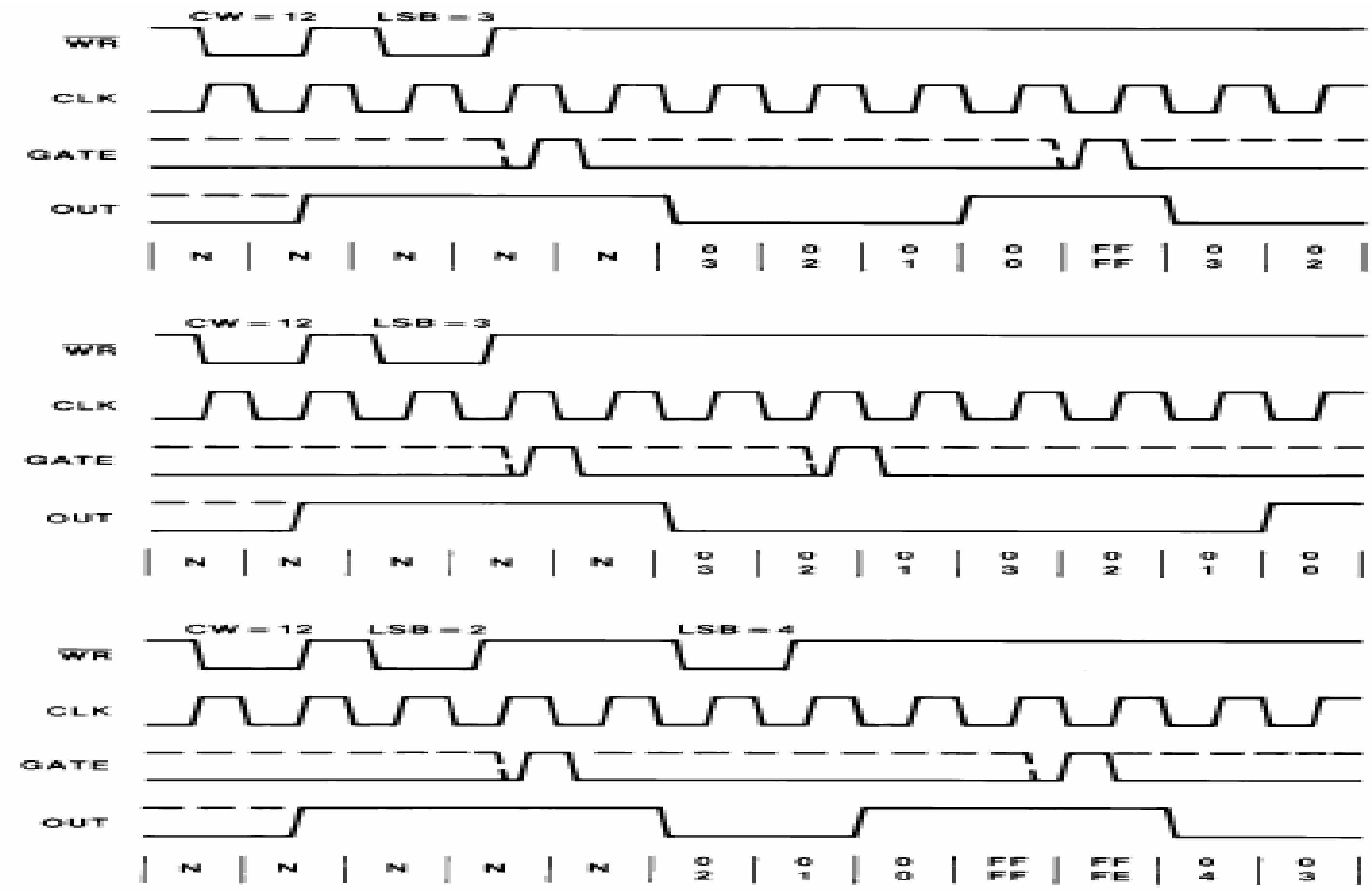


Figure 16. Mode 1

- **MODE 2**: RATE GENERATOR: This Mode functions like a divide-by-N counter. It is typically used to generate a Real Time Clock interrupt.
- OUT will initially be high. When the initial count has decremented to 1, OUT goes low for one CLK pulse. OUT then goes high again, the Counter reloads the initial count and the process is repeated.
- Mode 2 is periodic, the same sequence is repeated indefinitely. For an initial count of N, the sequence repeats every N CLK cycles.
- GATE = 1 enables counting; GATE = 0 disables counting. If GATE goes low during an output pulse, OUT is set high immediately.

- A trigger reloads the Counter with the initial count on the next CLK pulse, OUT goes low N CLK pulses after the trigger. Thus the GATE input can be used to synchronize the Counter.
- After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. OUT goes low N CLK Pulses after the initial count is written.
- This allows the Counter to be synchronized by software also. Writing a new count while counting does not affect the current counting sequence.

- If a trigger is received after writing a new count but before the end of the current period, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from the new count.
- Otherwise, the new count will be loaded at the end of the current counting cycle. In mode 2, a COUNT of 1 is illegal.
- **MODE 3**: SQUARE WAVE MODE :Mode 3 is typically used for Baud rate generation. Mode 3 is similar to Mode 2 except for the duty cycle of OUT. OUT will initially be high.

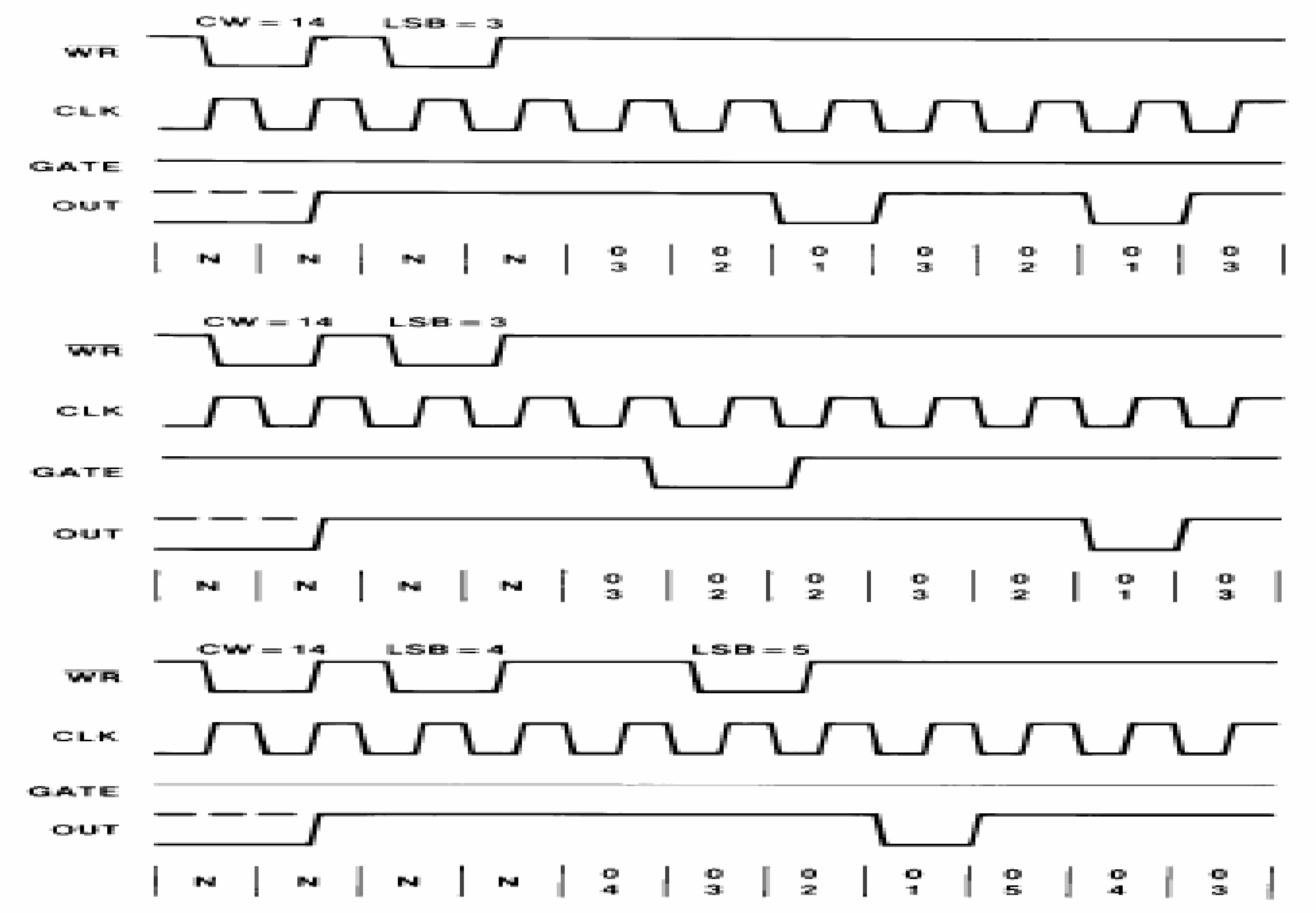


Figure 17. Mode 2

- When half the initial count has expired, OUT goes low for the remainder of the count. Mode 3 is periodic; the sequence above is repeated indefinitely.
- An initial count of N results in a square wave with a period of N CLK cycles. GATE = 1 enables counting; GATE = 0 disables counting. If GATE goes low while OUT is low, OUT is set high immediately; no CLK pulse is required.
- A trigger reloads the Counter with the initial count on the next CLK pulse. Thus the GATE input can be used to synchronize the Counter.

- After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. This allows the Counter to be synchronized by software also.
- Writing a new count while counting does not affect the current counting sequence. If a trigger is received after writing a new count but before the end of the current half-cycle of the square wave, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from the new count. Otherwise, the new count will be loaded at the end of the current half-cycle.

- **Mode 3:Even counts**: OUT is initially high. The initial count is loaded on one CLK pulse and then is decremented by two on succeeding CLK pulses.
- When the count expires OUT changes value and the Counter is reloaded with the initial count. The above process is repeated indefinitely.
- **Odd counts**: OUT is initially high. The initial count minus one (an even number) is loaded on one CLK pulse and then is decremented by two on succeeding CLK pulses.

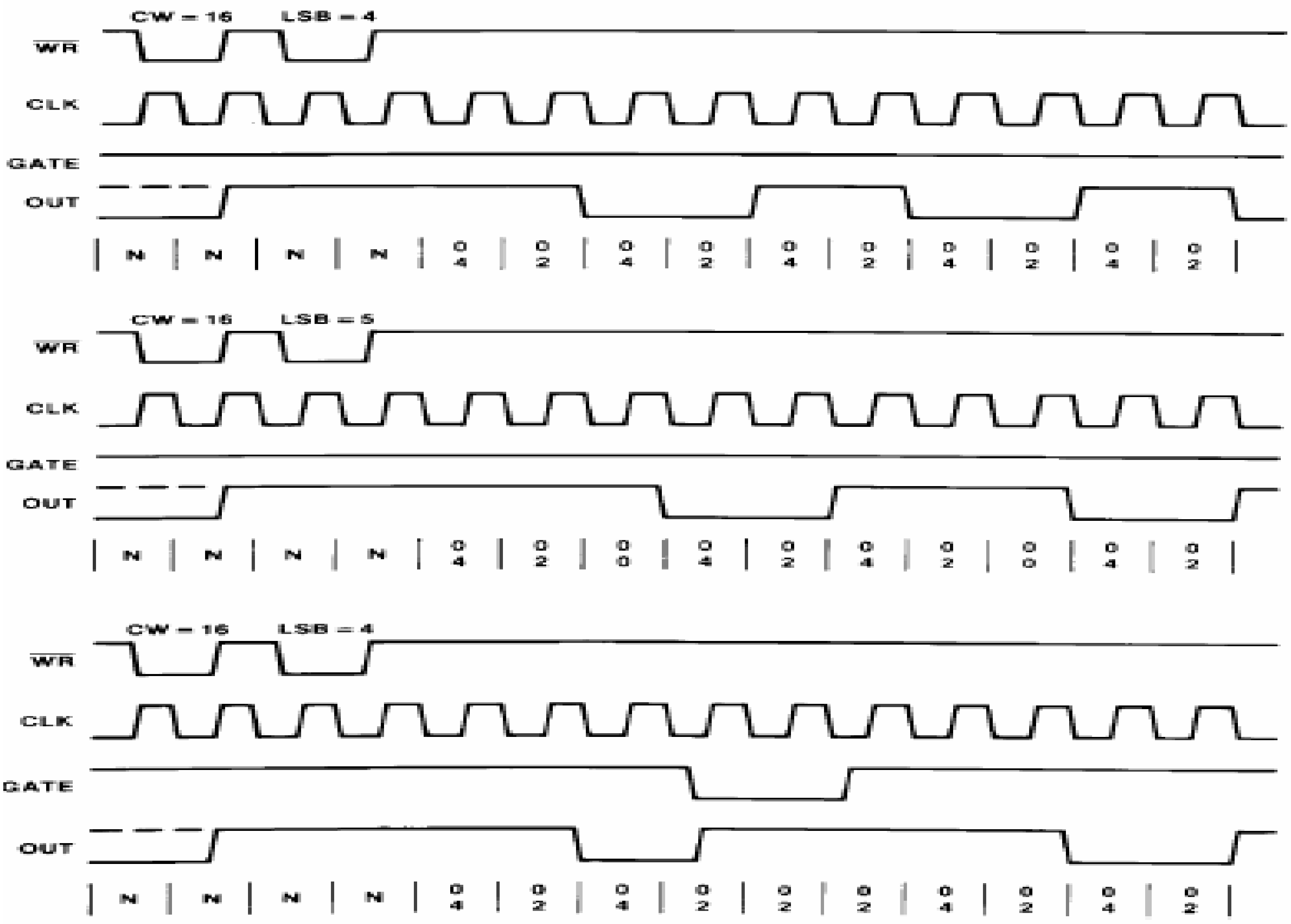

CW = 16
LSB = 4
WR
CLK
GATE
OUT
N N N N 4 2 4 2 4 2 4 2 4 2
CW = 16
LSB = 5
WR
CLK
GATE
OUT
N N N N 4 2 0 4 2 4 2 0 4 2
CW = 16
LSB = 4
WR
CLK
GATE
OUT
N N N N 4 2 4 2 2 2 4 2 4 2

Figure 18. Mode 3

- One CLK pulse after the count expires, OUT goes low and the Counter is reloaded with the initial count minus one.
- Succeeding CLK pulses decrement the count by two.
- When the count expires, OUT goes high again and the Counter is reloaded with the initial count minus one. The above process is repeated indefinitely.
- So for odd counts, OUT will be high for (N - 1)/2 counts and low for (N - 1)/2 counts.

- **MODE 4: SOFTWARE TRIGGERED STROBE :**
- OUT will be initially high. When the initial count expires, OUT will go low for one CLK pulse and then go high again. The counting sequence is ``triggered'‘ by writing the initial count.
- GATE = 1 enables counting; GATE = 0 disables counting. GATE has no effect on OUT. After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse.
- This CLK pulse does not decrement the count, so for an initial count of N, OUT does not strobe low until N + 1 CLK pulses after the initial count is written.

- If a new count is written during counting, it will be loaded on the next CLK pulse and counting will continue from the new count. If a two-byte count is written, the following happens:

1) Writing the first byte has no effect on counting.

2) Writing the second byte allows the new count to be loaded on the next CLK pulse.

- This allows the sequence to be ``retriggered" by software. OUT strobes low N a 1 CLK pulses after the new count of N is written.

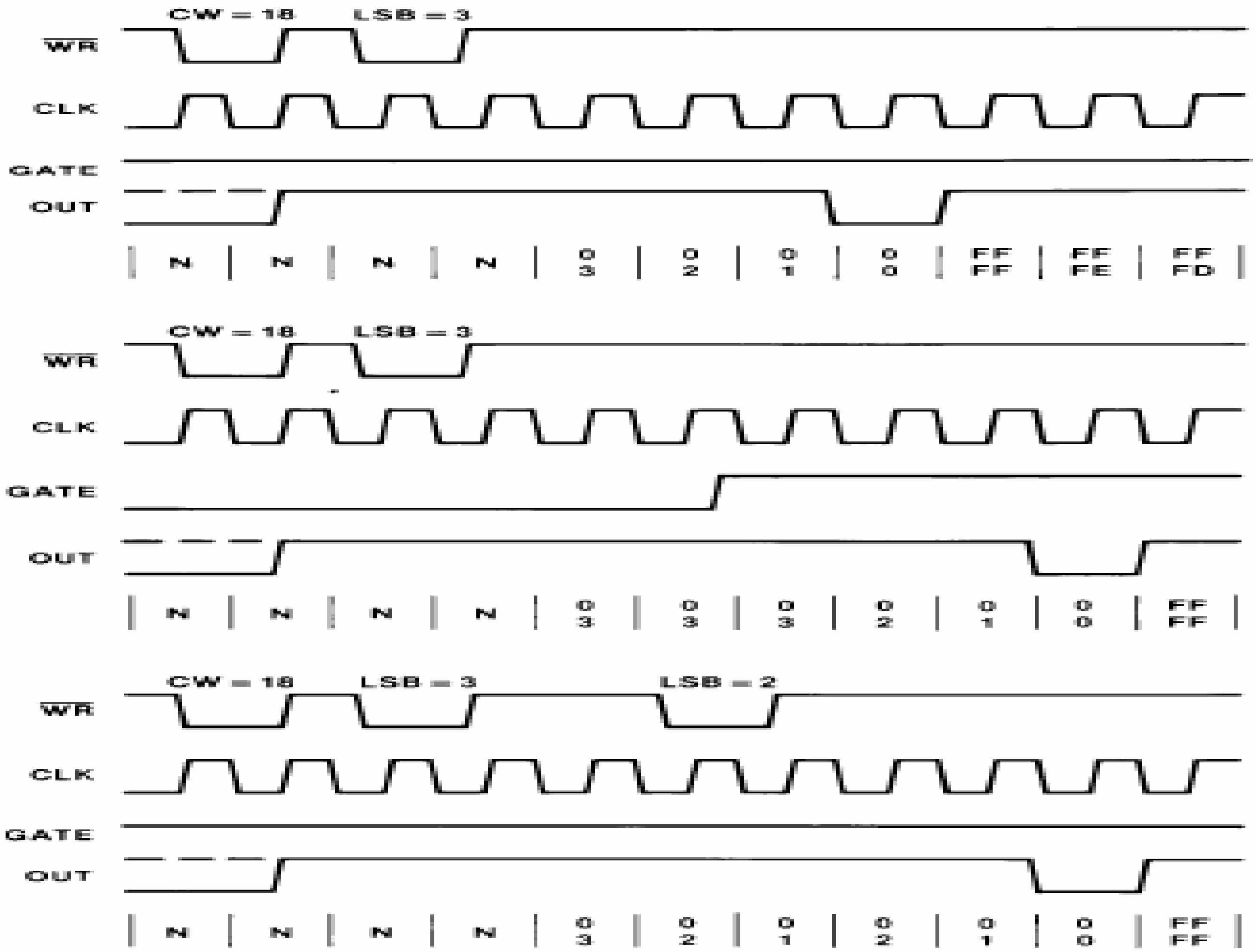


Figure 19. Mode 4

- **MODE 5: HARDWARE TRIGGERED STROBE (RETRIGGERABLE)** :OUT will initially be high. Counting is triggered by a rising edge of GATE. When the initial count has expired, OUT will go low for one CLK pulse and then go high again.
- After writing the Control Word and initial count, the counter will not be loaded until the CLK pulse after a trigger. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not strobe low until N = 1 CLK pulses after a trigger.

- A trigger results in the Counter being loaded with the initial count on the next CLK pulse. The counting sequence is retriggerable. OUT will not strobe low for N a 1 CLK pulses after any trigger. GATE has no effect on OUT.
- If a new count is written during counting, the current counting sequence will not be affected. If a trigger occurs after the new count is written but before the current count expires, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from there.

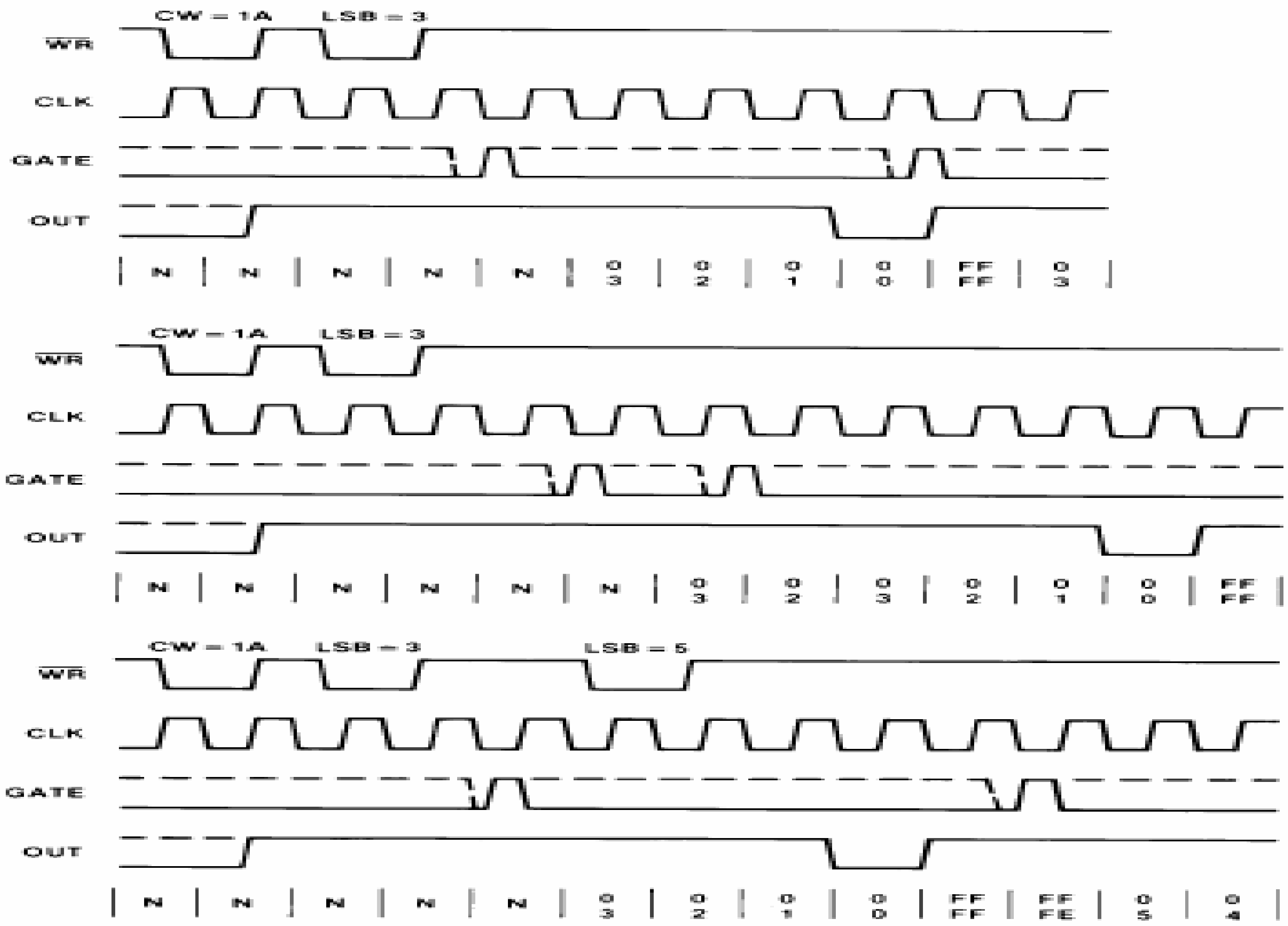
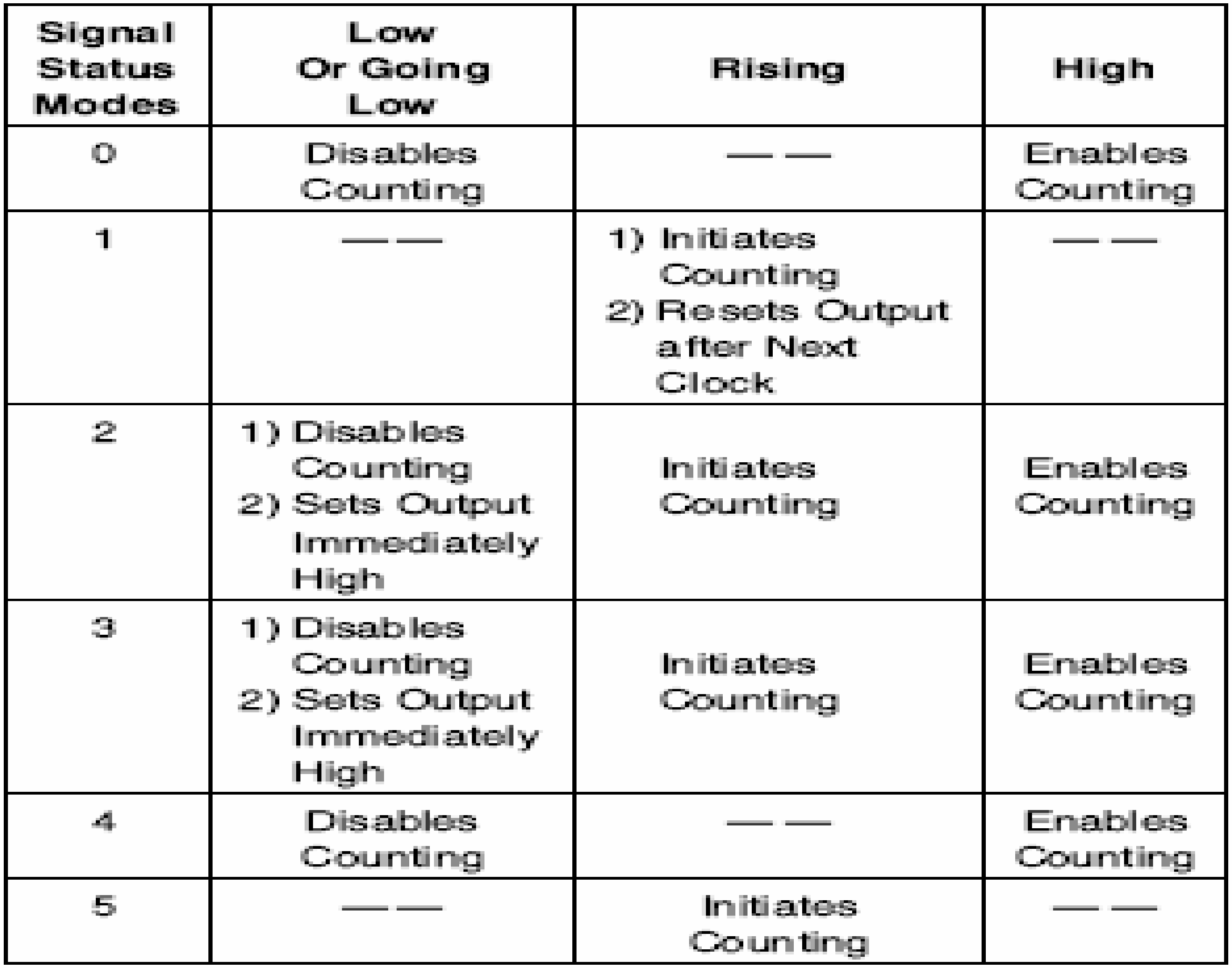


Figure 20. Mode 5

- **Operation Common to All Modes**:
- **PROGRAMMING:** When a Control Word is written to a Counter, all Control Logic is immediately reset and OUT goes to a known initial state; no CLK pulses are required for this.
- **GATE**: The GATE input is always sampled on the rising edge of CLK. In Modes 0, 2, 3, and 4 the GATE input is level sensitive, and the logic level is sampled on the rising edge of CLK. In Modes 1, 2, 3, and 5 the GATE input is rising-edge sensitive.

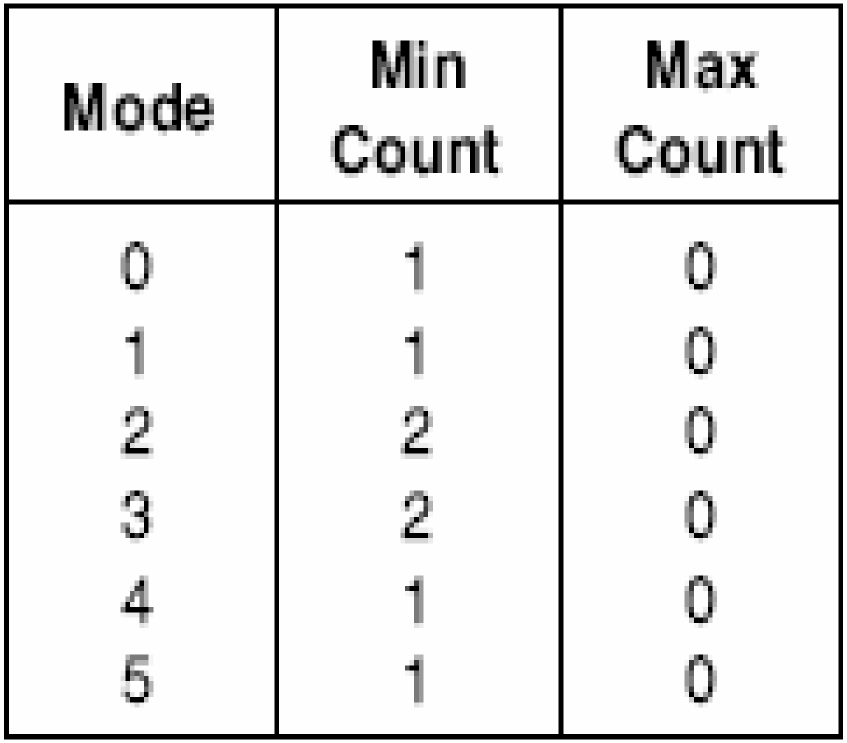
- In these Modes, a rising edge of GATE (trigger) sets an edge-sensitive flip-flop in the Counter. This flip-flop is then sampled on the next rising edge of CLK; the flip-flop is reset immediately after it is sampled. In this way, a trigger will be detected no matter when it occurs-a high logic level does not have to be maintained until the next rising edge of CLK.
- Note that in Modes 2 and 3, the GATE input is both edge- and level-sensitive. In Modes 2 and 3, if a CLK source other than the system clock is used, GATE should be pulsed immediately following WR of a new count value.



Signal Status Modes	Low Or Going Low	Rising	High
0	Disables Counting	— —	Enables Counting
1	— —	1) Initiates Counting 2) Resets Output after Next Clock	— —
2	1) Disables Counting 2) Sets Output Immediately High	Initiates Counting	Enables Counting
3	1) Disables Counting 2) Sets Output Immediately High	Initiates Counting	Enables Counting
4	Disables Counting	— —	Enables Counting
5	— —	Initiates Counting	— —

Figure 21. Gate Pin Operations Summary

- **COUNTER**: New counts are loaded and Counters are decremented on the falling edge of CLK.
- The largest possible initial count is 0, this is equivalent to $_{2}^{16}$ for binary counting and $_{10}^{4}$ for BCD counting. The Counter does not stop when it reaches zero.
- In Modes 0, 1, 4, and 5 the Counter ``wraps around" to the highest count, either FFFF hex for binary counting or 9999 for BCD counting, and continues counting.
- Modes 2 and 3 are periodic; the Counter reloads itself with the initial count and continues counting from there.



Mode	Min Count	Max Count
0	1	0
1	1	0
2	2	0
3	2	0
4	1	0
5	1	0

NOTE: 0 is equivalent to 2^{16} for binary counting and 10^{4} for BCD counting.

Figure 22. Minimum and Maximum Initial Counts

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.

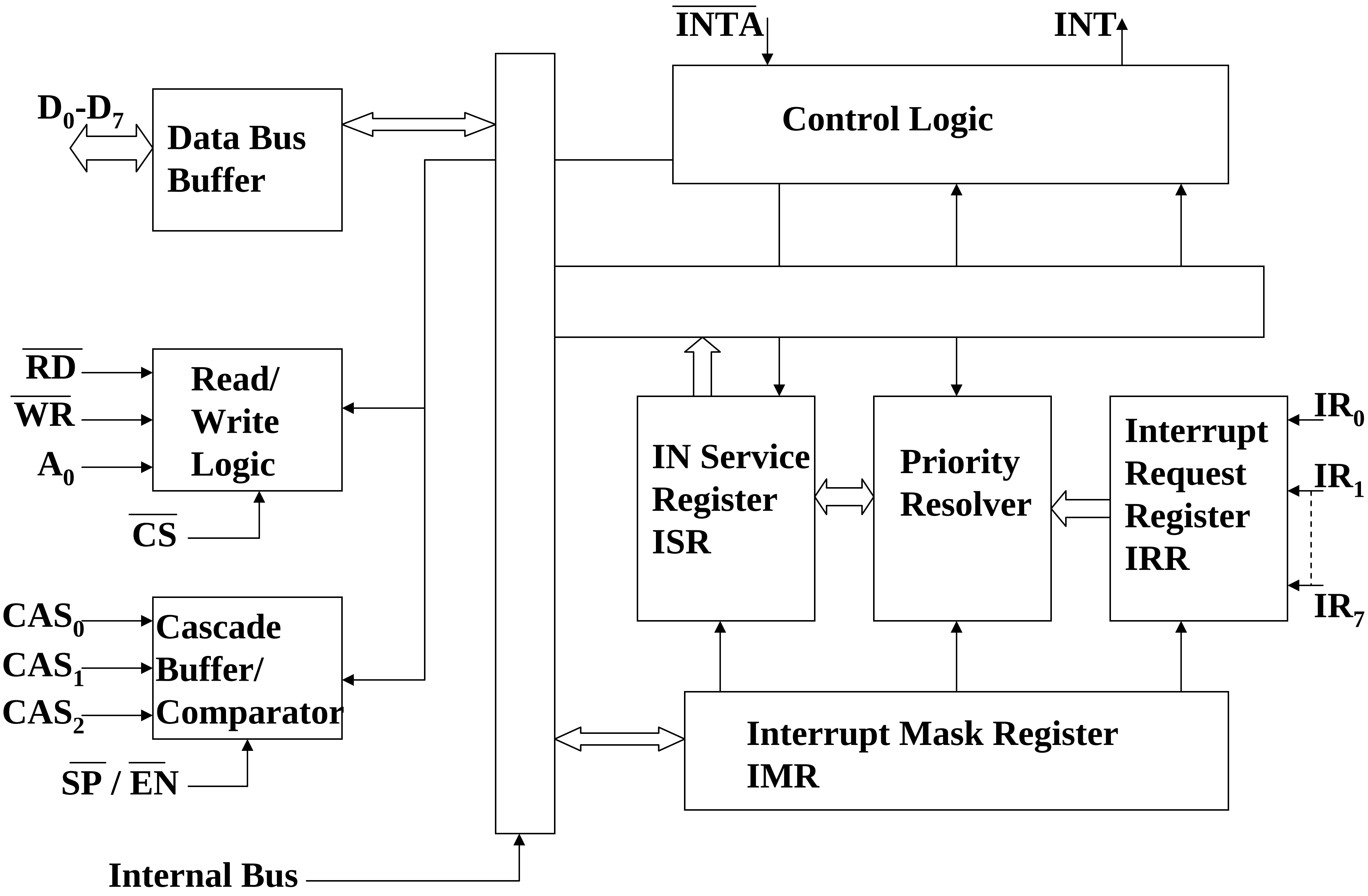


Fig:1 8259A Block Diagram

- **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 $\overline{\text{INTA}}$ pulse. The IR_0 has the highest priority while the IR_7 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 $\overline{\text{INTA}}$ pulse.

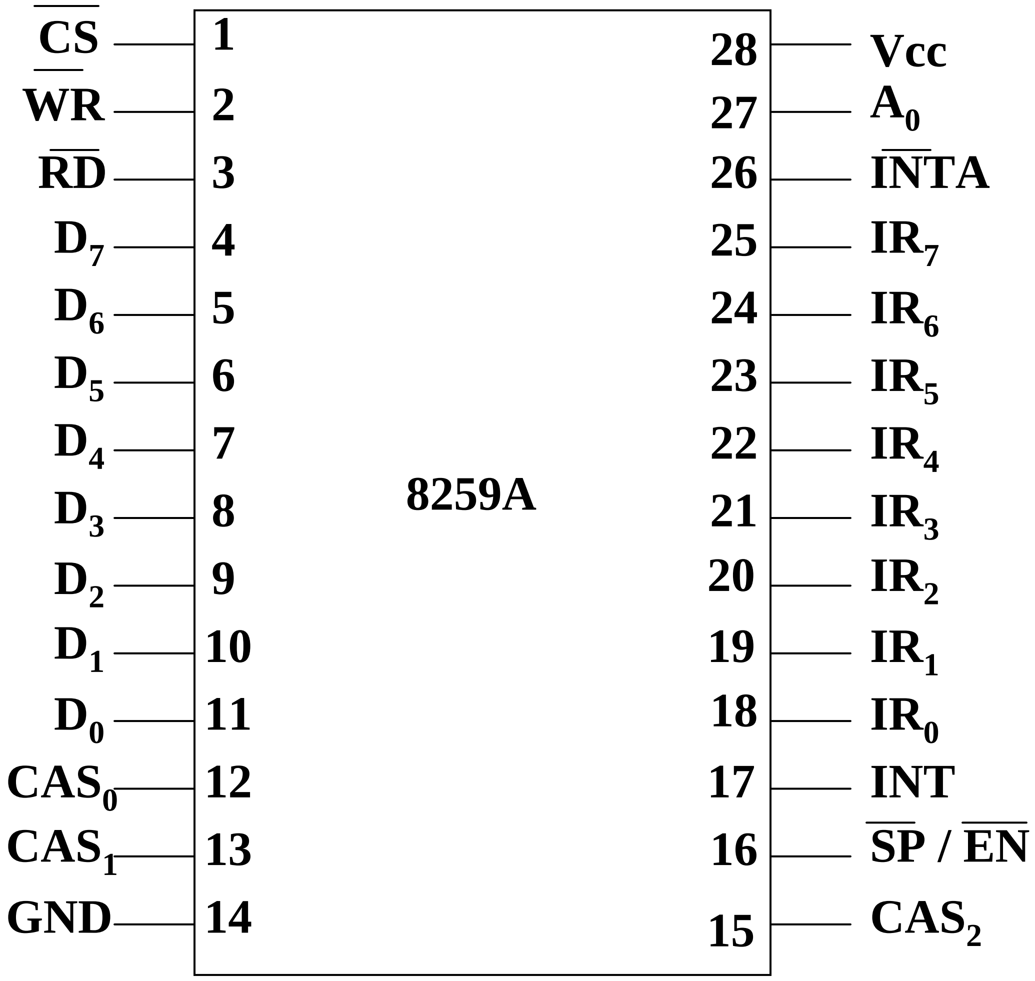


Fig : 8259 Pin Diagram

- **$\overline{\textbf{CS}}$**: This is an active-low chip select signal for enabling $\overline{\text{RD}}$ and $\overline{\text{WR}}$ operations of 8259A. $\overline{\text{INTA}}$ function is independent of $\overline{\text{CS}}$.
- **$\overline{\textbf{WR}}$** : This pin is an active-low write enable input to 8259A. This enables it to accept command words from CPU.
- **$\overline{\textbf{RD}}$** : 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.
- **$\mathbf{D_0}$-$\mathbf{D_7}$** : 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.

- **CAS_0 – CAS_2 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.
- **$\overline{\text{PS}}$/$\overline{\text{EN}}$** : 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 ($\overline{\text{SP}}$ =1) or slave ($\overline{\text{EN}}$ = 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.
- **IR_0 – IR_7 (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.

- **$\overline{\text{INTA}}$ (Interrupt acknowledge)**: This pin is an input used to strobe-in 8259A interrupt vector data on to the data bus. In conjunction with $\overline{\text{CS}}$, $\overline{\text{WR}}$ and $\overline{\text{RD}}$ 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 all the 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.

Interrupt Sequence in an 8086 system

- The Interrupt sequence in an 8086-8259A system is described as follows:

1. One or more IR lines are raised high that set corresponding IRR bits.
2. 8259A resolves priority and sends an INT signal to CPU.
3. The CPU acknowledge with INTA pulse.
4. Upon receiving an INTA signal from the CPU, the highest priority ISR bit is set and the corresponding IRR bit is reset. The 8259A does not drive data during this period.

5. The 8086 will initiate a second INTA pulse. During this period 8259A releases an 8-bit pointer on to a data bus from where it is read by the CPU.
6. This completes the interrupt cycle. The ISR bit is reset at the end of the second INTA pulse if automatic end of interrupt (AEOI) mode is programmed. Otherwise ISR bit remains set until an appropriate EOI command is issued at the end of interrupt subroutine.

Command Words of 8259A

- The command words of 8259A are classified in two groups

1. Initialization command words (ICW) and
2. Operation command words (OCW).

- Initialization Command Words (ICW): Before it starts functioning, the 8259A must be initialized by writing two to four command words into the respective command word registers. These are called as initialized command words.

- If A_0 = 0 and D_4 = 1, the control word is recognized as ICW_1. It contains the control bits for edge/level triggered mode, single/cascade mode, call address interval and whether ICW_4 is required or not.
- If A_0=1, the control word is recognized as ICW_2. The ICW_2 stores details regarding interrupt vector addresses. The initialisation sequence of 8259A is described in form of a flow chart in fig 3 below.
- The bit functions of the ICW_1 and ICW_2 are self explanatory as shown in fig below.

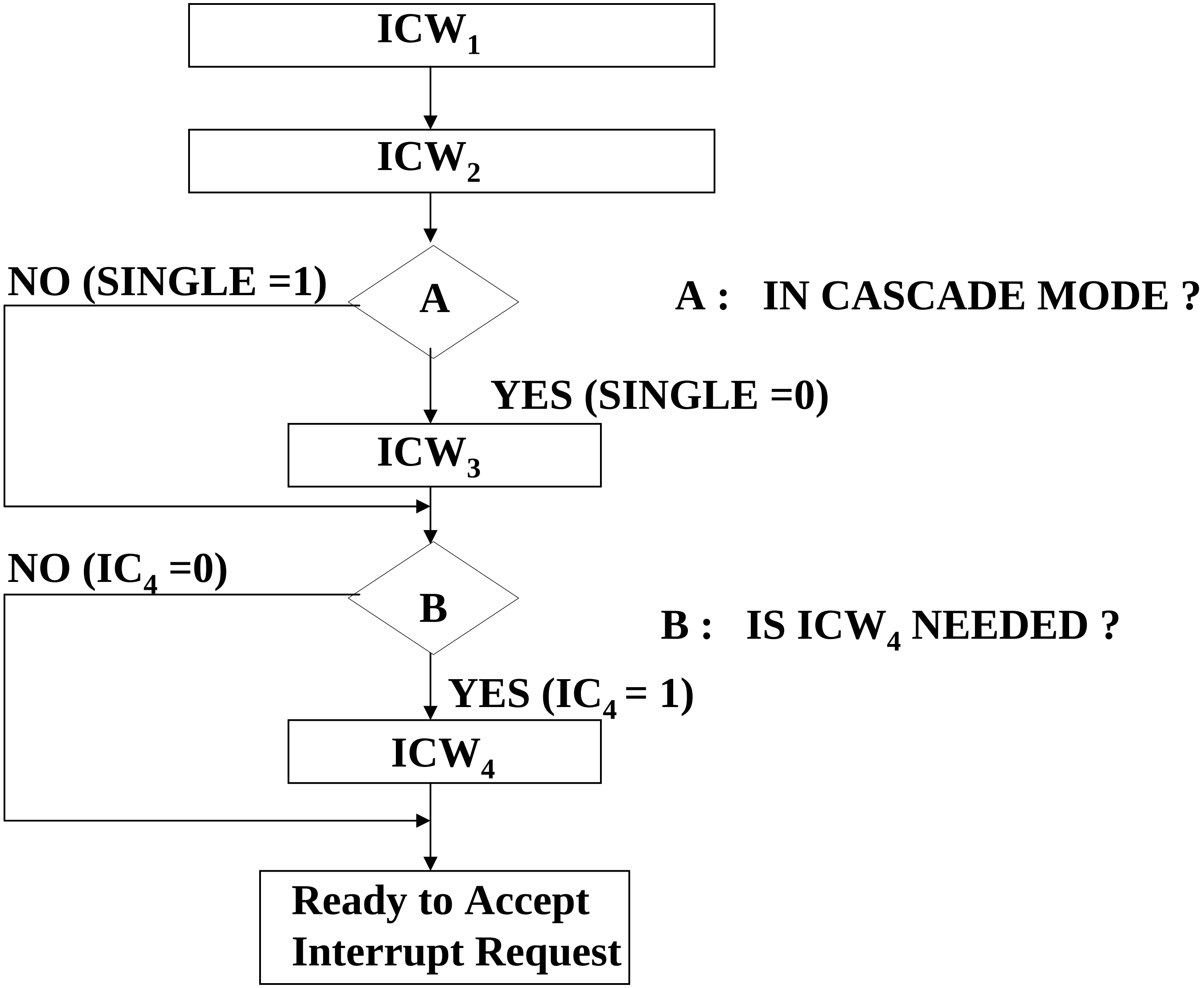
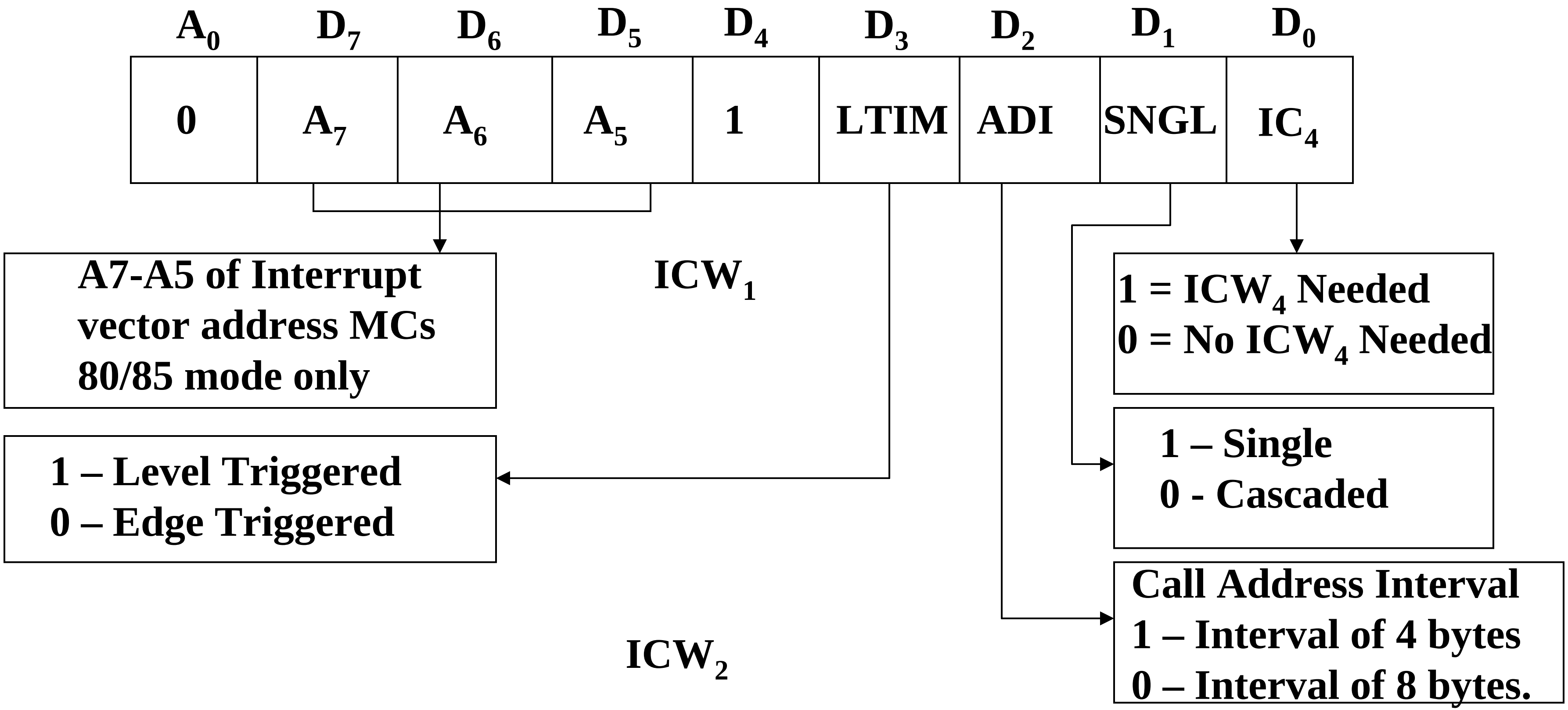


Fig 3: Initialisation Sequence of 8259A



A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	A_7	A_6	A_5	1	LTIM	ADI	SNGL	IC_4

ICW_1

A7-A5 of Interrupt
vector address MCs
80/85 mode only

1 – Level Triggered
0 – Edge Triggered

1 = ICW_4 Needed
0 = No ICW_4 Needed

1 – Single
0 - Cascaded

Call Address Interval
1 – Interval of 4 bytes
0 – Interval of 8 bytes.

ICW_2

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	T_7	T_6	T_5	T_4	T_3	A_{10}	A_9	A_8

- T_7 – T3 are A3 – A0 of interrupt address
- A_{10} – A_9, A_8 – Selected according to interrupt request level.
  They are not the address lines of Microprocessor
- A0 =1 selects ICW_2

Fig 4 : Instruction Command Words ICW_1 and ICW_2

- Once ICW_1 is loaded, the following initialization procedure is carried out internally.

a. The edge sense circuit is reset, i.e. by default 8259A interrupts are edge sensitive.

b. IMR is cleared.

c. IR7 input is assigned the lowest priority.

d. Slave mode address is set to 7.

e. Special mask mode is cleared and status read is set to IRR.

f. If $IC_4 = 0$, all the functions of ICW_4 are set to zero. Master/Slave bit in ICW_4 is used in the buffered mode only.

- In an 8085 based system A_{15}-A_8 of the interrupt vector address are the respective bits of ICW_2.
- In 8086 based system A_{15}-A_{11} of the interrupt vector address are inserted in place of T_7 – T_3 respectively and the remaining three bits A_8, A_9, A_{10} are selected depending upon the interrupt level, i.e. from 000 to 111 for IR_0 to IR_7.
- ICW_1 and ICW_2 are compulsory command words in initialization sequence of 8259A as is evident from fig, while ICW_3 and ICW_4 are optional. The ICW_3 is read only when there are more than one 8259A in the system, cascading is used (SNGL=0).

- The SNGL bit in ICW_1 indicates whether the 8259A in the cascade mode or not. The ICW_3 loads an 8-bit slave register. It detailed functions are as follows.
- In master mode [$\overline{SP}$ = 1 or in buffer mode M/S = 1 in ICW_4], the 8-bit slave register will be set bit-wise to 1 for each slave in the system as in fig 5.
- The requesting slave will then release the second byte of a CALL sequence. In slave mode [$\overline{SP}$=0 or if BUF =1 and M/S = 0 in ICW_4] bits D_2 to D_0 identify the slave, i.e. 000 to 111 for slave 1 to slave 8. The slave compares the cascade inputs with these bits and if they are equal, the second byte of the CALL sequence is released by it on the data bus.

Master mode ICW$_3$

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

Sn = 1-IRn Input has a slave
= 0 – IRn Input does not have a slave

Slave mode ICW$_3$

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	ID_2	ID_1	ID_0

$D_2D_1D_0$ – 000 to 111 for IR$_0$ to IR$_7$ or slave 1 to slave 8

ICW$_4$

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	SFNM	BUF	M/S	AEOI	µPM

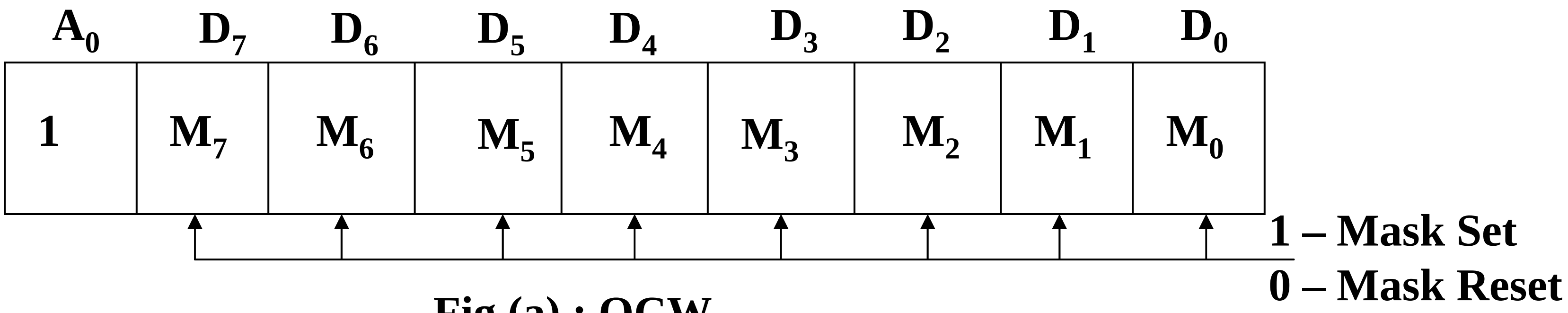
Fig : ICW$_3$ in Master and Slave Mode, ICW$_4$ Bit Functions

- **ICW$_4$**: The use of this command word depends on the IC$_4$ bit of ICW$_1$. If IC$_4$=1, IC$_4$ is used, otherwise it is neglected. The bit functions of ICW4 are described as follow:
- **SFNM**: If BUF = 1, the buffered mode is selected. In the buffered mode, SP/EN acts as enable output and the master/slave is determined using the M/S bit of ICW$_4$.
- **M/S**: If M/S = 1, 8259A is a master. If M/S =0, 8259A is slave. If BUF = 0, M/S is to be neglected.
- **AEOI**: If AEOI = 1, the automatic end of interrupt mode is selected.

- **µPM** : If the µPM bit is 0, the Mcs-85 system operation is selected and if µPM=1, 8086/88 operation is selected.
- **Operation Command Words:** Once 8259A is initialized using the previously discussed command words for initialisation, it is ready for its normal function, i.e. for accepting the interrupts but 8259A has its own way of handling the received interrupts called as modes of operation. These modes of operations can be selected by programming, i.e. writing three internal registers called as operation command words.

- In the three operation command words OCW_1, OCW_2 and OCW_3 every bit corresponds to some operational feature of the mode selected, except for a few bits those are either 1 or 0. The three operation command words are shown in fig with the bit selection details.
- OCW_1 is used to mask the masked and if it is 0 the request is enabled. In OCW_2 the three bits, R, SL and EOI control the end of interrupt, the rotate mode and their combinations as shown in fig below.
- The three bits L_2, L_1 and L_0 in OCW_2 determine the interrupt level to be selected for operation, if SL bit is active i.e. 1.

- The details of OCW_2 are shown in fig.
- In operation command word 3 (OCW_3), if the ESMM bit, i.e. enable special mask mode bit is set to 1, the SMM bit is neglected. If the SMM bit, i.e. special mask mode. When ESMM bit is 0 the SMM bit is neglected. If the SMM bit. i.e. special mask mode bit is 1, the 8259A will enter special mask mode provided ESMM=1.
- If ESMM=1 and SMM=0, the 8259A will return to the normal mask mode. The details of bits of OCW_3 are given in fig along with their bit definitions.

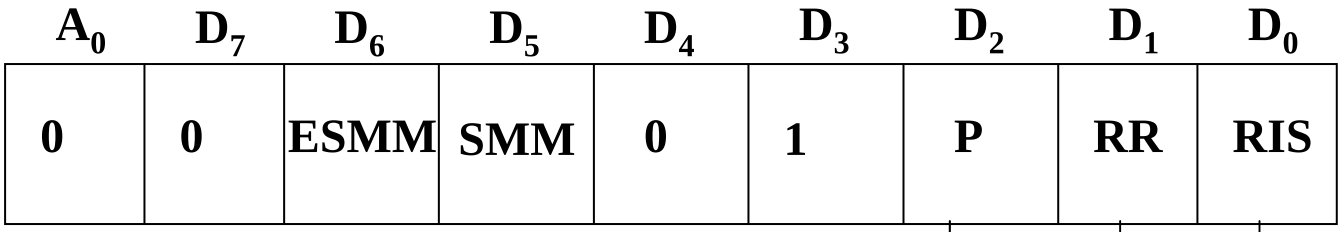


A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	M_7	M_6	M_5	M_4	M_3	M_2	M_1	M_0

1 – Mask Set

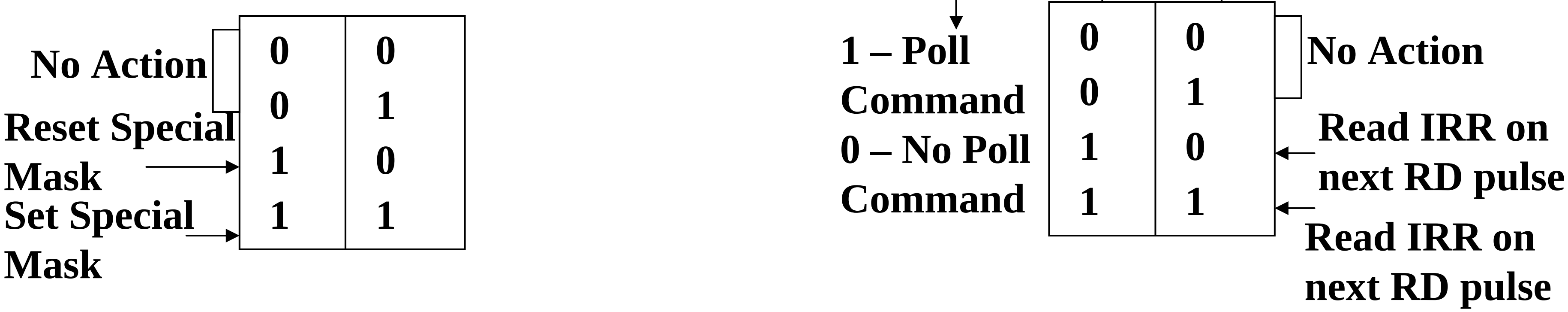
0 – Mask Reset

Fig (a) : OCW_1



A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	ESMM	SMM	0	1	P	RR	RIS

Fig (b) : OCW_3



No Action	0	0
No Action	0	1
Reset Special Mask	1	0
Set Special Mask	1	1

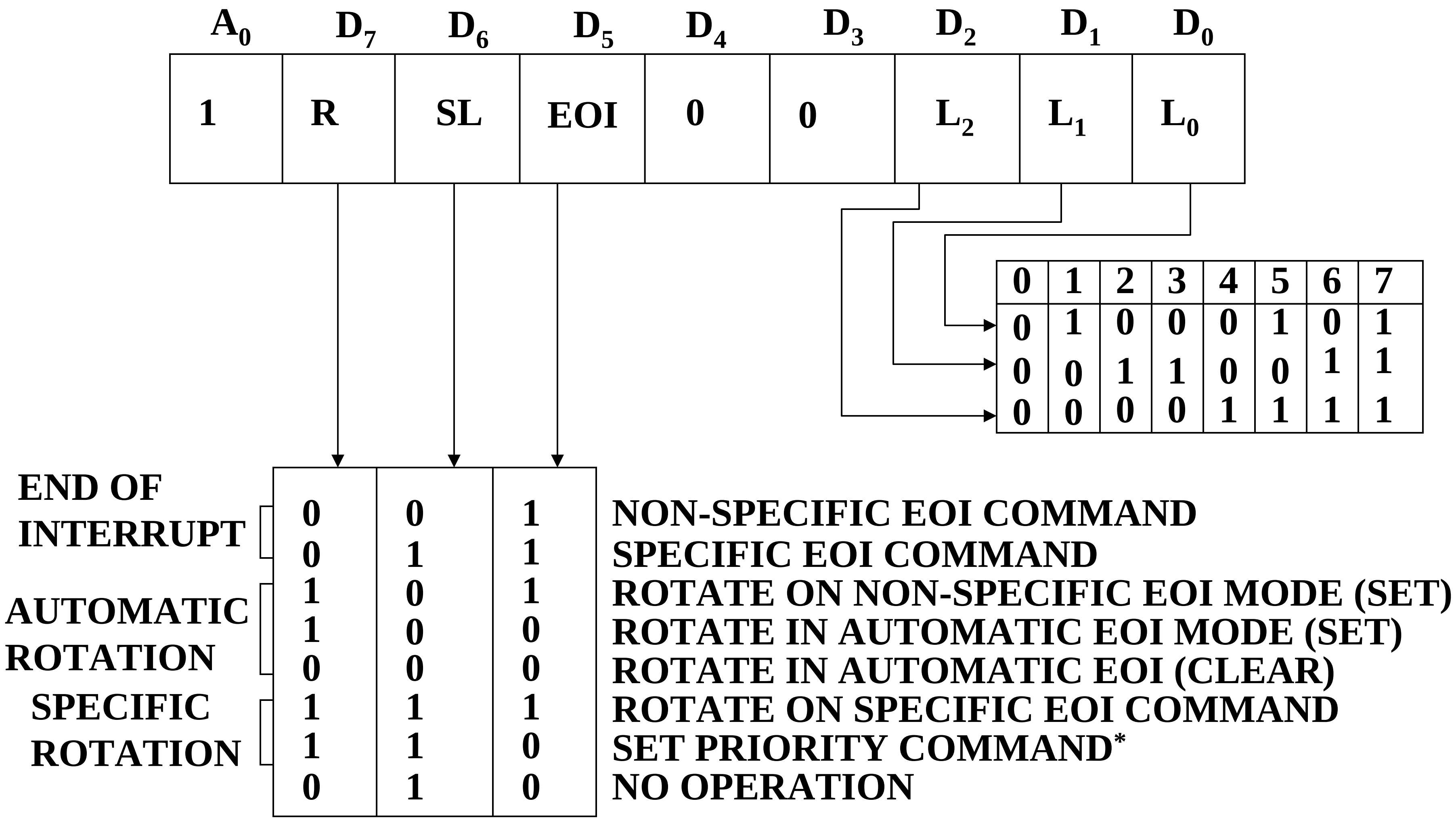
1 – Poll Command

0 – No Poll Command

0	0	No Action
0	1	No Action
1	0	Read IRR on next RD pulse
1	1	Read IRR on next RD pulse

Fig : Operation Command Words

Fig (c) :OCW_2



A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	R	SL	EOI	0	0	L_2	L_1	L_0

0	1	2	3	4	5	6	7
0	1	0	0	0	1	0	1
0	0	1	1	0	0	1	1
0	0	0	0	1	1	1	1

END OF INTERRUPT	0	0	1	NON-SPECIFIC EOI COMMAND
	0	1	1	SPECIFIC EOI COMMAND
AUTOMATIC ROTATION	1	0	1	ROTATE ON NON-SPECIFIC EOI MODE (SET)
	1	0	0	ROTATE IN AUTOMATIC EOI MODE (SET)
	0	0	0	ROTATE IN AUTOMATIC EOI (CLEAR)
SPECIFIC ROTATION	1	1	1	ROTATE ON SPECIFIC EOI COMMAND
	1	1	0	SET PRIORITY COMMAND*
	0	1	0	NO OPERATION

* - In this Mode $L_0 – L_2$ are used

Fig : Operation Command Word

Operating Modes of 8259

- The different modes of operation of 8259A can be programmed by setting or resting the appropriate bits of the ICW or OCW as discussed previously. The different modes of operation of 8259A are explained in the following.
- **Fully Nested Mode** : This is the default mode of operation of 8259A. IR0 has the highest priority and IR_7 has the lowest one. When interrupt request are noticed, the highest priority request amongst them is determined and the vector is placed on the data bus. The corresponding bit of ISR is set and remains set till the microprocessor issues an EOI command just before returning from the service routine or the AEOI bit is set.

- If the ISR (in service) bit is set, all the same or lower priority interrupts are inhibited but higher levels will generate an interrupt, that will be acknowledge only if the microprocessor interrupt enable flag IF is set. The priorities can afterwards be changed by programming the rotating priority modes.
- **End of Interrupt (EOI)** : The ISR bit can be reset either with AEOI bit of ICW1 or by EOI command, issued before returning from the interrupt service routine. There are two types of EOI commands specific and non-specific. When 8259A is operated in the modes that preserve fully nested structure, it can determine which ISR bit is to be reset on EOI.

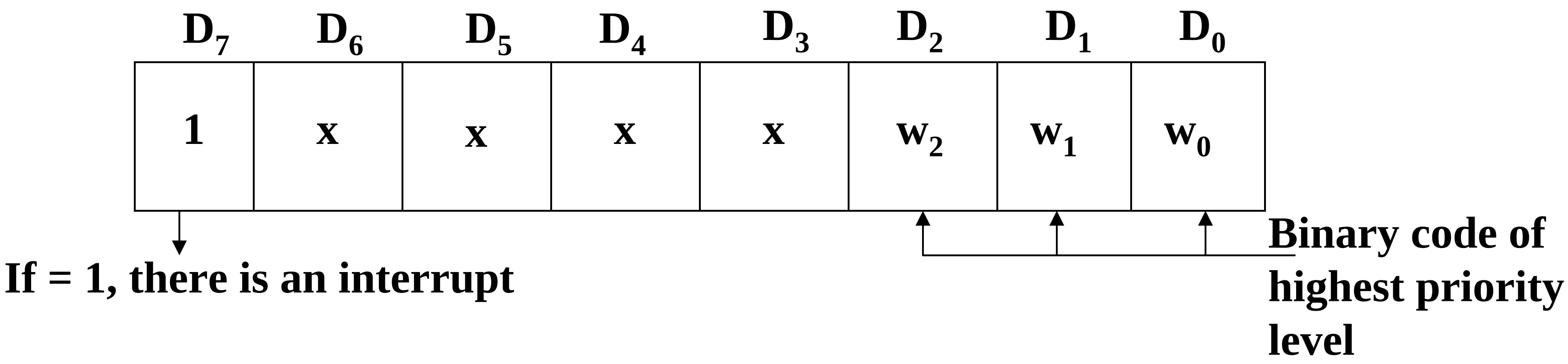
- When non-specific EOI command is issued to 8259A it will be automatically reset the highest ISR bit out of those already set.
- When a mode that may disturb the fully nested structure is used, the 8259A is no longer able to determine the last level acknowledged. In this case a specific EOI command is issued to reset a particular ISR bit. An ISR bit that is masked by the corresponding IMR bit, will not be cleared by non-specific EOI of 8259A, if it is in special mask mode.
- **Automatic Rotation** : This is used in the applications where all the interrupting devices are of equal priority.

- In this mode, an interrupt request IR level receives priority after it is served while the next device to be served gets the highest priority in sequence. Once all the device are served like this, the first device again receives highest priority.
- **Automatic EOI Mode** : Till AEOI=1 in ICW_4, the 8259A operates in AEOI mode. In this mode, the 8259A performs a non-specific EOI operation at the trailing edge of the last INTA pulse automatically. This mode should be used only when a nested multilevel interrupt structure is not required with a single 8259A.

- **Specific Rotation** : In this mode a bottom priority level can be selected, using L2, L1 and L0 in OCW_2 and R=1, SL=1, EOI=0.
- The selected bottom priority fixes other priorities. If IR_5 is selected as a bottom priority, then IR_5 will have least priority and IR4 will have a next higher priority. Thus IR_6 will have the highest priority.
- These priorities can be changed during an EOI command by programming the rotate on specific EOI command in OCW_2.

- **Specific Mask Mode**: In specific mask mode, when a mask bit is set in OCW_1, it inhibits further interrupts at that level and enables interrupt from other levels, which are not masked.
- **Edge and Level Triggered Mode** : This mode decides whether the interrupt should be edge triggered or level triggered. If bit LTIM of ICW_1 =0 they are edge triggered, otherwise the interrupts are level triggered.
- **Reading 8259 Status** : The status of the internal registers of 8259A can be read using this mode. The OCW_3 is used to read IRR and ISR while OCW_1 is used to read IMR. Reading is possible only in no polled mode.

- **Poll Command** : In polled mode of operation, the INT output of 8259A is neglected, though it functions normally, by not connecting INT output or by masking INT input of the microprocessor. The poll mode is entered by setting P=1 in OCW_3.
- The 8259A is polled by using software execution by microprocessor instead of the requests on INT input. The 8259A treats the next RD pulse to the 8259A as an interrupt acknowledge. An appropriate ISR bit is set, if there is a request. The priority level is read and a data word is placed on to data bus, after RD is activated. A poll command may give more than 64 priority levels.



D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	x	x	x	x	w_2	w_1	w_0

If = 1, there is an interrupt

Binary code of highest priority level

Fig : Data Word of 8259

- **Special Fully Nested Mode** : This mode is used in more complicated system, where cascading is used and the priority has to be programmed in the master using ICW$_4$. this is somewhat similar to the normal nested mode.
- In this mode, when an interrupt request from a certain slave is in service, this slave can further send request to the master, if the requesting device connected to the slave has higher priority than the one being currently served. In this mode, the master interrupt the CPU only when the interrupting device has a higher or the same priority than the one current being served. In normal mode, other requests than the one being served are masked out.

- When entering the interrupt service routine the software has to check whether this is the only request from the slave. This is done by sending a non-specific EOI can be sent to the master, otherwise no EOI should be sent. This mode is important, since in the absence of this mode, the slave would interrupt the master only once and hence the priorities of the slave inputs would have been disturbed.
- **Buffered Mode**: When the 83259A is used in the systems where bus driving buffers are used on data buses. The problem of enabling the buffers exists. The 8259A sends buffer enable signal on $\overline{SP}$/ $\overline{EN}$ pin, whenever data is placed on the bus.

- **Cascade Mode** : The 8259A can be connected in a system containing one master and eight slaves (maximum) to handle upto 64 priority levels. The master controls the slaves using CAS_0-CAS_2 which act as chip select inputs (encoded) for slaves.
- In this mode, the slave INT outputs are connected with master IR inputs. When a slave request line is activated and acknowledged, the master will enable the slave to release the vector address during second pulse of $\overline{\text{INTA}}$ sequence.

- The cascade lines are normally low and contain slave address codes from the trailing edge of the first INTA pulse to the trailing edge of the second INTA pulse. Each 8259A in the system must be separately initialized and programmed to work in different modes. The EOI command must be issued twice, one for master and the other for the slave.
- A separate address decoder is used to activate the chip select line of each 8259A.
- Following Fig shows the details of the circuit connections of 8259A in cascade scheme.

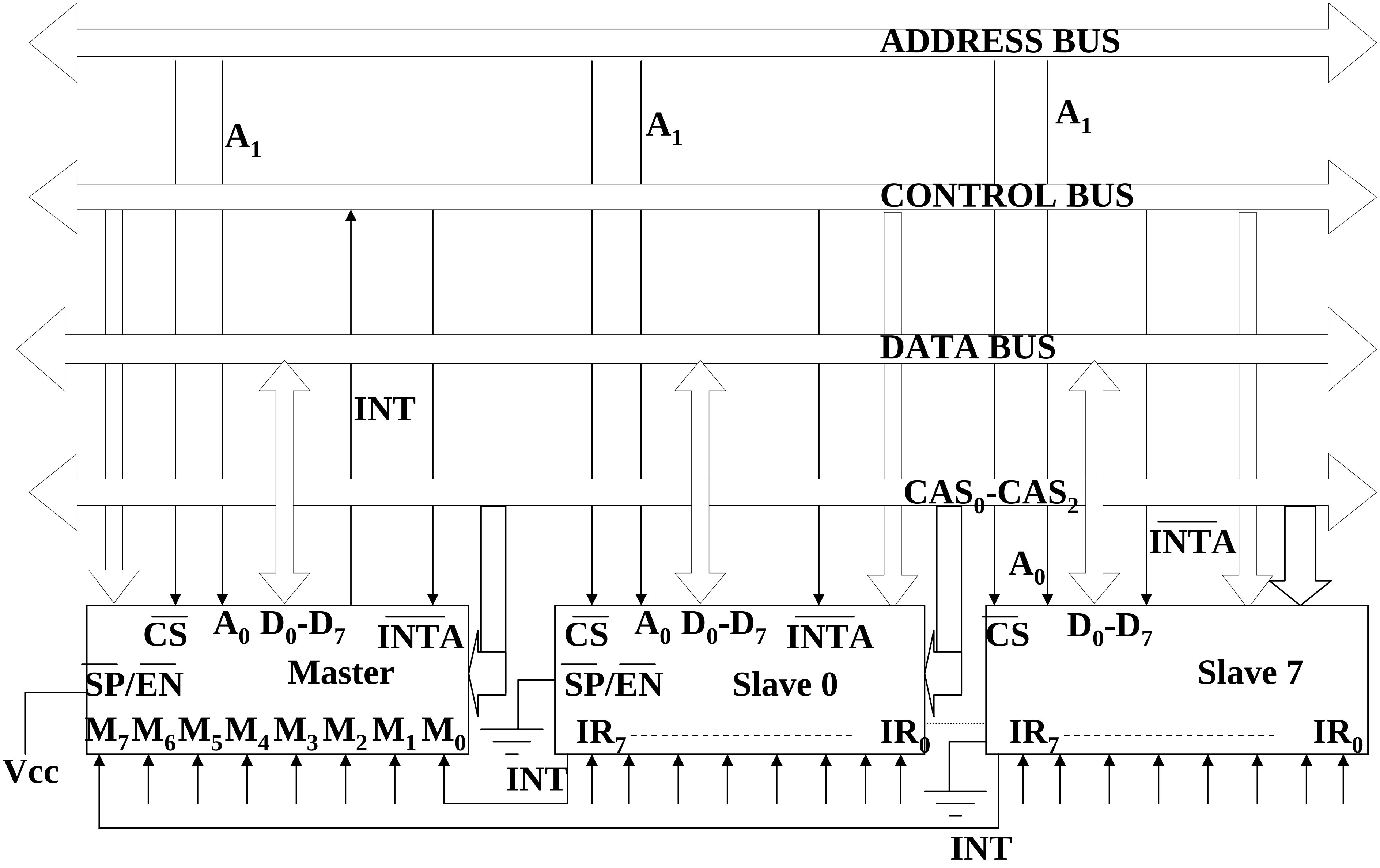


Fig : 8259A in Cascade Mode

8279

- While studying 8255, we have explained the use of 8255 in interfacing keyboards and displays with 8086. The disadvantages of this method of interfacing keyboard and display with 8086 is that the processor has to refresh the display and check the status of the keyboard periodically using polling technique. Thus a considerable amount of CPU time is wasted, reducing the system operating speed.
- Intel’s 8279 is a general purpose keyboard display controller that simultaneously drives the display of a system and interfaces a keyboard with the CPU, leaving it free for its routine task.

Architecture and Signal Descriptions of 8279

- The keyboard display controller chip 8279 provides:

a) a set of four scan lines and eight return lines for interfacing keyboards

b) A set of eight output lines for interfacing display.

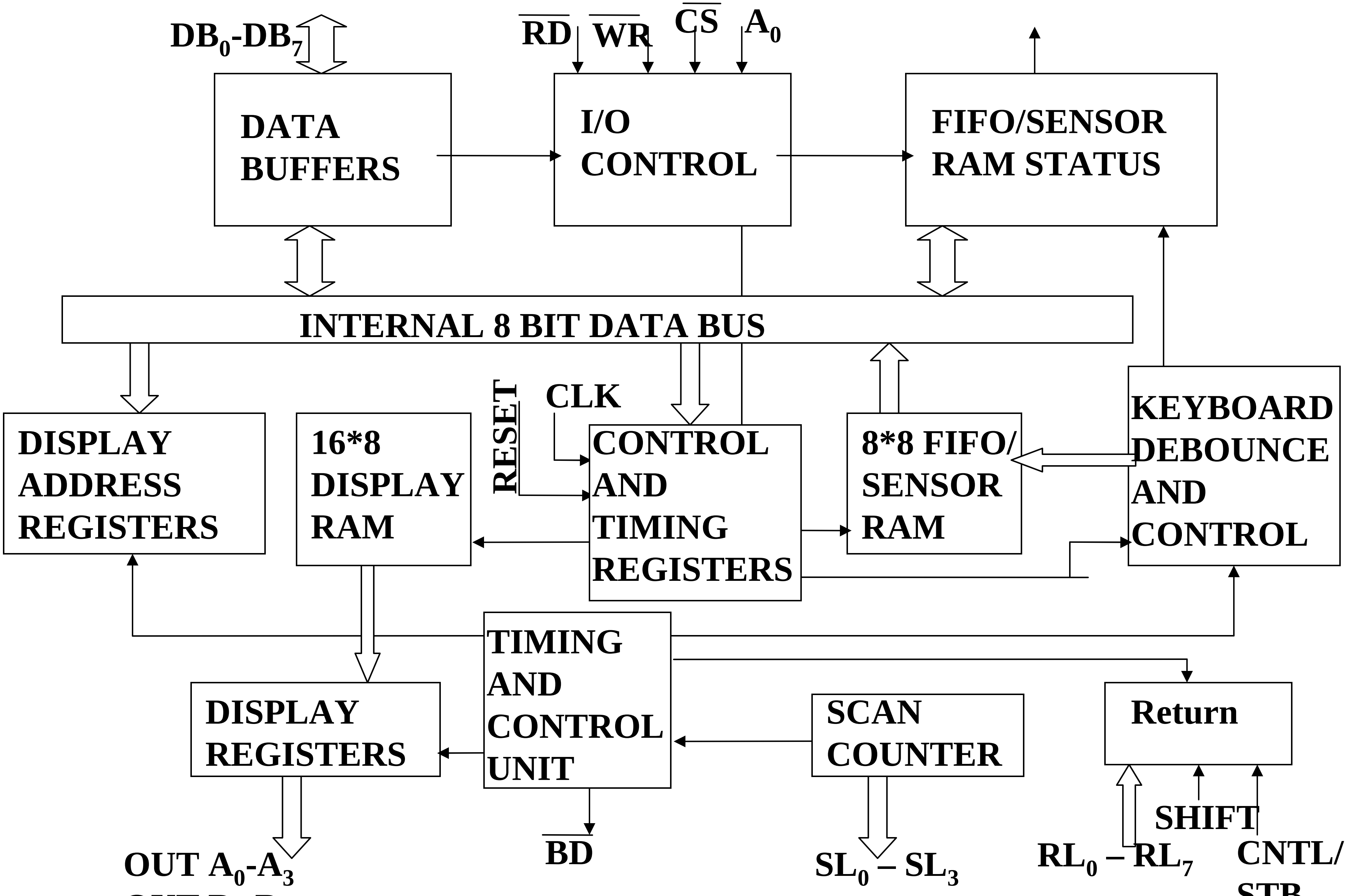
- Fig shows the functional block diagram of 8279 followed by its brief description.
- **I/O Control and Data Buffers** : The I/O control section controls the flow of data to/from the 8279. The data buffers interface the external bus of the system with internal bus of 8279.

- The I/O section is enabled only if $\overline{CS}$ is low. The pins A_0, $\overline{RD}$ and $\overline{WR}$ select the command, status or data read/write operations carried out by the CPU with 8279.
- **Control and Timing Register and Timing Control** : These registers store the keyboard and display modes and other operating conditions programmed by CPU. The registers are written with A_0=1 and $\overline{WR}$=0. The Timing and control unit controls the basic timings for the operation of the circuit. Scan counter divide down the operating frequency of 8279 to derive scan keyboard and scan display frequencies.

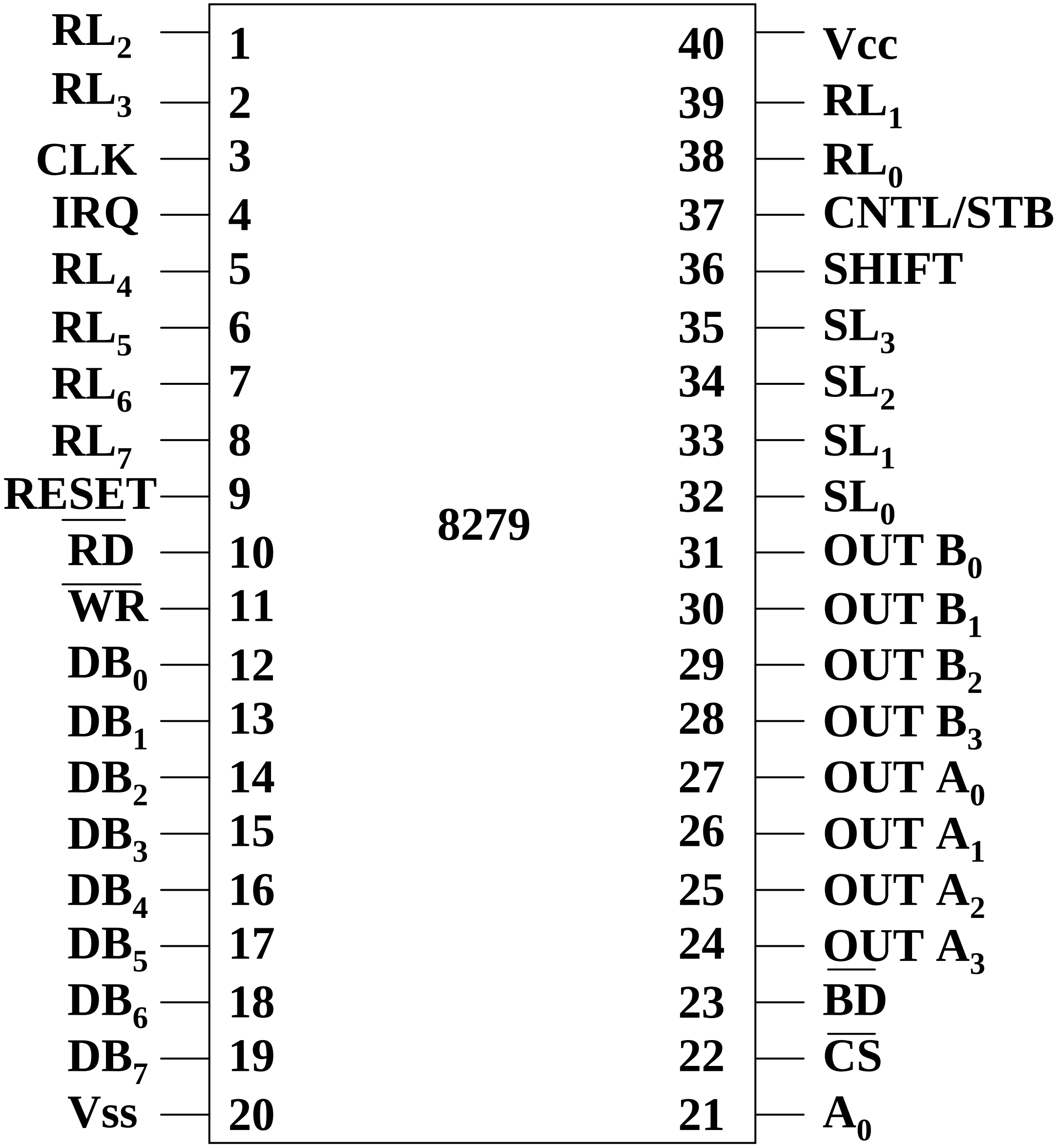
- **Scan Counter** : The scan counter has two modes to scan the key matrix and refresh the display. In the encoded mode, the counter provides binary count that is to be externally decoded to provide the scan lines for keyboard and display (Four externally decoded scan lines may drive upto 16 displays). In the decode scan mode, the counter internally decodes the least significant 2 bits and provides a decoded 1 out of 4 scan on SL_0-SL_3(Four internally decoded scan lines may drive upto 4 displays). The keyboard and display both are in the same mode at a time.

- **Return Buffers and Keyboard Debounce and Control**: This section for a key closure row wise. If a key closer is detected, the keyboard debounce unit debounces the key entry (i.e. wait for 10 ms). After the debounce period, if the key continues to be detected. The code of key is directly transferred to the sensor RAM along with SHIFT and CONTROL key status.
- **FIFO/Sensor RAM and Status Logic**: In keyboard or strobed input mode, this block acts as 8-byte first-in-first-out (FIFO) RAM. Each key code of the pressed key is entered in the order of the entry and in the mean time read by the CPU, till the RAM become empty.

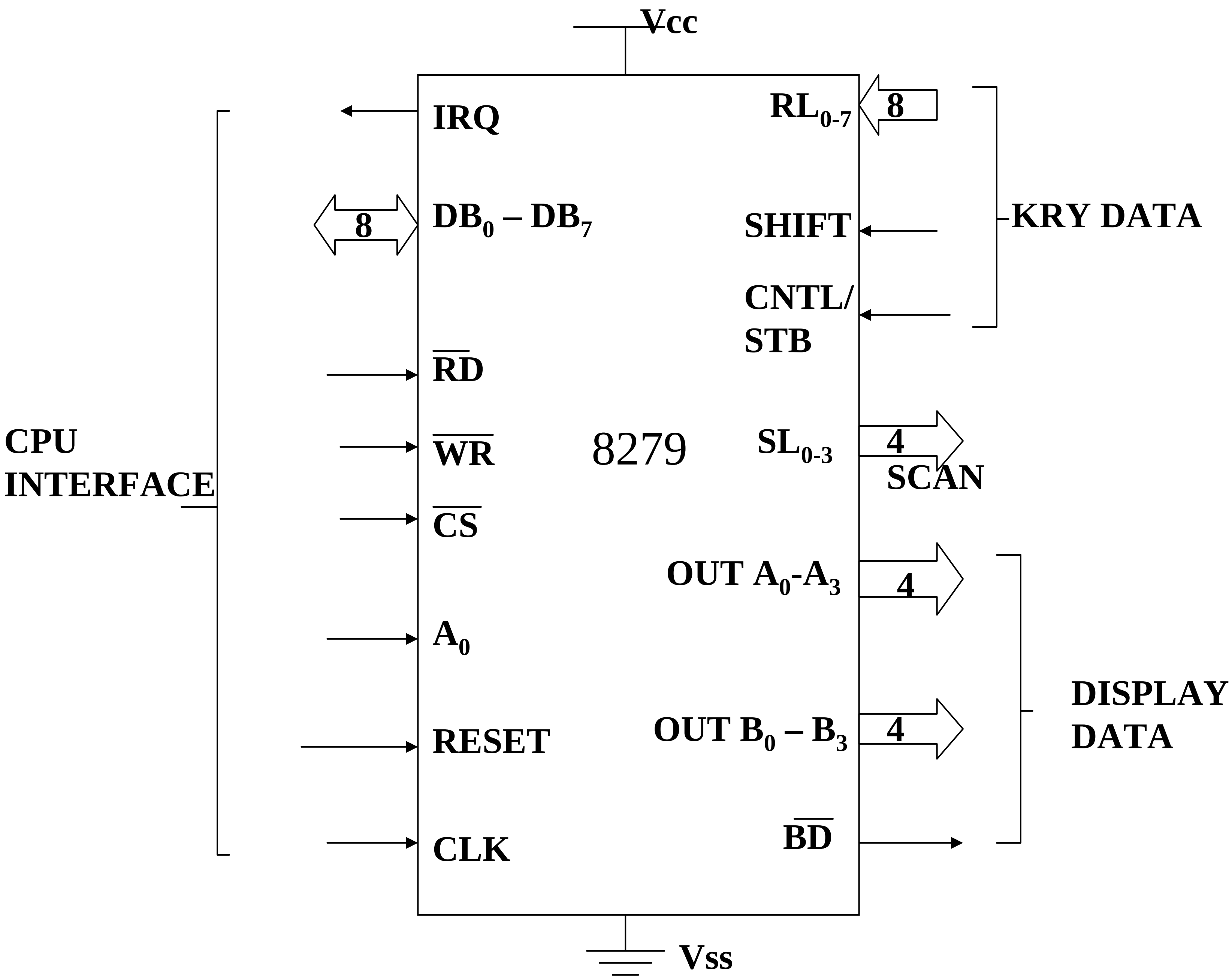
- The status logic generates an interrupt after each FIFO read operation till the FIFO is empty. In scanned sensor matrix mode, this unit acts as sensor RAM. Each row of the sensor RAM is loaded with the status of the corresponding row of sensors in the matrix. If a sensor changes its state, the IRQ line goes high to interrupt the CPU.
- **Display Address Registers and Display RAM** : The display address register holds the address of the word currently being written or read by the CPU to or from the display RAM. The contents of the registers are automatically updated by 8279 to accept the next data entry by CPU.



8279 Internal Architecture



8279 Pin Configuration


Vcc
IRQ
8
DB0 – DB7
RD
WR
CS
A0
RESET
CLK
CPU
INTERFACE
8279
RL0-7
8
SHIFT
CNTL/
STB
KRY DATA
SL0-3
4
SCAN
OUT A0-A3
4
OUT B0 – B3
4
BD
DISPLAY
DATA
Vss

- The signal discription of each of the pins of 8279 as follows :
- **DB_0-DB_7** : These are bidirectional data bus lines. The data and command words to and from the CPU are transferred on these lines.
- **CLK** : This is a clock input used to generate internal timing required by 8279.
- **RESET** : This pin is used to reset 8279. A high on this line reset 8279. After resetting 8279, its in sixteen 8-bit display, left entry encoded scan, 2-key lock out mode. The clock prescaler is set to 31.

- $\overline{\mathbf{CS}}$: Chip Select – A low on this line enables 8279 for normal read or write operations. Other wise, this pin should remain high.
- $\mathbf{A_0}$: A high on this line indicates the transfer of a command or status information. A low on this line indicates the transfer of data. This is used to select one of the internal registers of 8279.
- $\overline{\mathbf{RD}}$**,** $\overline{\mathbf{WR}}$ **(Input/Output) READ/WRITE –** These input pins enable the data buffers to receive or send data over the data bus.

- **IRQ** : This interrupt output lines goes high when there is a data in the FIFO sensor RAM. The interrupt lines goes low with each FIFO RAM read operation but if the FIFO RAM further contains any key-code entry to be read by the CPU, this pin again goes high to generate an interrupt to the CPU.
- **Vss, Vcc** : These are the ground and power supply lines for the circuit.
- **SL_0-SL_3-Scan Lines** : These lines are used to scan the key board matrix and display digits. These lines can be programmed as encoded or decoded, using the mode control register.

- **RL_0 - RL_7 - Return Lines** : These are the input lines which are connected to one terminal of keys, while the other terminal of the keys are connected to the decoded scan lines. These are normally high, but pulled low when a key is pressed.
- **SHIFT** : The status of the shift input lines is stored along with each key code in FIFO, in scanned keyboard mode. It is pulled up internally to keep it high, till it is pulled low with a key closure.
- **BD – Blank Display** : This output pin is used to blank the display during digit switching or by a blanking closure.

- **OUT A_0 – OUT A_3 and OUT B_0 – OUT B_3** – These are the output ports for two 16*4 or 16*8 internal display refresh registers. The data from these lines is synchronized with the scan lines to scan the display and keyboard. The two 4-bit ports may also as one 8-bit port.
- **CNTL/STB- CONTROL/STROBED I/P Mode** : In keyboard mode, this lines is used as a control input and stored in FIFO on a key closure. The line is a strobed lines that enters the data into FIFO RAM, in strobed input mode. It has an interrupt pull up. The lines is pulled down with a key closer.

Modes of Operation of 8279

- The modes of operation of 8279 are as follows :

1. Input (Keyboard) modes.
2. Output (Display) modes.

- **Input (Keyboard) Modes :** 8279 provides three input modes. These modes are as follows:

1. **Scanned Keyboard Mode** : This mode allows a key matrix to be interfaced using either encoded or decoded scans. In encoded scan, an 8*8 keyboard or in decoded scan, a 4*8 keyboard can be interfaced. The code of key pressed with SHIFT and CONTROL status is stored into the FIFO RAM.

2. **Scanned Sensor Matrix** : In this mode, a sensor array can be interfaced with 8279 using either encoded or decoded scans. With encoded scan 8*8 sensor matrix or with decoded scan 4*8 sensor matrix can be interfaced. The sensor codes are stored in the CPU addressable sensor RAM.
3. **Strobed input**: In this mode, if the control lines goes low, the data on return lines, is stored in the FIFO byte by byte.

- **Output (Display) Modes** : 8279 provides two output modes for selecting the display options. These are discussed briefly.

1. **Display Scan** : In this mode 8279 provides 8 or 16 character multiplexed displays those can be organized as dual 4- bit or single 8-bit display units.
2. **Display Entry** : (right entry or left entry mode) 8279 allows options for data entry on the displays. The display data is entered for display either from the right side or from the left side.

Keyboard Modes

i. **Scanned Keyboard mode with 2 Key Lockout** : In this mode of operation, when a key is pressed, a debounce logic comes into operation. During the next two scans, other keys are checked for closure and if no other key is pressed the first pressed key is identified.

- The key code of the identified key is entered into the FIFO with SHIFT and CNTL status, provided the FIFO is not full, i.e. it has at least one byte free. If the FIFO does not have any free byte, naturally the key data will not be entered and the error flag is set.

- If FIFO has at least one byte free, the above code is entered into it and the 8279 generates an interrupt on IRQ line to the CPU to inform about the previous key closures. If another key is found closed during the first key, the keycode is entered in FIFO.
- If the first pressed key is released before the others, the first will be ignored. A key code is entered to FIFO only once for each valid depression, independent of other keys pressed along with it, or released before it.

- If two keys are pressed within a debounce cycle (simultaneously), no key is recognized till one of them remains closed and the other is released. The last key, that remains depressed is considered as single valid key depression.

ii. **Scanned Keyboard with N-Key Rollover** : In this mode, each key depression is treated independently. When a key is pressed, the debounce circuit waits for 2 keyboards scans and then checks whether the key is still depressed. If it is still depressed, the code is entered in FIFO RAM.

- Any number of keys can be pressed simultaneously and recognized in the order, the keyboard scan recorded them. All the codes of such keys are entered into FIFO.
- In this mode, the first pressed key need not be released before the second is pressed. All the keys are sensed in the order of their depression, rather in the order the keyboard scan senses them, and independent of the order of their release.

iii. **Scanned Keyboard Special Error Mode :** This mode is valid only under the N-Key rollover mode. This mode is programmed using end interrupt / error mode set command. If during a single debounce period (two keyboard scans) two keys are found pressed , this is considered a simultaneous depression and an error flag is set**.**

- This flag, if set, prevents further writing in FIFO but allows the generation of further interrupts to the CPU for FIFO read. The error flag can be read by reading the FIFO status word. The error Flag is set by sending normal clear command with CF = 1.

iv. **Sensor Matrix Mode** : In the sensor matrix mode, the debounce logic is inhibited. The 8-byte FIFO RAM now acts as 8 * 8 bit memory matrix. The status of the sensor switch matrix is fed directly to sensor RAM matrix. Thus the sensor RAM bits contains the row-wise and column wise status of the sensors in the sensor matrix.

- The IRQ line goes high, if any change in sensor value is detected at the end of a sensor matrix scan or the sensor RAM has a previous entry to be read by the CPU. The IRQ line is reset by the first data read operation, if AI = 0, otherwise, by issuing the end interrupt command. AI is a bit in read sensor RAM word.

Display Modes

- There are various options of data display. For example, the command number of characters can be 8 or 16, with each character organised as single 8-bit or dual 4-bit codes. Similarly there are two display formats.
- The first one is known as left entry mode or type writer mode, since in a type writer the first character typed appears at the left-most position, while the subsequent characters appear successively to the right of the first one. The other display format is known as right entry mode, or calculator mode, since in a calculator the first character entered appears at the rightmost position and this character is shifted one position left when the next characters is entered.

- Thus all the previously entered characters are shifted left by one position when a new characters is entered.

i. **Left Entry Mode** : In the left entry mode, the data is entered from left side of the display unit. Address 0 of the display RAM contains the leftmost display characters and address 15 of the RAM contains the right most display characters. It is just like writing in our address is automatically updated with successive reads or writes. The first entry is displayed on the leftmost display and the sixteenth entry on the rightmost display. The seventeenth entry is again displayed at the leftmost display position.

ii. **Right Entry Mode** : In this right entry mode, the first entry to be displayed is entered on the rightmost display. The next entry is also placed in the right most display but after the previous display is shifted left by one display position. The leftmost characters is shifted out of that display at the seventeenth entry and is lost, i.e. it is pushed out of the display RAM.

Command Words of 8279

- All the command words or status words are written or read with A0 = 1 and CS = 0 to or from 8279. This section describes the various command available in 8279.

a) **Keyboard Display Mode Set** – The format of the command word to select different modes of operation of 8279 is given below with its bit definitions.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
0	0	D	D	D	K	K	K	1

D	D	Display modes
0	0	Eight 8-bit character Left entry
0	1	Sixteen 8-bit character left entry
1	0	Eight 8-bit character Right entry
1	1	Sixteen 8-bit character Right entry

K	K	K	Keyboard modes
0	0	0	Encoded Scan, 2 key lockout (Default after reset)
0	0	1	Decoded Scan, 2 key lockout
0	1	0	Encoded Scan, N- key Roll over
0	1	1	Decoded Scan, N- key Roll over
1	0	0	Encode Scan, N- key Roll over
1	0	1	Decoded Scan, N- key Roll over
1	1	0	Strobed Input Encoded Scan
1	1	1	Strobed Input Decoded Scan

b) **Programmable clock** : The clock for operation of 8279 is obtained by dividing the external clock input signal by a programmable constant called prescaler.

- PPPPP is a 5-bit binary constant. The input frequency is divided by a decimal constant ranging from 2 to 31, decided by the bits of an internal prescaler, PPPPP.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
0	0	1	P	P	P	P	P	1

c) **Read FIFO / Sensor RAM** : The format of this command is given below.

- This word is written to set up 8279 for reading FIFO/ sensor RAM. In scanned keyboard mode, AI and AAA bits are of no use. The 8279 will automatically drive data bus for each subsequent read, in the same sequence, in which the data was entered.
- In sensor matrix mode, the bits AAA select one of the 8 rows of RAM. If AI flag is set, each successive read will be from the subsequent RAM location.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
0	1	0	AI	X	A	A	A	1

X – don’t care

AI – Auto Increment Flag

AAA – Address pointer to 8 bit FIFO RAM

d) **Read Display RAM** : This command enables a programmer to read the display RAM data. The CPU writes this command word to 8279 to prepare it for display RAM read operation. AI is auto increment flag and AAAA, the 4-bit address points to the 16-byte display RAM that is to be read. If AI=1, the address will be automatically, incremented after each read or write to the Display RAM. The same address counter is used for reading and writing.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
0	1	1	AI	A	A	A	A	1

e) **Write Display RAM** :

AI – Auto increment Flag.

AAAA – 4 bit address for 16-bit display RAM to be written.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
1	0	0	AI	A	A	A	A	1

f) **Display Write Inhibit/Blanking** : The IW (inhibit write flag) bits are used to mask the individual nibble as shown in the below command word. The output lines are divided into two nibbles ($OUTA_0$ – $OUTA_3$) and ($OUTB_0$ – $OUTB_3$), those can be masked by setting the corresponding IW bit to 1.

- Once a nibble is masked by setting the corresponding IW bit to 1, the entry to display RAM does not affect the nibble even though it may change the unmasked nibble. The blank display bit flags (BL) are used for blanking A and B nibbles.

- Here D_0, D_2 corresponds to $OUTB_0$ – $OUTB_3$ while D1 and D3 corresponds to $OUTA_0$-$OUTA_3$ for blanking and masking.
- If the user wants to clear the display, blank (BL) bits are available for each nibble as shown in format. Both BL bits will have to be cleared for blanking both the nibbles.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
1	0	1	X	IW	IW	BL	BL	1

g) **Clear Display RAM** : The CD_2, CD_1, CD_0 is a selectable blanking code to clear all the rows of the display RAM as given below. The characters A and B represents the output nibbles.

- CD_2 must be 1 for enabling the clear display command. If $CD_2 = 0$, the clear display command is invoked by setting CA=1 and maintaining CD_1, CD_0 bits exactly same as above. If CF=1, FIFO status is cleared and IRQ line is pulled down.
- Also the sensor RAM pointer is set to row 0. if CA=1, this combines the effect of CD and CF bits. Here, CA represents Clear All and CF as Clear FIFO RAM.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
1	1	0	CD_2	CD_1	CD_0	CF	CA	1

CD_2	CD_1	CD_0
1	0	X
1	1	0
1	1	1

All zeros (x don't care) AB=00

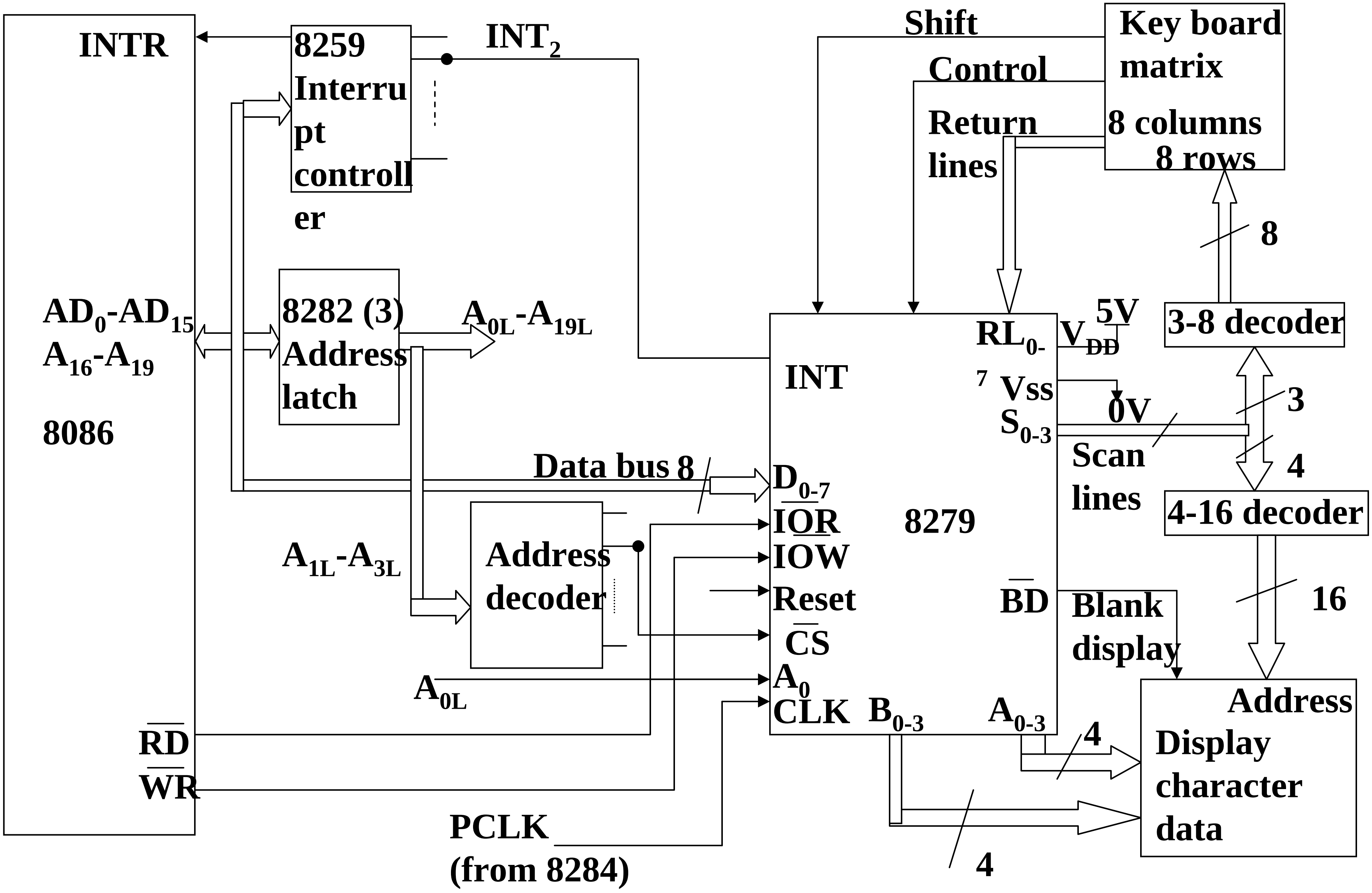
A3-A0 =2 (0010) and B3-B0=00 (0000)

All ones (AB =FF), i.e. clear RAM

h) **End Interrupt / Error mode Set** : For the sensor matrix mode, this command lowers the IRQ line and enables further writing into the RAM. Otherwise, if a change in sensor value is detected, IRQ goes high that inhibits writing in the sensor RAM.

- For N-Key roll over mode, if the E bit is programmed to be ‘1’, the 8279 operates in special Error mode. Details of this mode are described in scanned keyboard special error mode. **X**- don’t care.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
1	1	1	E	X	X	X	X	1


INTR
8086
AD0-AD15
A16-A19
RD
WR
8259 Interrupt controller
INT2
8282 (3) Address latch
A0L-A19L
Data bus 8
A1L-A3L
Address decoder
A0L
PCLK (from 8284)
INT
D0-7
IOR
IOW
Reset
CS
A0
CLK
B0-3
A0-3
8279
RL0-7
VDD
5V
Vss
0V
S0-3
BD
Shift
Control
Return lines
Key board matrix
8 columns
8 rows
8
3-8 decoder
3
4
Scan lines
4-16 decoder
16
Blank display
Address
Display character data
4
4

Interfacing To Alphanumeric Displays

- To give directions or data values to users, many microprocessor-controlled instruments and machines need to display letters of the alphabet and numbers. In systems where a large amount of data needs to be displayed a CRT is used to display the data. In system where only a small amount of data needs to be displayed, simple digit-type displays are often used.
- There are several technologies used to make these digit-oriented displays but we are discussing only the two major types.

- These are *light emitting diodes* (LED) and *liquid-crystal displays* (LCD).
- LCD displays use very low power, so they are often used in portable, battery-powered instruments. They do not emit their own light, they simply change the reflection of available light. Therefore, for an instrument that is to be used in low-light conditions, you have to include a light source for LCDs or use LEDs which emit their own light.

- Alphanumeric LED displays are available in three common formats. For displaying only number and hexadecimal letters, simple 7-segment displays such as that as shown in fig are used.
- To display numbers and the entire alphabet, 18 segment displays such as shown in fig or 5 by 7 dot-matrix displays such as that shown in fig can be used. The 7-segment type is the least expensive, most commonly used and easiest to interface with, so we will concentrate first on how to interface with this type.

1. ***Directly Driving LED Displays:*** Figure shows a circuit that you might connect to a parallel port on a microcomputer to drive a single 7-segment , common-anode display. For a common-anode display, a segment is tuned on by applying a logic low to it.

- The 7447 converts a BCD code applied to its inputs to the pattern of lows required to display the number represented by the BCD code. This circuit connection is referred to as a *static display* because current is being passed through the display at all times.

- Each segment requires a current of between 5 and 30mA to light. Let’s assume you want a current of 20mA. The voltage drop across the LED when it is lit is about 1.5V.
- The output low voltage for the 7447 is a maximum of 0.4V at 40mA. So assume that it is about 0.2V at 20mA. Subtracting these two voltage drop from the supply voltage of 5V leaves 3.3V across the current limiting resistor. Dividing 3.3V by 20mA gives a value of 168Ω for the current-limiting resistor. The voltage drops across the LED and the output of 7447 are not exactly predictable and exact current through the LED is not critical as long as we don’t exceed its maximum rating.

2. ***Software-Multiplexed LED Display:***

- The circuit in fig works for driving just one or two LED digits with a parallel output port. However, this scheme has several problem if you want to drive, eight digits.
- The first problem is power consumption. For worst-case calculations, assume that all 8 digits are displaying the digit 8, so all 7 segments are all lit. Seven segment time 20mA per segment gives a current of 140mA per digit. Multiplying this by 8 digits gives a total current of 1120mA or 1.12A for 8 digits.

- A second problem of the static approach is that each display digit requires a separate 7447 decoder, each of which uses of another 13mA. The current required by the decoders and the LED displays might be several times the current required by the reset of the circuitry in the instrument.
- To solve the problem of the static display approach, we use a *multiplex method*, example for an explanation of the multiplexing.

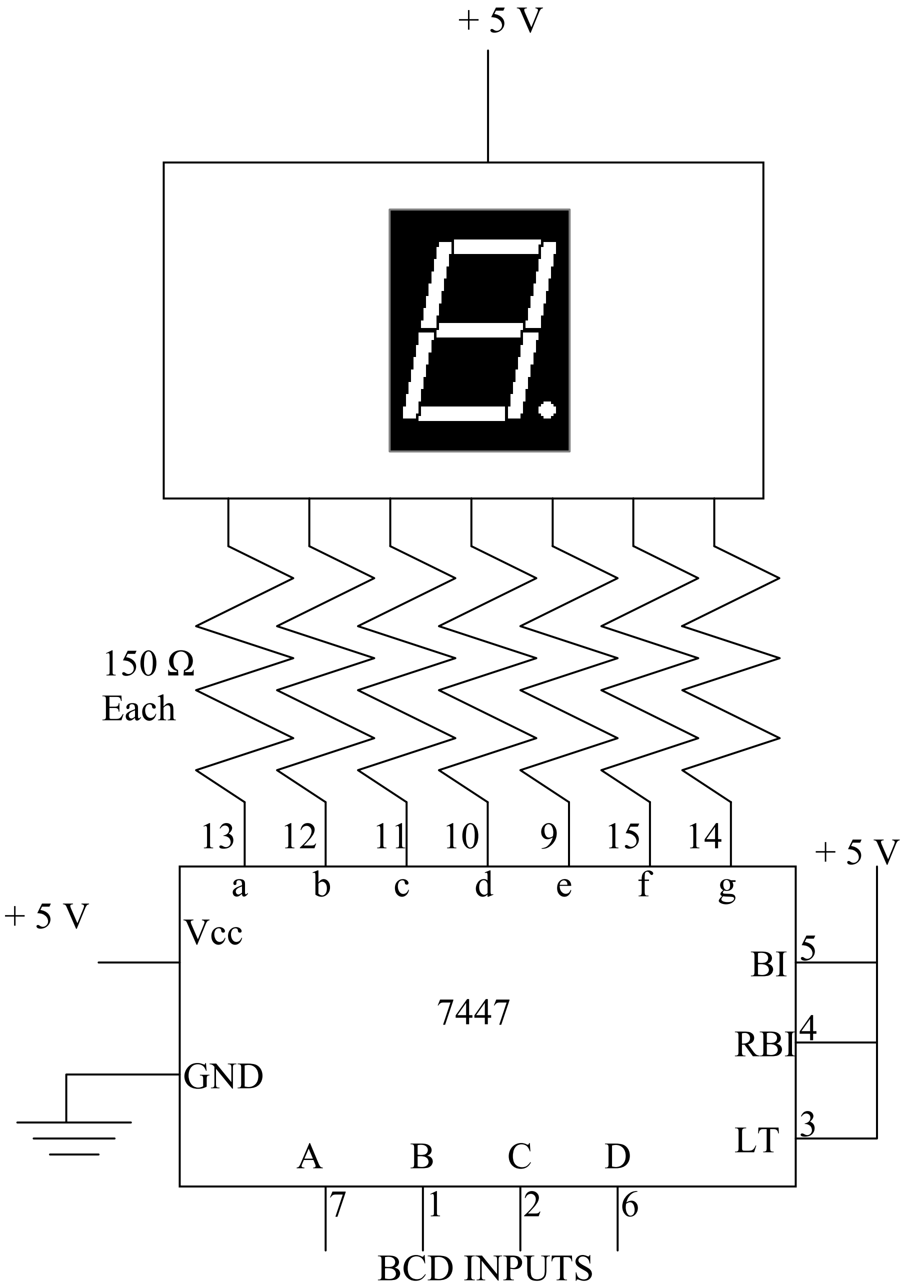
- The fig shows a circuit you can add to a couple of microcomputer ports to drive some common anode LED displays in a multiplexed manner. The circuit has only one 7447 and that the segment outputs of the 7447 are bused in parallel to the segment inputs of all the digits.
- The question that may occur to you on first seeing this is: Aren’t all the digits going to display the same number? The answer is that they would if all the digits were turned on at the same time. The tricky of multiplexing displays is that only one display digit is turned on at a time.

- The PNP transistor is series with the common anode of each digit acts as on/off switch for that digit. Here’s how the multiplexing process works.
- The BCD code for digit 1 is first output from port B to the 7447. the 7447 outputs the corresponding 7-segment code on the segment bus lines. The transistor connected to digit 1 is then turned on by outputting a low to the appropriate bit of port A. All the rest of the bits of port A are made high to make sure no other digits are turned on. After 1 or 2 ms, digit 1 is turned off by outputting all highs to port A.

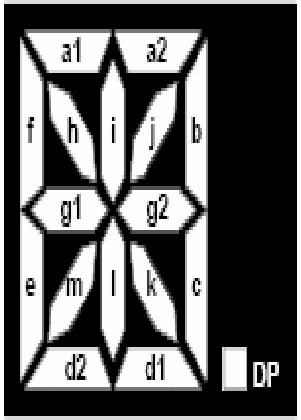
- The BCD code for digit 2 is then output to the 7447 on port B, and a word to turn on digit 2 is output on port A.
- After 1 or 2 ms, digit 2 is turned off and the process is repeated for digit 3. the process is continued until all the digits have had a turn. Then digit 1 and the following digits are lit again in turn.
- A procedure which is called on an interrupt basis every 2ms to keep these displays refreshed wit some values stored in a table. With 8 digits and 2ms per digit, you get back to digit 1 every 16ms or about 60 times a second.

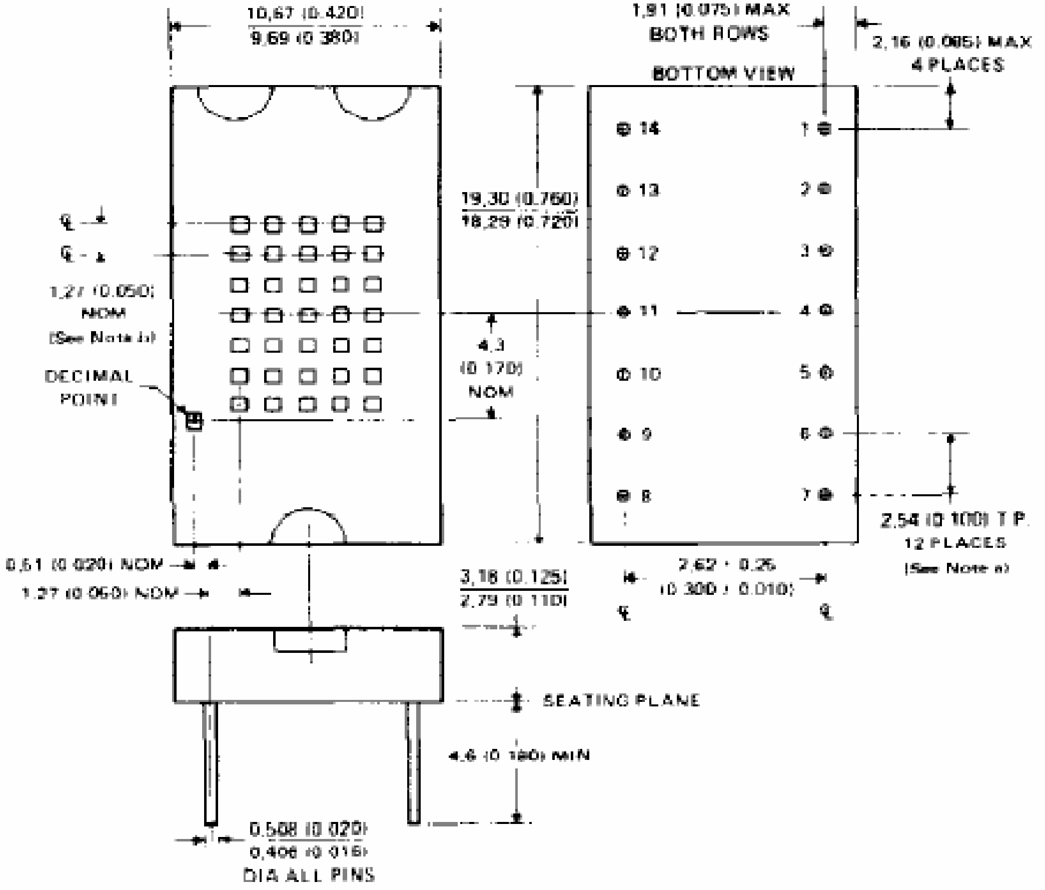
- This refresh rate is fast enough so that the digits will each appear to be lit all time. Refresh rates of 40 to 200 times a second are acceptable.
- The immediately obvious advantages of multiplexing the displays are that only one 7447 is required, and only one digit is lit at a time. We usually increase the current per segment to between 40 and 60 mA for multiplexed displays so that they will appear as bright as they would if they were not multiplexed. Even with this increased segment current, multiplexing gives a large saving in power and parts.

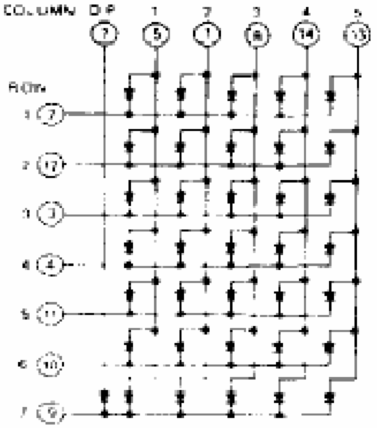
- The software-multiplexed approach we have just described can also be used to drive 18-segment LED devices and dot-matrix LED device. For these devices, however you replace the 7447 in fig with ROM which generates the required segment codes when the ASCII code for a character is applied to the address inputs of the ROM.



Circuit for driving single 7-segment LED display with 7447


a1
a2
f
h
i
j
b
g1
g2
e
m
l
k
c
d2
d1
DP





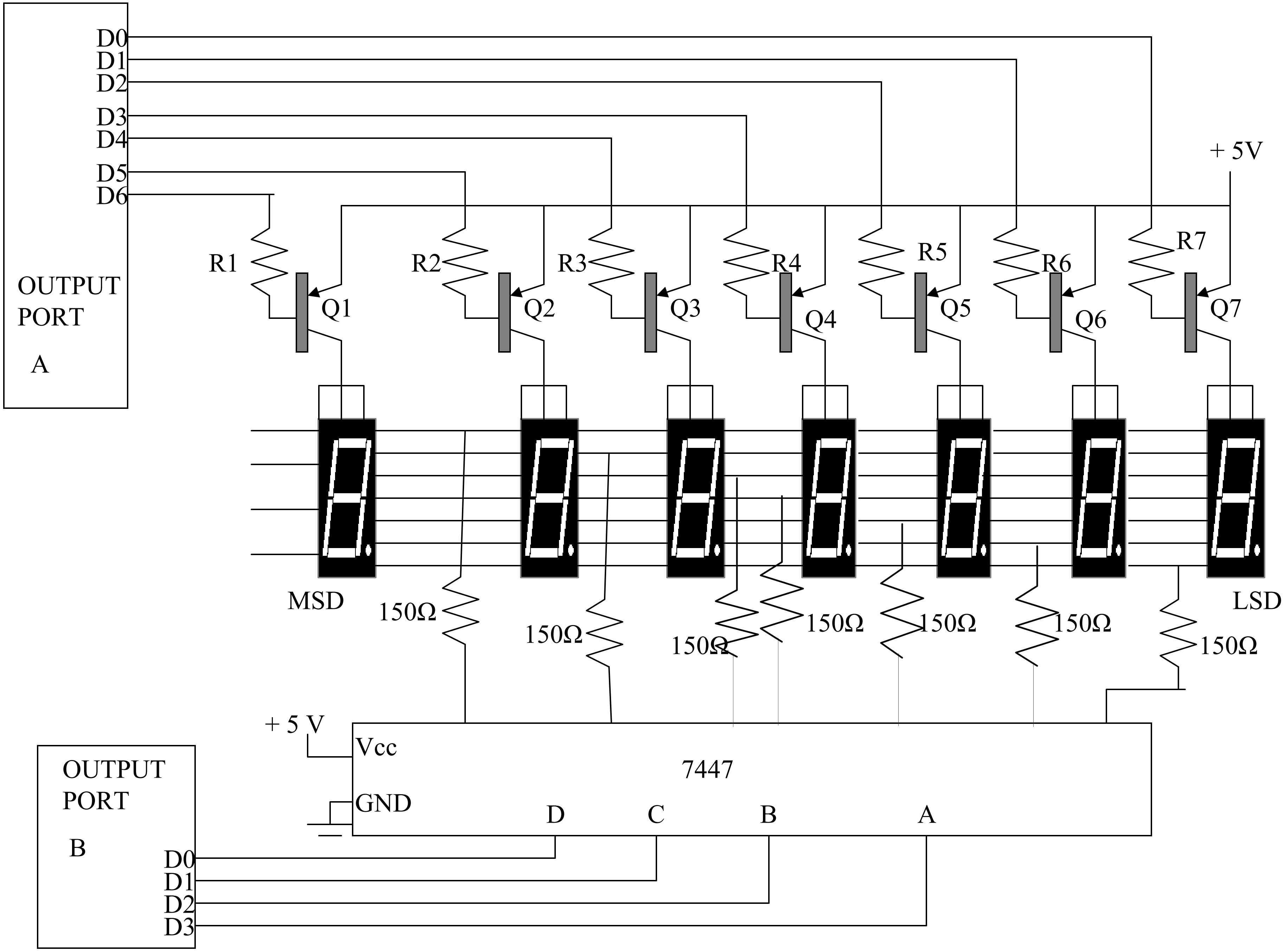
TOP VIEW ORIENTATION

NOTES: a. The true position spacing is 2,54 mm (0.100 inch) between lead centerlines. Each pin centerline is located within 0,25 mm (0.010 inch) of its true longitudinal position.

b. Vertical and horizontal spacing between centerlines of rows and columns nominally 1,27 mm (0.050 inch).

ALL DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

CL 37


OUTPUT
PORT
A
D0
D1
D2
D3
D4
D5
D6
+ 5V
R1
R2
R3
R4
R5
R6
R7
Q1
Q2
Q3
Q4
Q5
Q6
Q7
MSD
LSD
150Ω
150Ω
150Ω
150Ω
150Ω
150Ω
150Ω
+ 5 V
Vcc
GND
7447
D
C
B
A
OUTPUT
PORT
B
D0
D1
D2
D3

Liquid Crystal Display

- Liquid Crystal displays are created by sandwiching a thin 10-12 μm layer of a liquid-crystal fluid between two glass plates. A transparent, electrically conductive film or backplane is put on the rear glass sheet. Transparent sections of conductive film in the shape of the desired characters are coated on the front glass plate.
- When a voltage is applied between a segment and the backplane, an electric field is created in the region under the segment. This electric field changes the transmission of light through the region under the segment film.

- There are two commonly available types of LCD : dynamic scattering and field-effect.
- The Dynamic scattering types of LCD: It scrambles the molecules where the field is present. This produces an etched-glass-looking light character on a dark background.
- Field-effect types use polarization to absorb light where the electric field is present. This produces dark characters on a silver- gray background.
- Most LCD’s require a voltage of 2 or 3 V between the backplane and a segment to turn on the segment.

- We cannot just connect the backplane to ground and drive the segment with the outputs of a TTL decoder. The reason for this is a steady dc voltage of more than about 50mV is applied between a segment and the backplane.
- To prevent a dc buildup on the segments, the segment-drive signals for LCD must be square waves with a frequency of 30 to 150 Hz.
- Even if you pulse the TTL decoder, it still will not work because the output low voltage of TTL devices is greater than 50mV.
- CMOS gates are often used to drive LCDs.

- The Following fig shows how two CMOS gate outputs can be connected to drive an LCD segment and backplane.
- The off segment receives the same drive signal as the backplane. There is never any voltage between them, so no electric field is produced. The waveform for the on segment is 180 out of phase with the backplane signal, so the voltage between this segment and the backplane will always be +V.
- The logic for this signal, a square wave and its complement. To the driving gates, the segment-backplane sandwich appears as a somewhat leaky capacitor.

- The CMOS gates can be easily supply the current required to charge and discharge this small capacitance.
- Older inexpensive LCD displays turn on and off too slowly to be multiplexed the way we do LED display.
- At 0c some LCD may require as mush as 0.5s to turn on or off. To interface to those types we use a nonmultiplexed driver device.
- More expensive LCD can turn on and off faster, so they are often multiplexed using a variety of techniques.
- In the following section we show you how to interface a nonmultiplexed LCD to a microprocessor such as SDK-86.

- Intersil ICM7211M can be connected to drive a 4-digit, nonmultiplexed, 7-segment LCD display.
- The 7211M input can be connected to port pins or directly to microcomputer bus. We have connected the CS inputs to the Y2 output of the 74LS138 port decoder.
- According to the truth table the device will then be addressable as ports with a base address of FF10H. SDK-86 system address lines A2 is connected to the digit-select input (DS2) and system address lines A1 is connected to the DS1 input. This gives digit 4 a system address of FF10H.

A8-A15	A5-A7	A4	A3	A2	A1	A0	M/I$\overline{\text{O}}$	Y Output Selected	System Base Address	Device
1	0	0	0	X	X	0	0	00	F F 0 0	8259A #1
1	0	0	1	X	X	0	0	1	F F 0 8	8259A #2
1	0	1	0	X	X	0	0	2	F F 1 0	
1	0	1	1	X	X	0	0	3	F F 1 8	
1	0	0	0	X	X	1	0	4	F F 0 1	8254
1	0	0	1	X	X	1	0	5	F F 0 9	
1	0	1	0	X	X	1	0	6	F F 1 1	
1	0	1	1	X	X	1	0	7	F F 1 9	
ALL OTHER STATES								NONE		

Fig : Truth table for 74LS138 address decoder

- Digit 3 will be addressed at FF12H, digit 2 at FF14H and digit 1 at FF16H.
- The data inputs are connected to the lower four lines of the SDK-86 data bus. The oscillator input is left open. To display a character on one of the digits, you simply keep the 4-bit hex code for that digit in the lower 4 bits of the AL register and output it to the system address for that digit.
- The ICM7211M converts the 4-bit hex code to the required 7-segment code.

- The rising edge of the CS input signal causes the 7-segment code to be latched in the output latches for the address digit.
- An internal oscillator automatically generates the segment and backplane drive waveforms as in fig . For interfacing with the LCD displays which can be multiplexed the Intersil ICM7233 can be use.

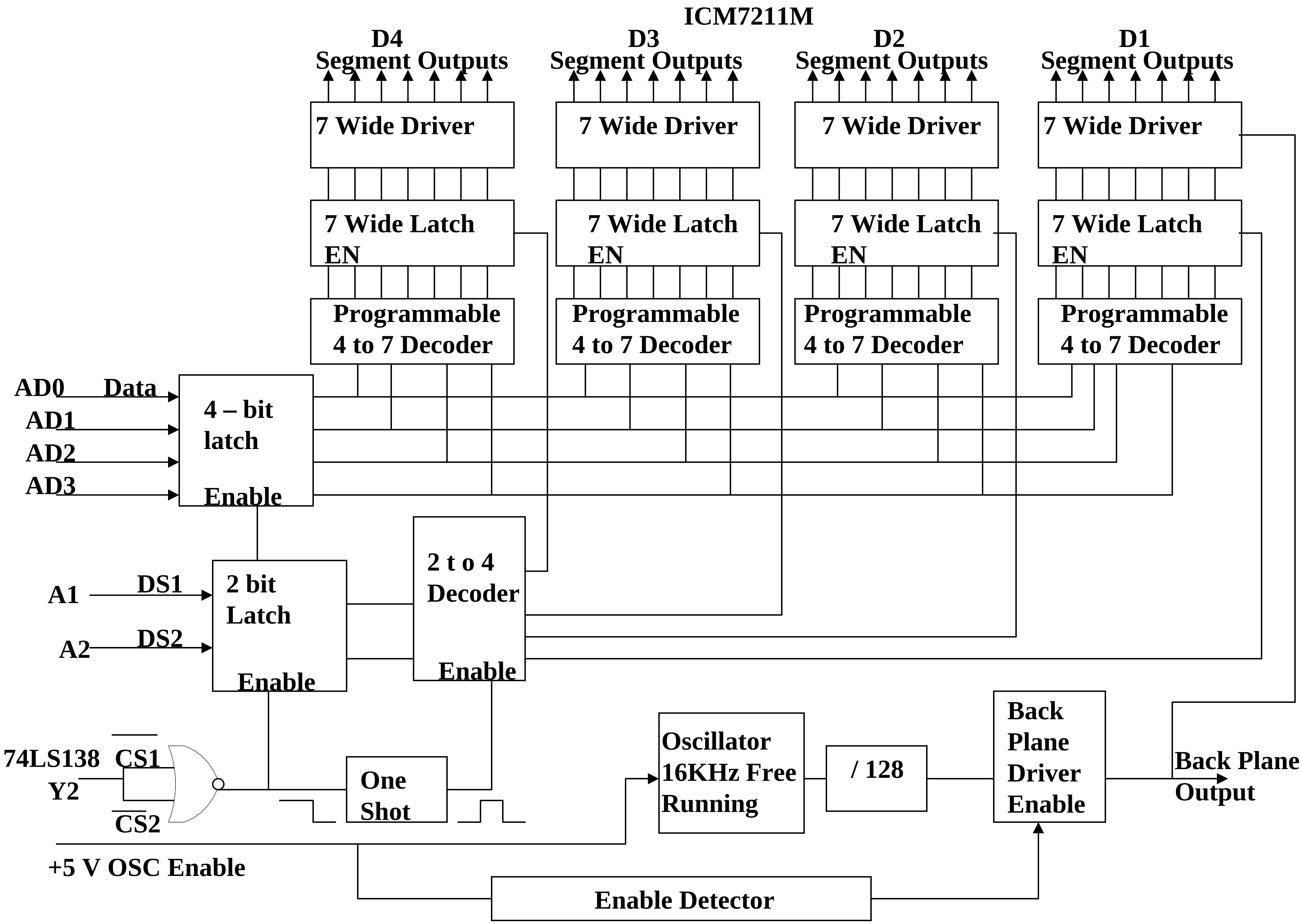


Fig : Circuit for interfacing four LCD digits to an SDK-86 bus using ICM7211M

PIO 8255

- 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.

- 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).

- 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.

- 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.

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

- $\overline{\textbf{CS}}$: This is a chip select line. If this line goes low, it enables the 8255 to respond to $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals, otherwise $\overline{\text{RD}}$ and $\overline{\text{WR}}$ 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 $\overline{\text{RD}}$, $\overline{\text{WR}}$ and $\overline{\text{CS}}$ 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

- **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)

- 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.

- ***Read/Write control logic***: This unit accepts control signals ($\overline{\text{RD}}$, $\overline{\text{WR}}$) 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) $\overline{\textbf{CS}}$ **–** Chipselect : A low on this PIN enables the communication between CPU and 8255.

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

c) $\overline{\mathbf{WR}}$ (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) $\mathbf{A_0}$ and $\mathbf{A_1}$ (Address pins): These pins in conjunction with $\overline{\text{RD}}$ and $\overline{\text{WR}}$ 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.

- 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.

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

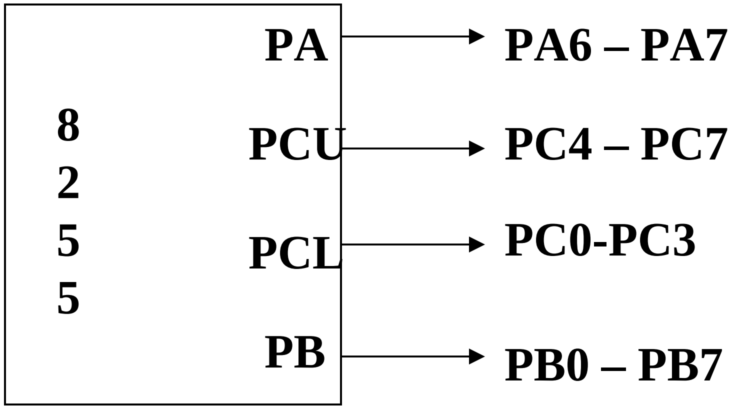
- These are two basic modes of operation of 8255. I/O mode and Bit Set-Reset mode (BSR).
- In I/O mode, the 8255 ports work as programmable I/O ports, while in BSR mode only port C (PC_0-PC_7) can be used to set or reset its individual port bits.
- Under the I/O mode of operation, further there are three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

- ***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



8255

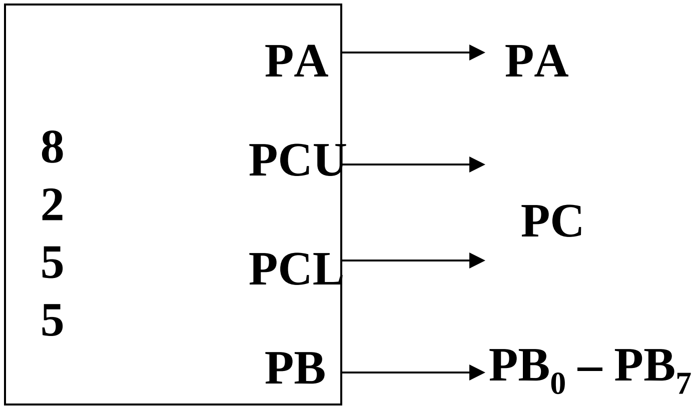
PA → PA6 – PA7

PCU → PC4 – PC7

PCL → PC0-PC3

PB → PB0 – PB7

All Output



8255

PA → PA

PCU → PC

PCL → PC

PB → PB_0 – PB_7

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

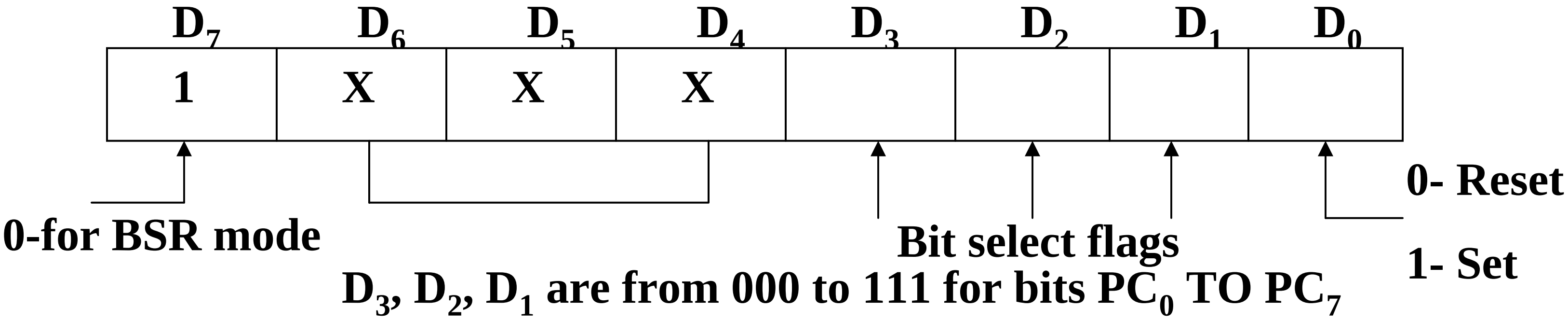
Mode 0

- 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.

- 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.



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

0-for BSR mode

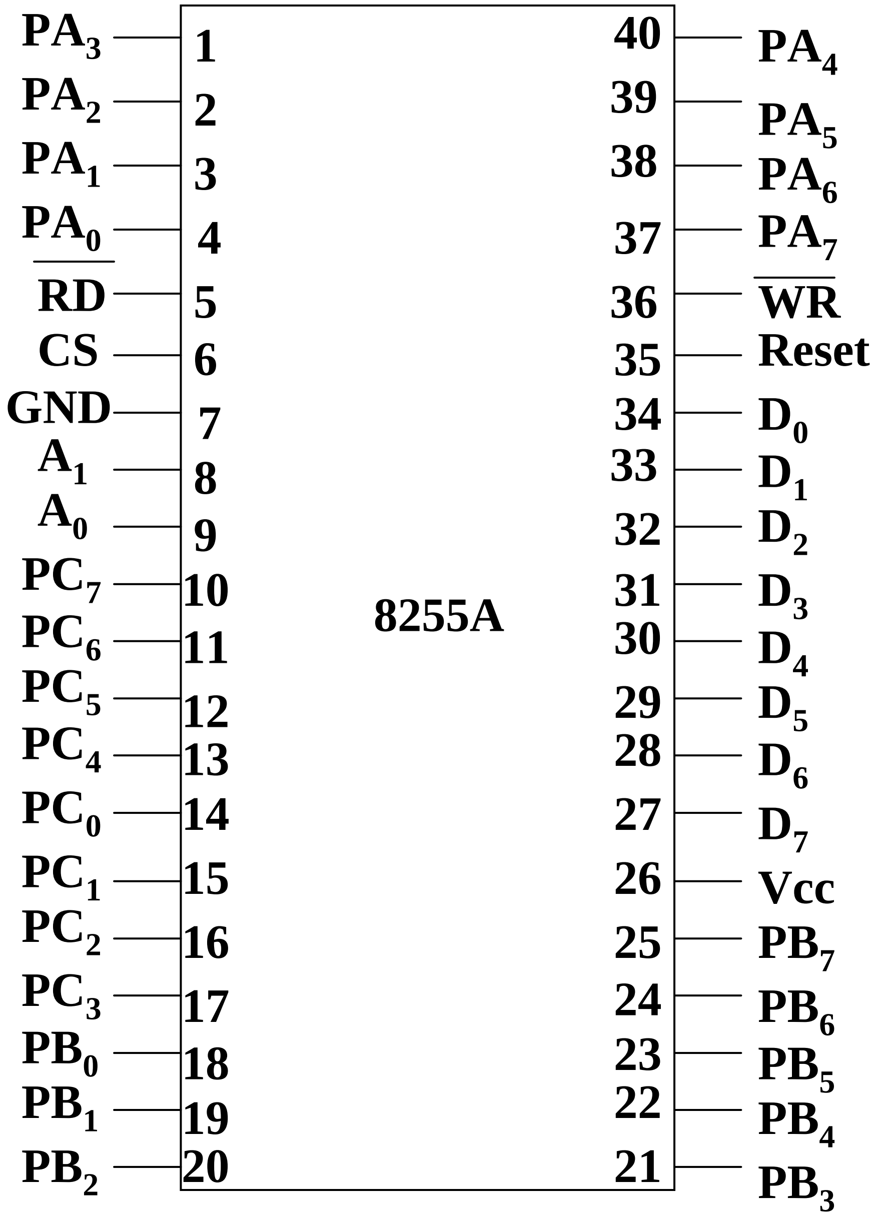
Bit select flags

D_3, D_2, D_1 are from 000 to 111 for bits PC_0 TO PC_7

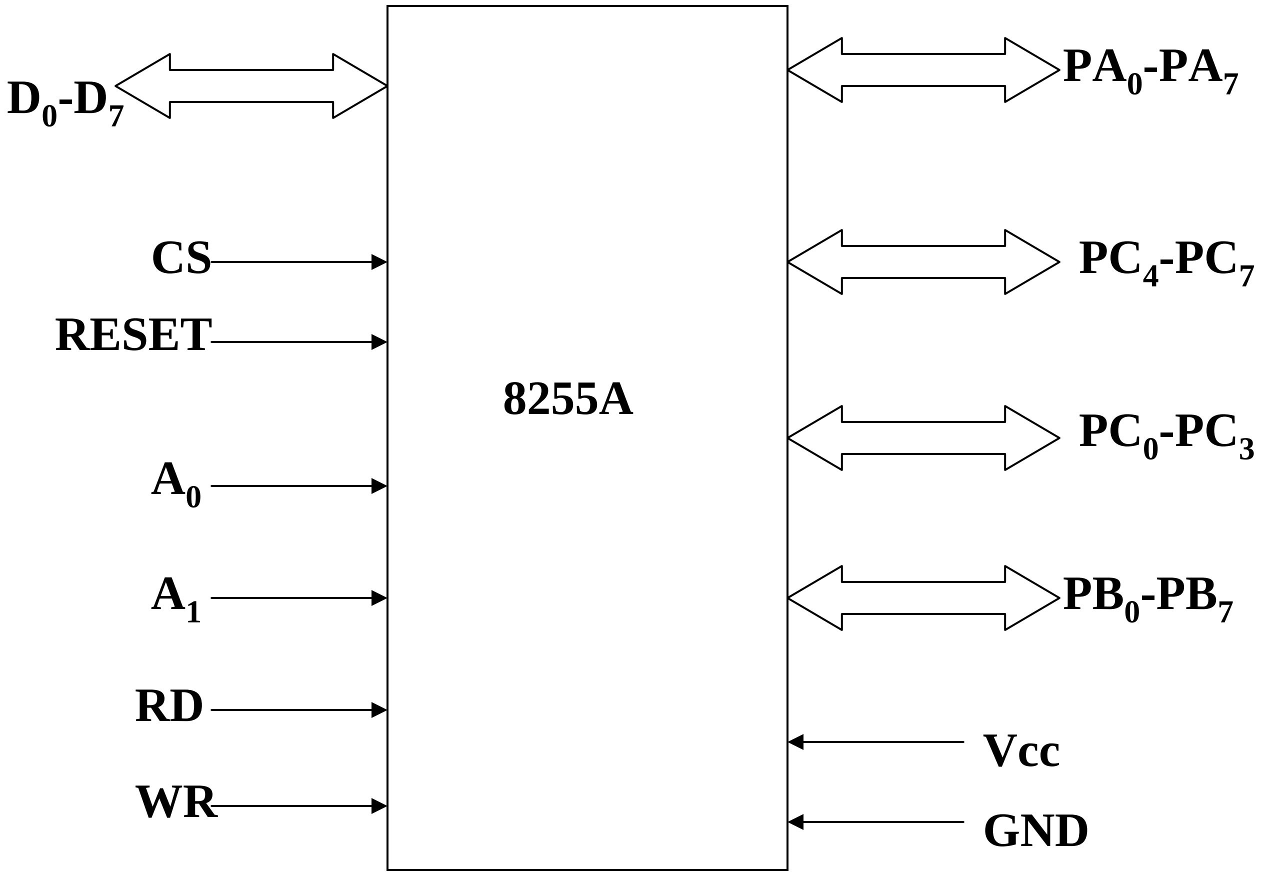
0- Reset

1- Set

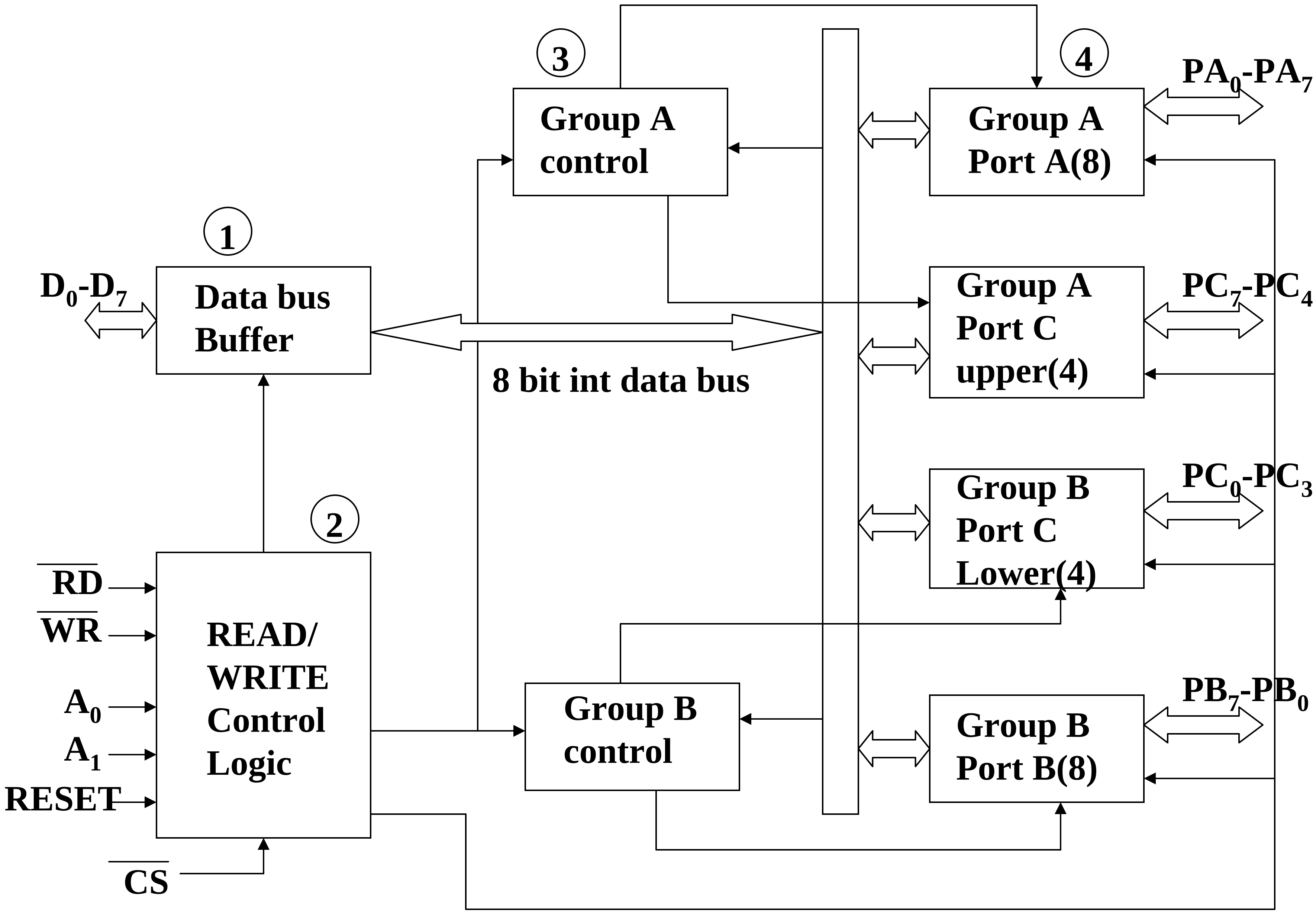
I/O Mode Control Word Register Format and
BSR Mode Control Word Register Format



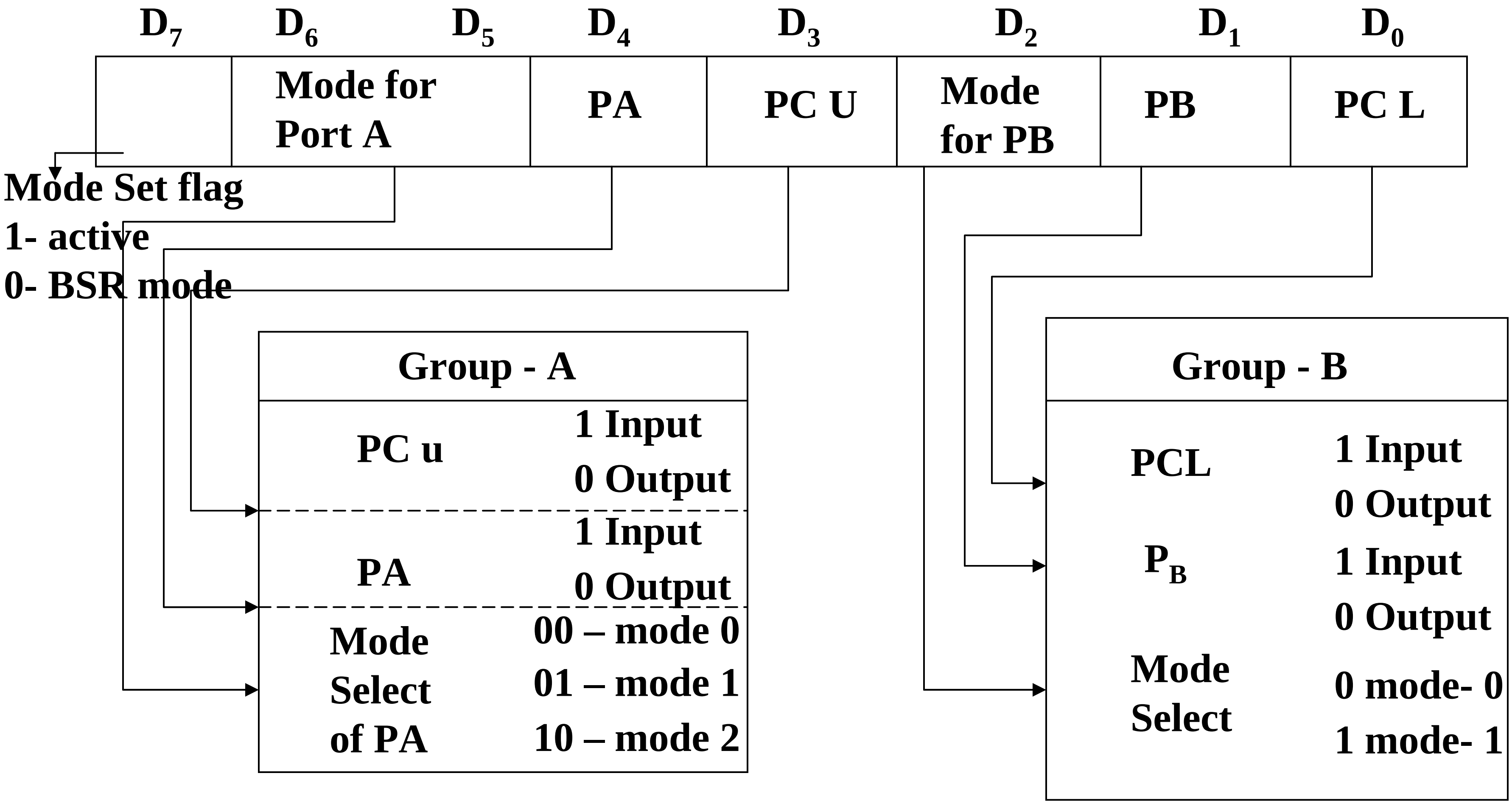
8255A Pin Configuration



Signals of 8255



Block Diagram of 8255



D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0

Mode for Port A

PA

PC U

Mode for PB

PB

PC L

Mode Set flag
1- active
0- BSR mode

Group - A

PC u 1 Input 0 Output

PA 1 Input 0 Output

Mode Select of PA 00 – mode 0 01 – mode 1 10 – mode 2

Group - B

PCL 1 Input 0 Output

P_B 1 Input 0 Output

Mode Select 0 mode- 0 1 mode- 1

Control Word Format of 8255

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:

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.

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

Input control signal definitions (mode 1):

- $\overline{\textbf{STB}}$**(** 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.

- **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.

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 $\overline{\mathrm{WR}}$ signal and reset by a low going edge at the $\overline{\mathrm{ACK}}$ input.
- $\overline{\textbf{ACK}}$ (Acknowledge input) – $\overline{\mathrm{ACK}}$ signal acts as an acknowledgement to be given by an output device. $\overline{\mathrm{ACK}}$ 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.

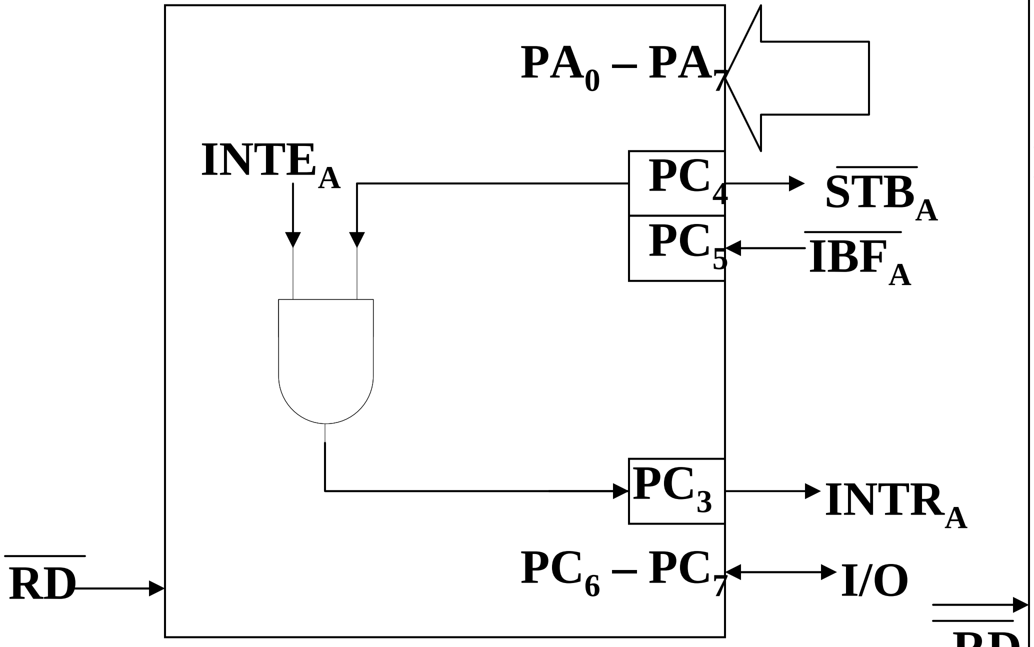
- **INTR** (Interrupt request) – Thus an output signal that can be used to interrupt the CPU when an output device acknowledges the data received from the CPU. INTR is set when ACK, OBF and INTE are 1. It is reset by a falling edge on WR input. The INTEA and INTEB flags are controlled by the bit set-reset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

Input control signal definitions in Mode 1

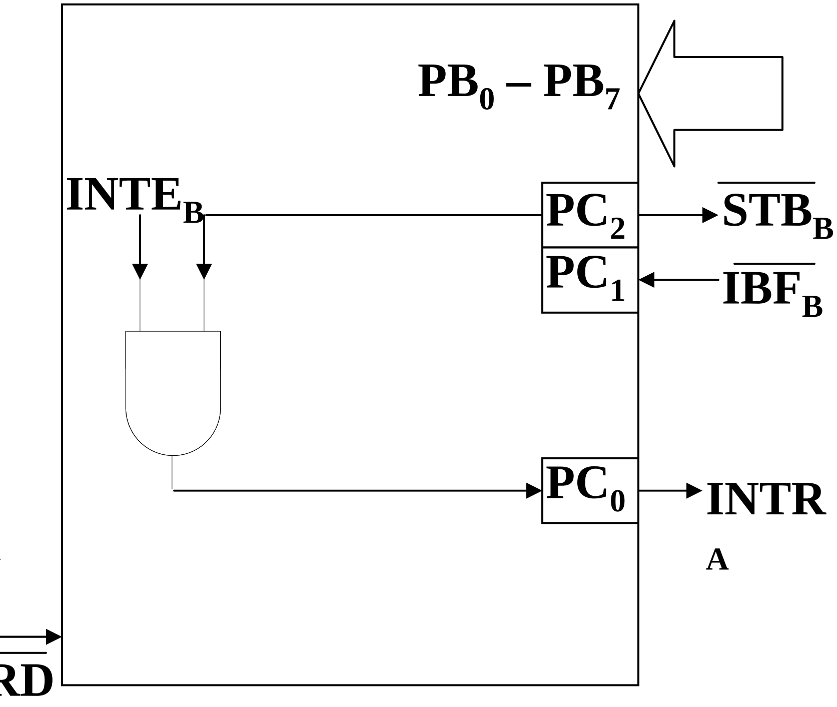
1	0	1	0	1/0	X	X	X
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

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

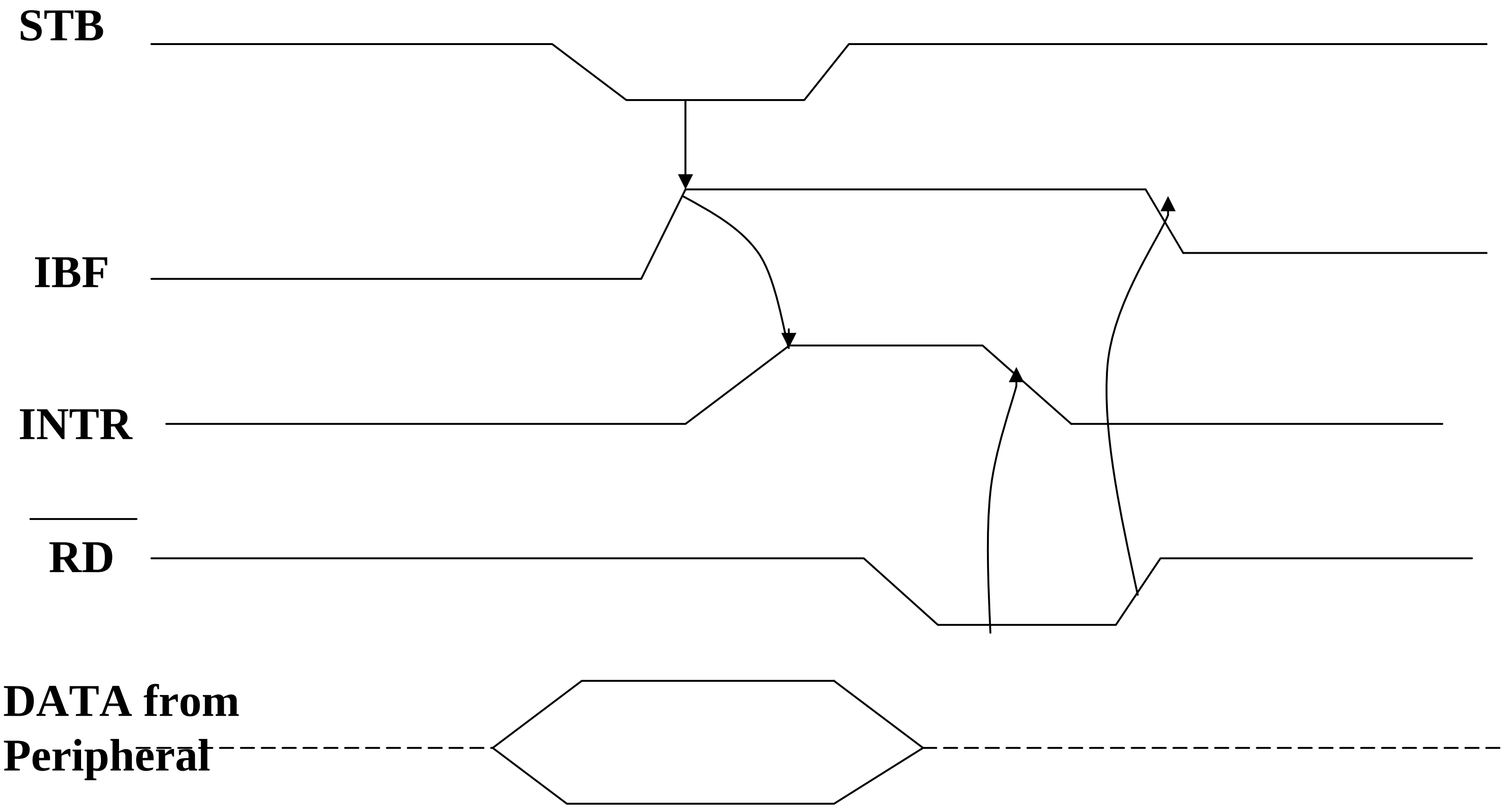
1	X	X	X	X	1	1	X
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0



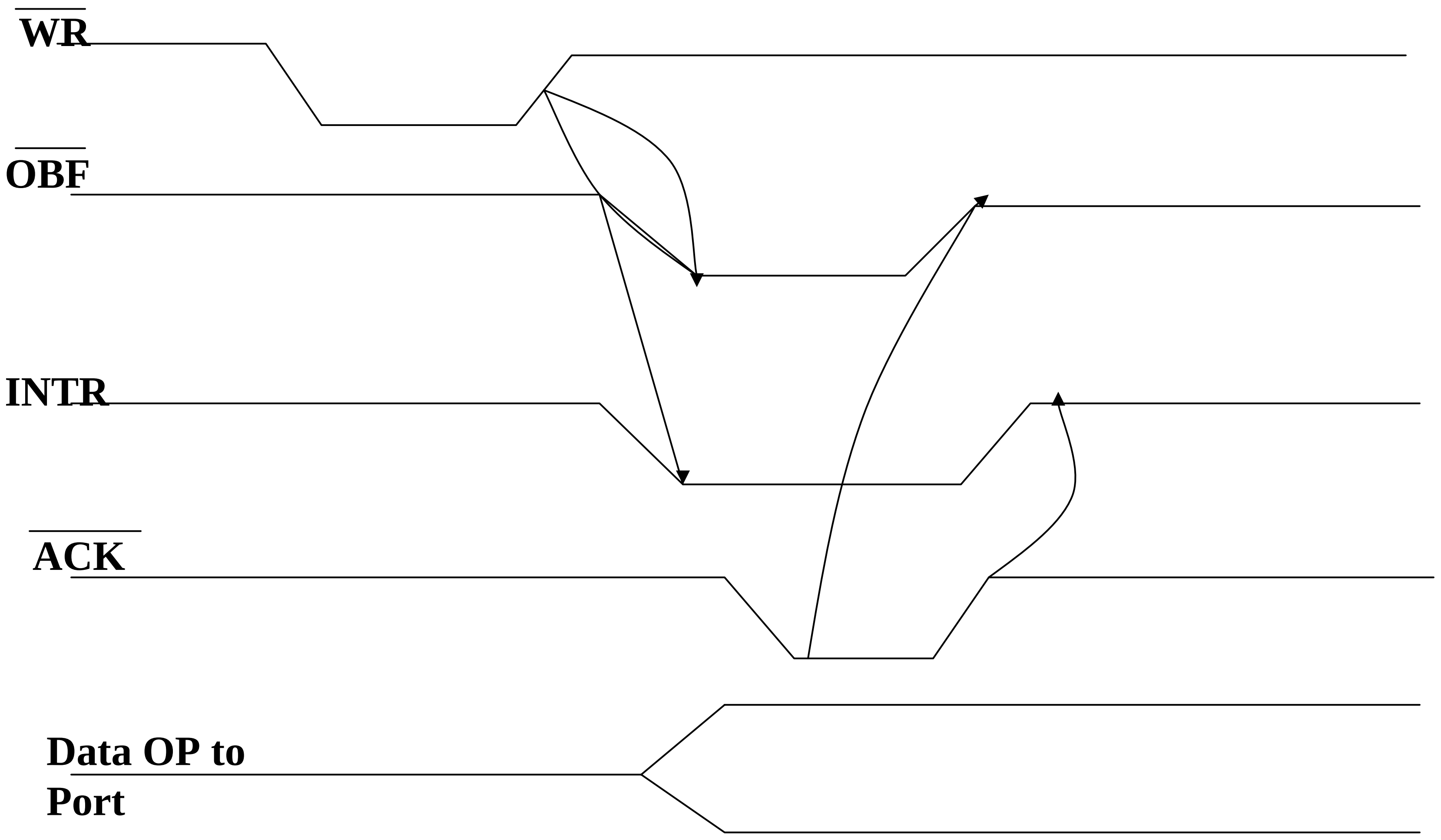
Mode 1 Control Word Group A I/P



Mode 1 Control Word Group B I/P



Mode 1 Strobed Input Data Transfer



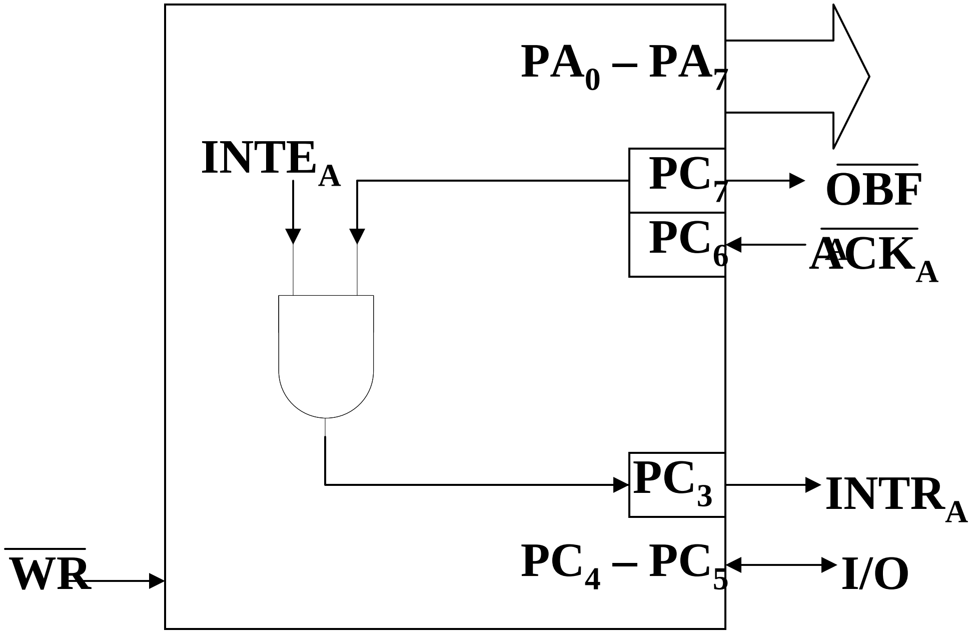
Mode 1 Strobed Data Output

Output control signal definitions Mode 1

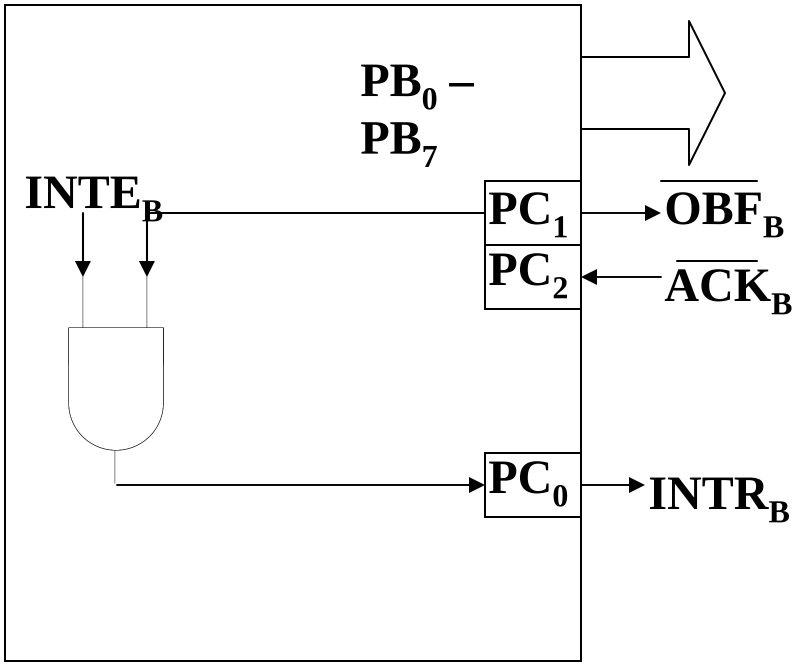
1	0	1	0	1/0	X	X	X
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

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

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



Mode 1 Control Word Group A



Mode 1 Control Word Group B

- **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.

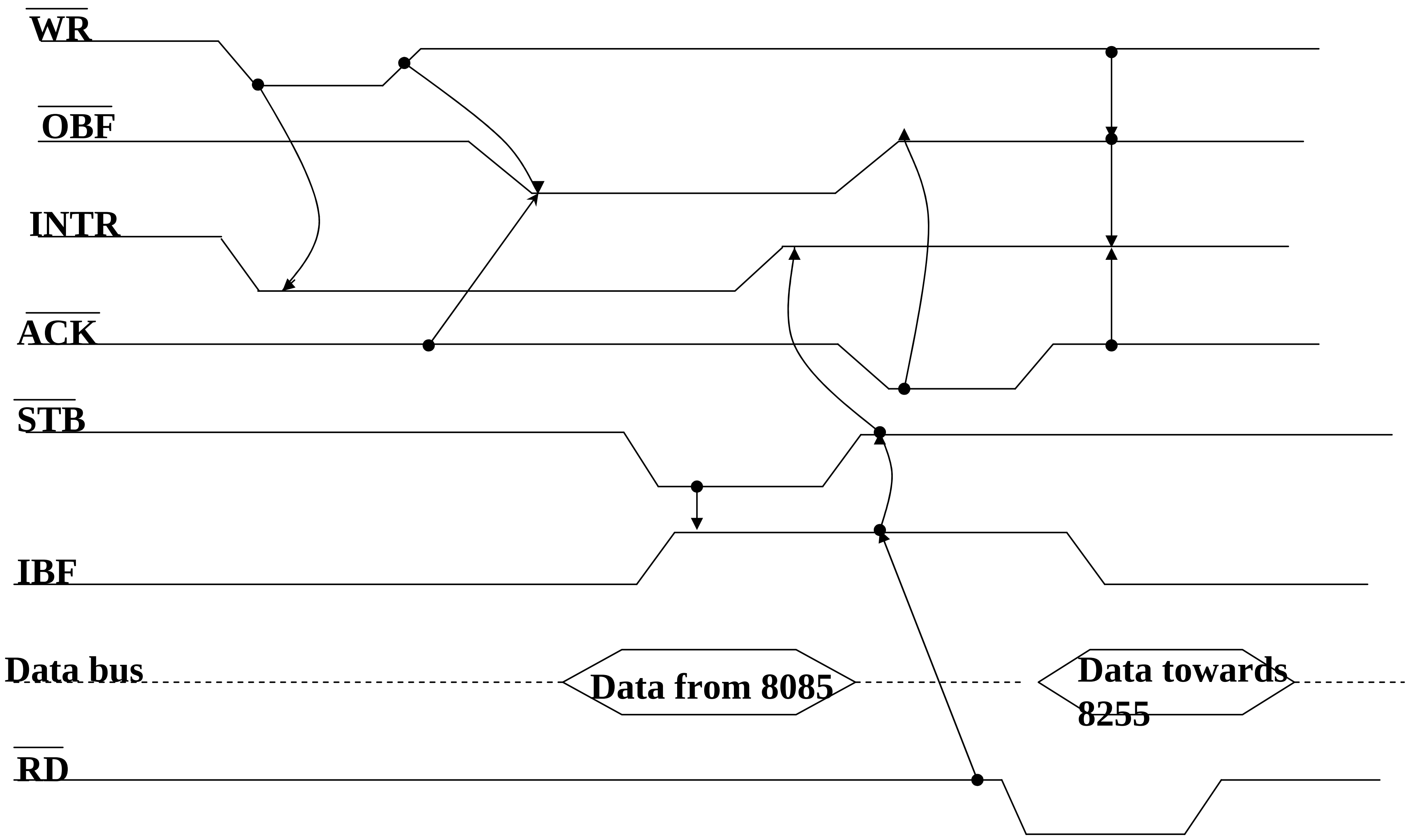
- The Salient features of Mode 2 of 8255 are listed as follows:

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.

- ***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.

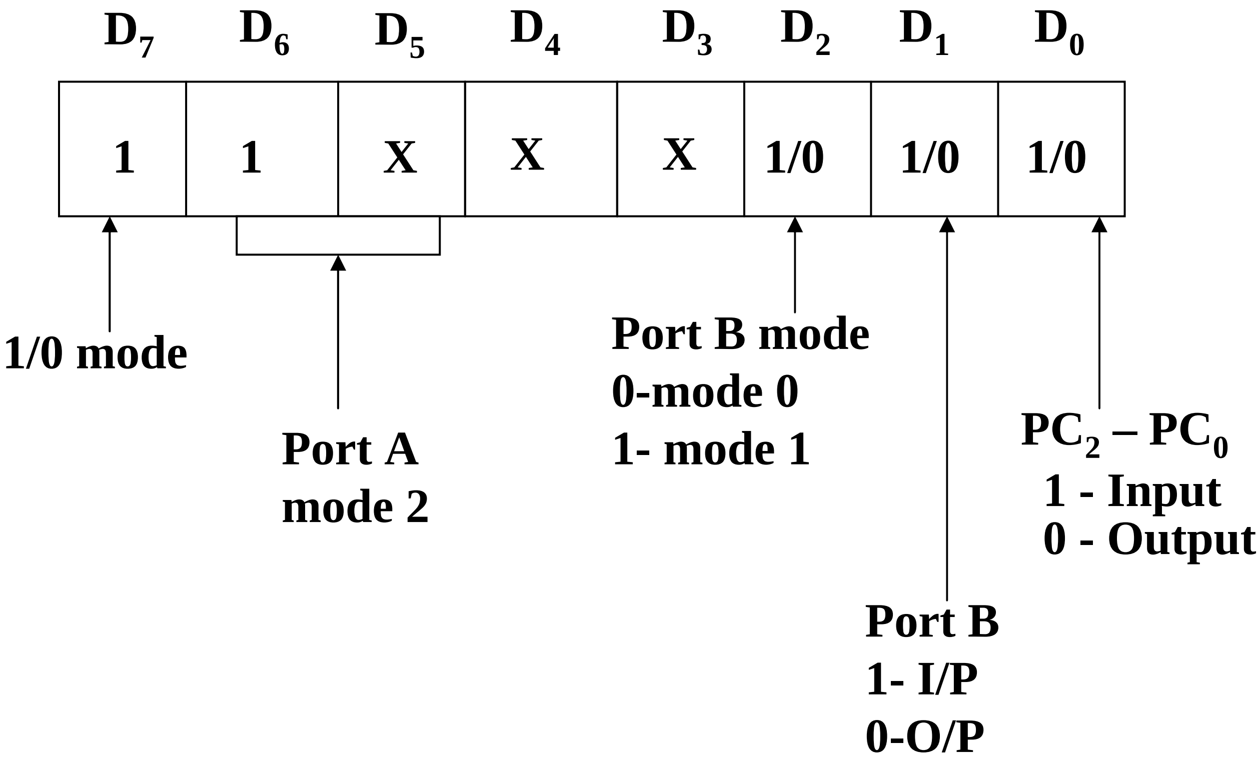
- $\overline{\mathbf{ACK}}$ (Acknowledge) This control input, when falls to logic low level, acknowledges that the previous data byte is received by the destination and next byte may be sent by the processor. This signal enables the internal tristate buffers to send the next data byte on port A.
- **INTE1**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

- **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: $\overline{\text{WR}}$ must occur before $\overline{\text{ACK}}$ and $\overline{\text{STB}}$ must be activated before $\overline{\text{RD}}$.



Mode 2 Bidirectional Data Transfer

- 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.



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

1/0 mode

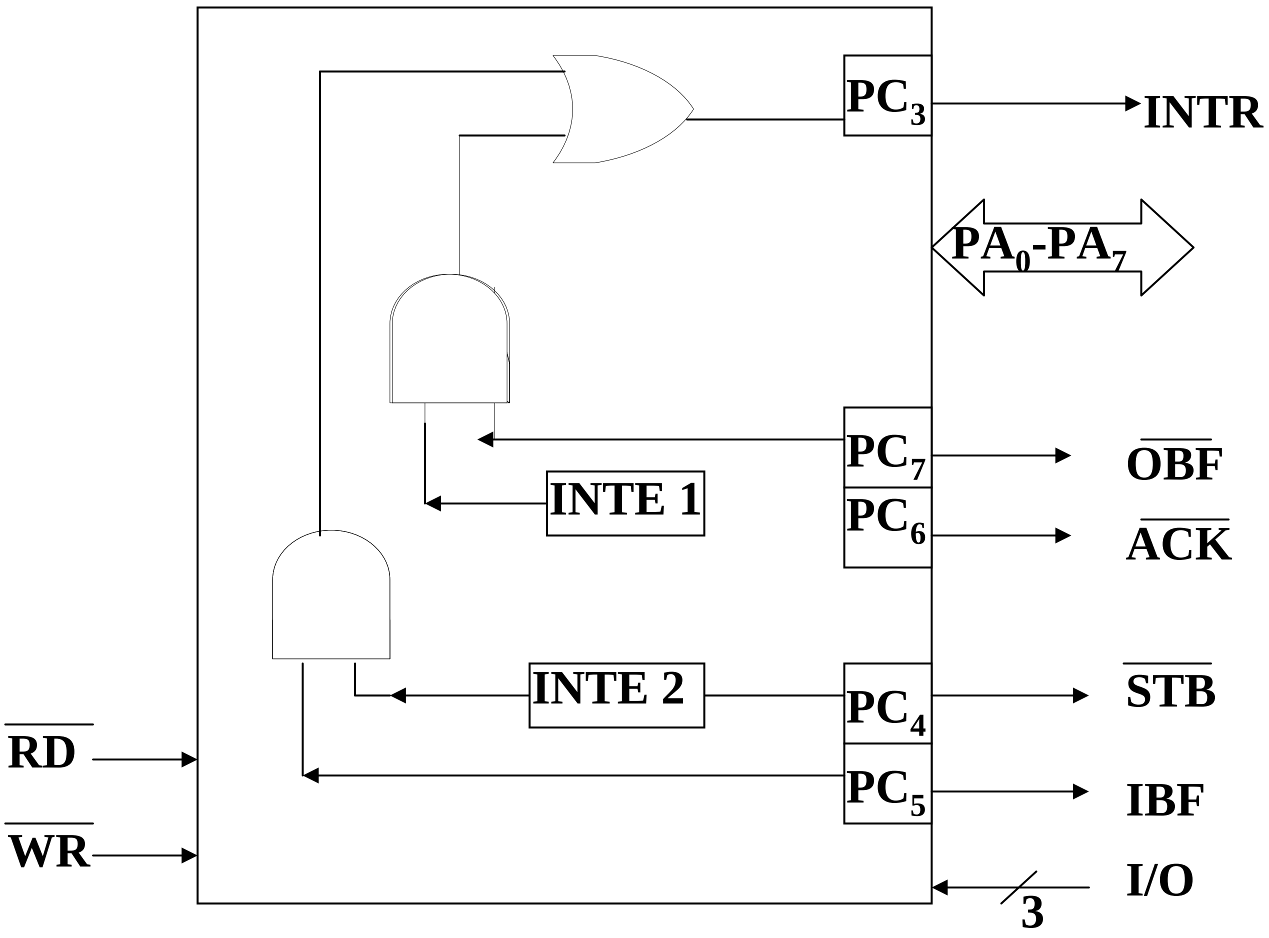
Port A
mode 2

Port B mode
0-mode 0
1- mode 1

Port B
1- I/P
0-O/P

$PC_2 – PC_0$
1 - Input
0 - Output

Mode 2 control word



PC$_3$
INTR
PA$_0$-PA$_7$
PC$_7$
$\overline{OBF}$
INTE 1
PC$_6$
$\overline{ACK}$
INTE 2
$\overline{STB}$
PC$_4$
$\overline{RD}$
PC$_5$
IBF
$\overline{WR}$
I/O
3

Mode 2 pins

Interfacing a Microprocessor To Keyboard

- When you press a key on your computer, you are activating a switch. There are many different ways of making these switches. An overview of the construction and operation of some of the most common types.

1. ***Mechanical key switches:*** In mechanical-switch keys, two pieces of metal are pushed together when you press the key. The actual switch elements are often made of a phosphor-bronze alloy with gold platting on the contact areas. The key switch usually contains a spring to return the key to the nonpressed position and perhaps a small piece of foam to help damp out bouncing.

Next Page

- Some mechanical key switches now consist of a molded silicon dome with a small piece of conductive rubber foam short two trace on the printed-circuit board to produce the key pressed signal.
- Mechanical switches are relatively inexpensive but they have several disadvantages. First, they suffer from contact bounce. A pressed key may make and break contact several times before it makes solid contact.
- Second, the contacts may become oxidized or dirty with age so they no longer make a dependable connection.

Next Page

- Higher-quality mechanical switches typically have a rated life time of about 1 million keystrokes. The silicone dome type typically last 25 million keystrokes.

2. ***Membrane key switches:*** These switches are really a special type of mechanical switches. They consist of a three-layer plastic or rubber sandwich.

- The top layer has a conductive line of silver ink running under each key position. The bottom layer has a conductive line of silver ink running under each column of keys.

Next Page

- When u press a key, you push the top ink line through the hole to contact the bottom ink line.
- The advantages of membrane keyboards is that they can be made as very thin, sealed units.
- They are often used on cash registers in fast food restaurants. The lifetime of membrane keyboards varies over a wide range.

3. ***Capacitive key switches:*** A capacitive keyswitch has two small metal plates on the printed circuit board and another metal plate on the bottom of a piece of foam.

Next Page

- When u press the key, the movable plate is pushed closer to fixed plate. This changes the capacitance between the fixed plates. Sense amplifier circuitry detects this change in capacitance and produce a logic level signal that indicates a key has been pressed.
- The big advantages of a capacitive switch is that it has no mechanical contacts to become oxidized or dirty.
- A small disadvantage is the specified circuitry needed to detect the change in capacitance.
- Capacitive keyswitches typically have a rated lifetime of about 20 million keystrokes.

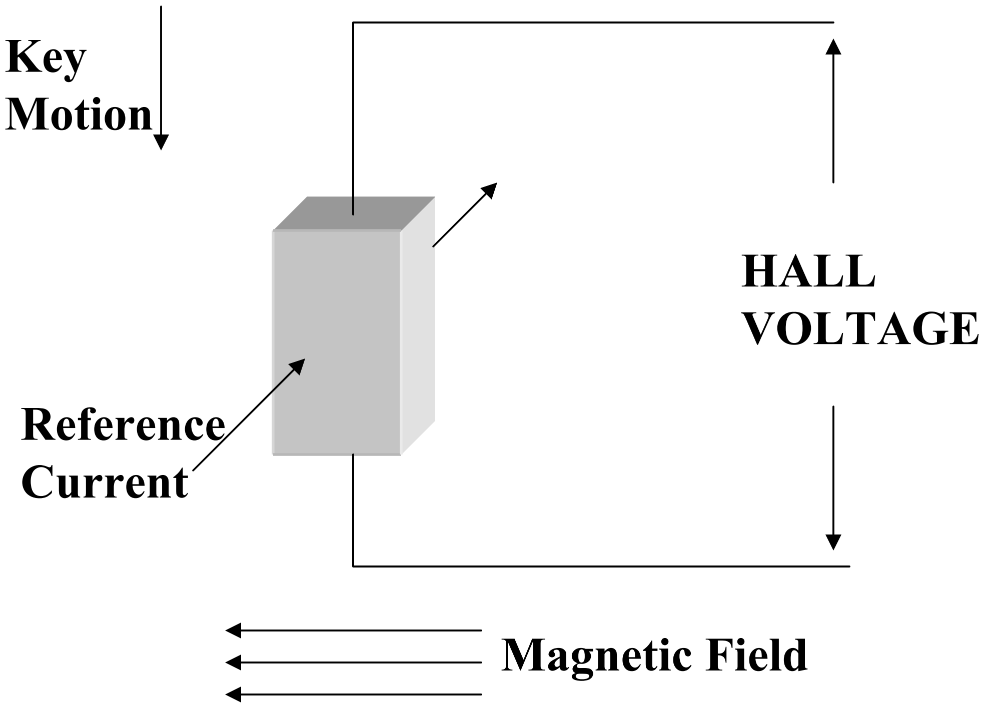
Next Page

4. ***Hall effect keyswitches:*** This is another type of switch which has no mechanical contact. It takes advantage of the deflection of a moving charge by a magnetic field.

- A reference current is passed through a semiconductor crystal between two opposing faces. When a key is pressed, the crystal is moved through a magnetic field which has its flux lines perpendicular to the direction of current flow in the crystal.
- Moving the crystal through the magnetic field causes a small voltage to be developed between two of the other opposing faces of the crystal.

Next Page

- This voltage is amplified and used to indicate that a key has been pressed. Hall effect sensors are also used to detect motion in many electrically controlled machines.
- Hall effect keyboards are more expensive because of the more complex switch mechanism, but they are very dependable and have typically rated lifetime of 100 million or more keystrokes.



HALL EFFECT

Keyboard Circuit Connections and Interfacing

- In most keyboards, the keyswitches are connecting in a matrix of rows and columns, as shown in fig.
- We will use simple mechanical switches for our examples, but the principle is same for other type of switches.
- Getting meaningful data from a keyboard, it requires the following three major tasks:

1. Detect a keypress.
2. Debounce the keypress.
3. Encode the keypress

Next Page

- Three tasks can be done with hardware, software, or a combination of two, depending on the application.

1. **Software Keyboard Interfacing:**

- ***Circuit connection and algorithm :*** The following fig (a) shows how a hexadecimal keypad can be connected to a couple of microcomputer ports so the three interfacing tasks can be done as part of a program.
- The rows of the matrix are connected to four output port lines. The column lines of matrix are connected to four input-port lines. To make the program simpler, the row lines are also connected to four input lines.

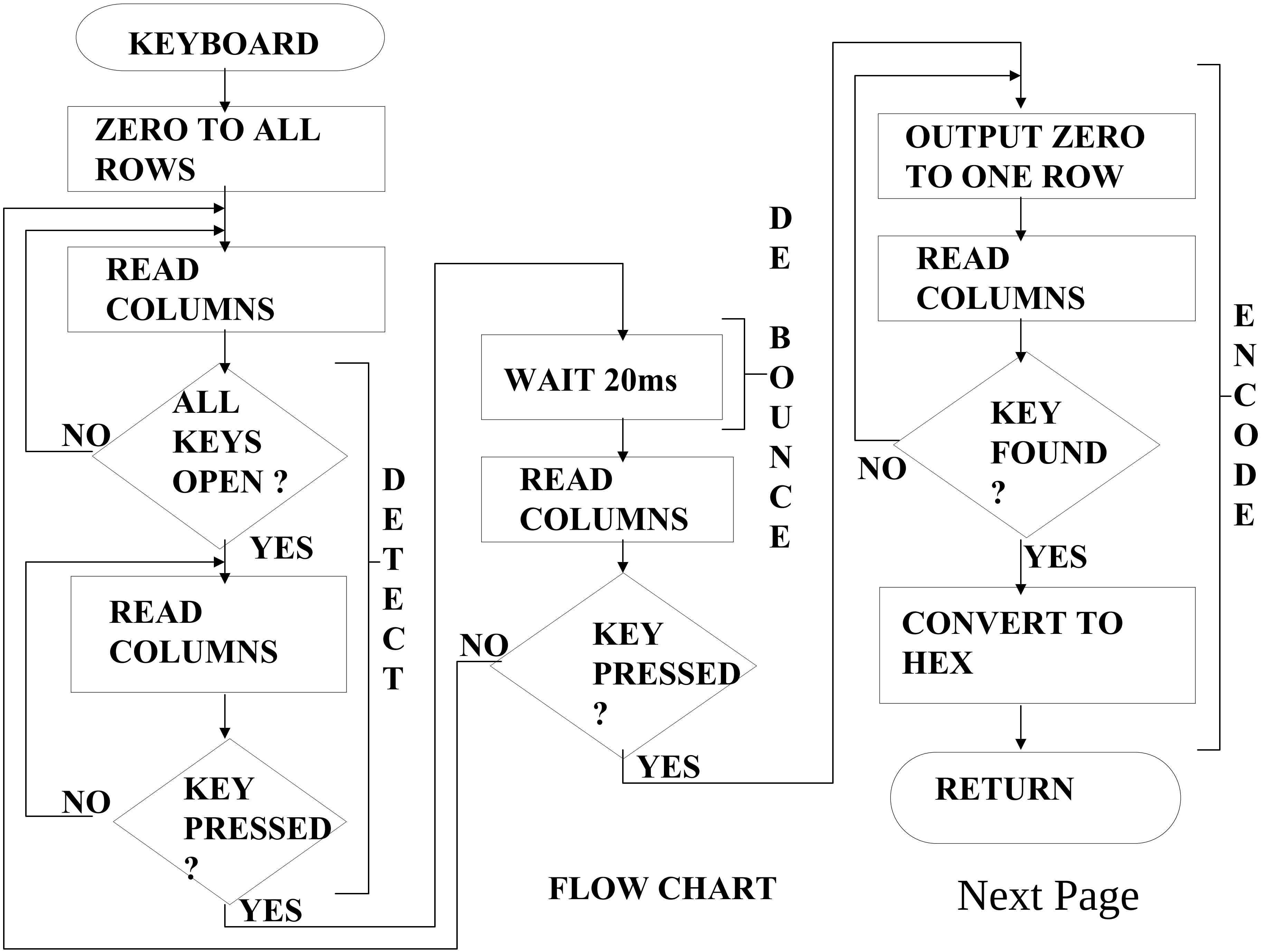
Next Page

- When no keys are pressed, the column lines are held high by the pull-up resistor connected to +5V. Pressing a key connects a row to a column. If a low is output on a row and a key in that row is pressed, then the low will appear on the column which contains that key and can be detected on the input port.
- If you know the row and column of the pressed key, you then know which key was pressed, and you can convert this information into any code you want to represent that key.

Next Page

- The following flow chart for a procedure to detect, debounce and produce the hex code for a pressed key.
- An easy way to detect if any key in the matrix is pressed is to output 0’s to all rows and then check the column to see if a pressed key has connected a low to a column.
- In the algorithm we first output lows to all the rows and check the columns over and over until the column are all high. This is done before the previous key has been released before looking for the next one. In the standard keyboard terminology, this is called two-key lockout.

Next Page



FLOW CHART

Next Page

- Once the columns are found to be all high, the program enters another loop, which waits until a low appears on one of the columns, indicating that a key has been pressed. This second loop does the detect task for us. A simple 20-ms delay procedure then does the debounce task.
- After the debounce time, another check is made to see if the key is still pressed. If the columns are now all high, then no key is pressed and the initial detection was caused by a noise pulse or a light brushing past a key. If any of the columns are still low, then the assumption is made that it was a valid keypress.

Next Page

- The final task is to determine the row and column of the pressed key and convert this row and column information to the hex code for the pressed key. To get the row and column information, a low is output to one row and the column are read. If none of the columns is low, the pressed key is not in that row. So the low is rotated to the next row and the column are checked again. The process is repeated until a low on a row produces a low on one of the column.
- The pressed key then is in the row which is low at that time.

Next Page

- The connection fig shows the byte read in from the input port will contain a 4-bit code which represents the row of the pressed key and a 4-bit code which represent the column of the pressed key.
- ***Error trapping:*** The concept of detecting some error condition such as “ no match found” is called error trapping. Error trapping is a very important part of real programs. Even in simple programs, think what might happen with no error trap if two keys in the same row were pressed at exactly at the same time and a column code with two lows in it was produced.

Next Page

- This code would not match any of the row-column codes in the table, so after all the values in the table were checked, assigned register in program would be decremented from 0000H to FFFFH. The compare decrement cycle would continue through 65,536 memory locations until, by change the value in a memory location matched the row-column code. The contents of the lower byte register at hat point would be passed back to the calling routine. The changes are 1 in 256 that would be the correct value for one of the pressed keys. You should keep an error trap in a program whenever there is a chance for it.

Next Page

2. **Keyboard Interfacing with Hardware:** For the system where the CPU is too busy to be bothered doing these tasks in software, an external device is used to do them.

- One of a MOS device which can be do this is the General Instruments AY5-2376 which can be connected to the rows and columns of a keyboard switch matrix.
- The AY5-2376 independently detects a keypress by cycling a low down through the rows and checking the columns. When it finds a key pressed, it waits a debounce time.

Next Page

- If the key is still pressed after the debounce time, the AY5-2376 produces the 8-bit code for the pressed key and send it out to microcomputer port on 8 parallel lines. The microcomputer knows that a valid ASCII code is on the data lines, the AY5-2376 outputs a strobe pulse.
- The microcomputer can detect this strobe pulse and read in ASCII code on a polled basis or it can detect the strobe pulse on an interrupt basis.
- With the interrupt method the microcomputer doesn't have to pay any attention to the keyboard until it receives an interrupt signal.

Next Page

- So this method uses very little of the microcomputer time. The AY5-2376 has a feature called *two-key rollover*. This means that if two keys are pressed at nearly the same time, each key will be detected, debounced and converted to ASCII.
- The ASCII code for the first key and a strobe signal for it will be sent out then the ASCII code for the second key and a strobe signal for it will be sent out and compare this with two-key lockout.

Next Page

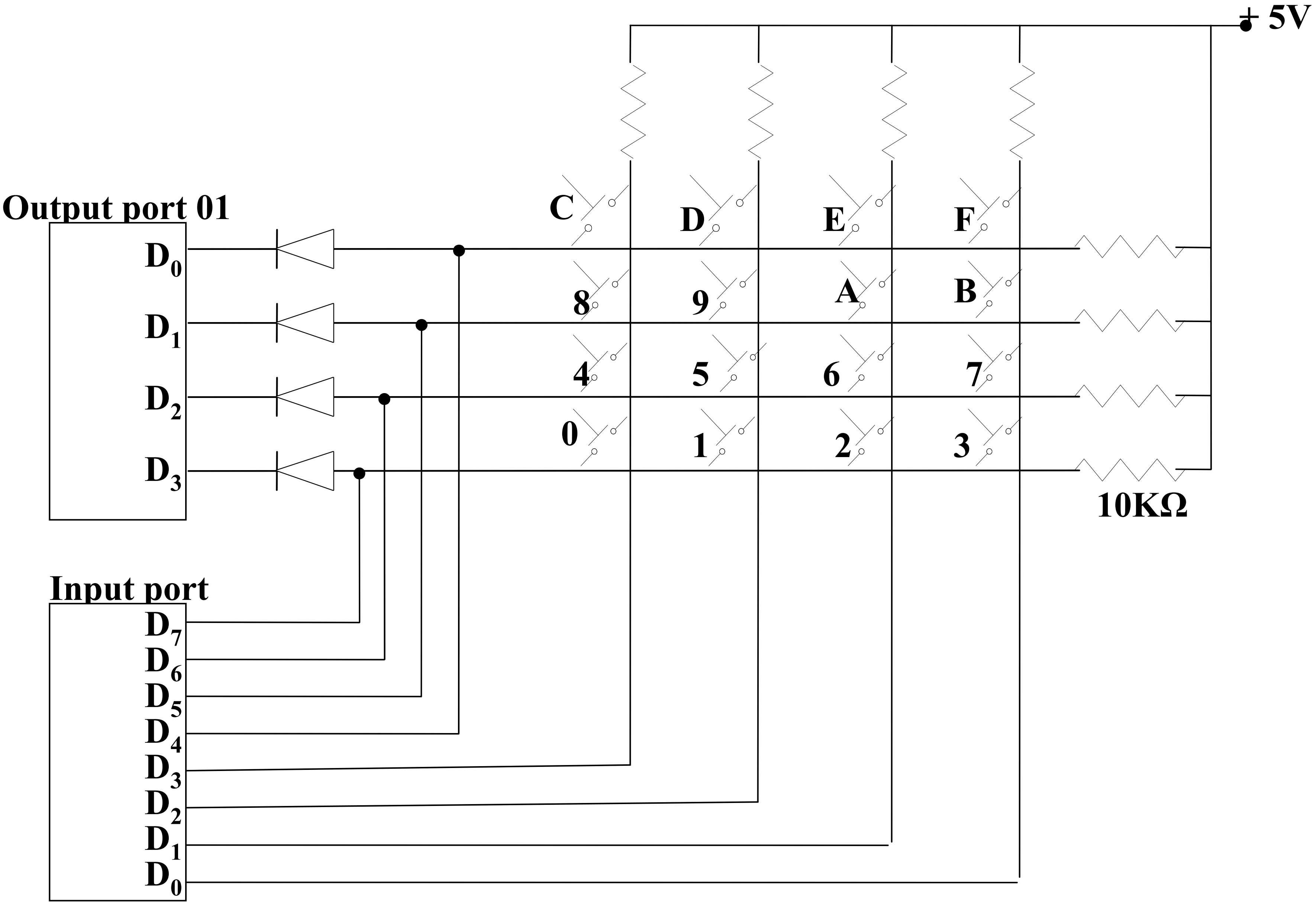
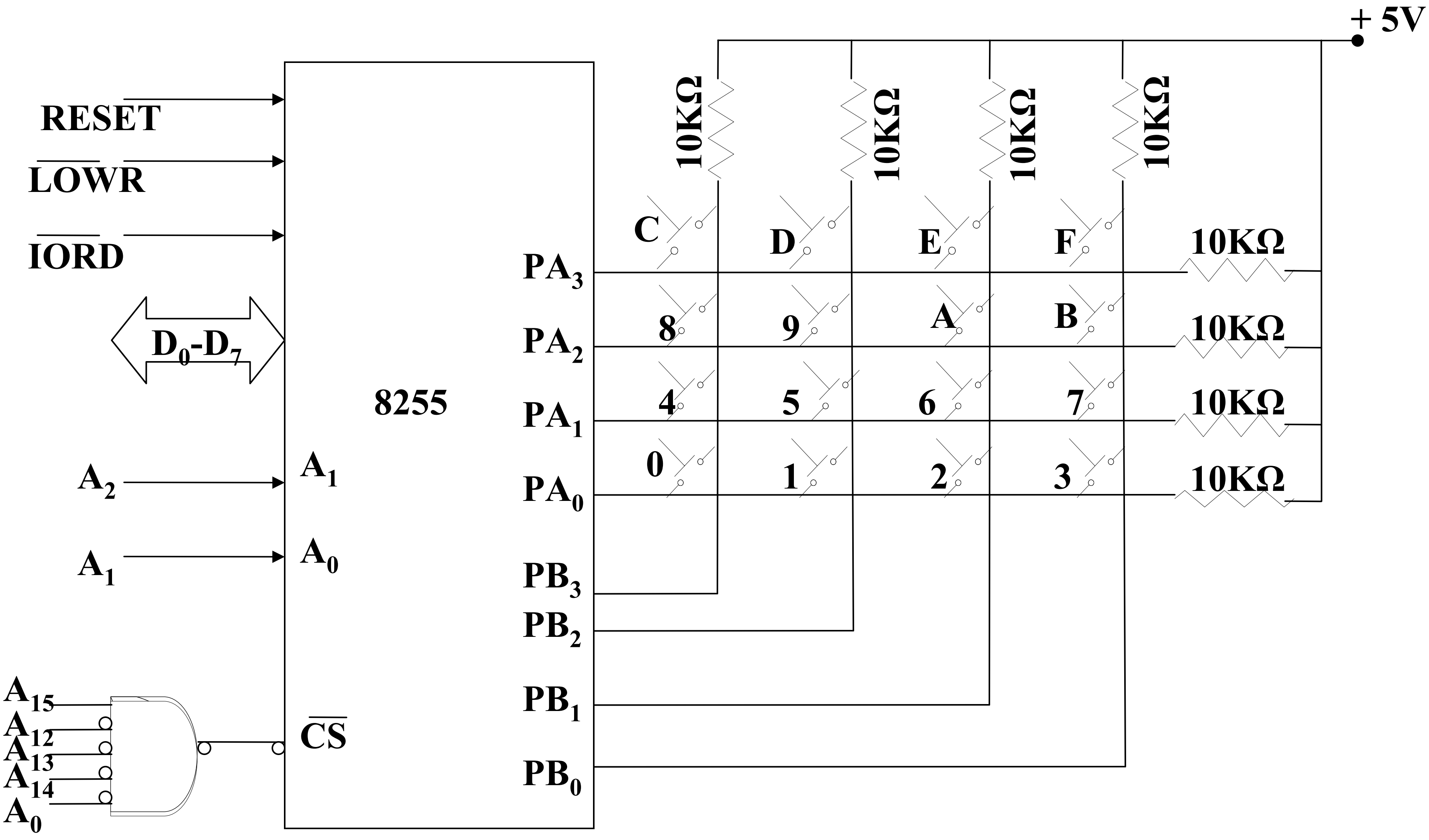


Fig: (a) Port connections

Example

- Interface a 4 * 4 keyboard with 8086 using 8255 an write an ALP for detecting a key closure and return the key code in AL. The debounce period for a key is 10ms. Use software debouncing technique. DEBOUNCE is an available 10ms delay routine.
- **Solution**: Port A is used as output port for selecting a row of keys while Port B is used as an input port for sensing a closed key. Thus the keyboard lines are selected one by one through port A and the port B lines are polled continuously till a key closure is sensed. The routine DEBOUNCE is called for key debouncing. The key code is depending upon the selected row and a low sensed column.

Next Page

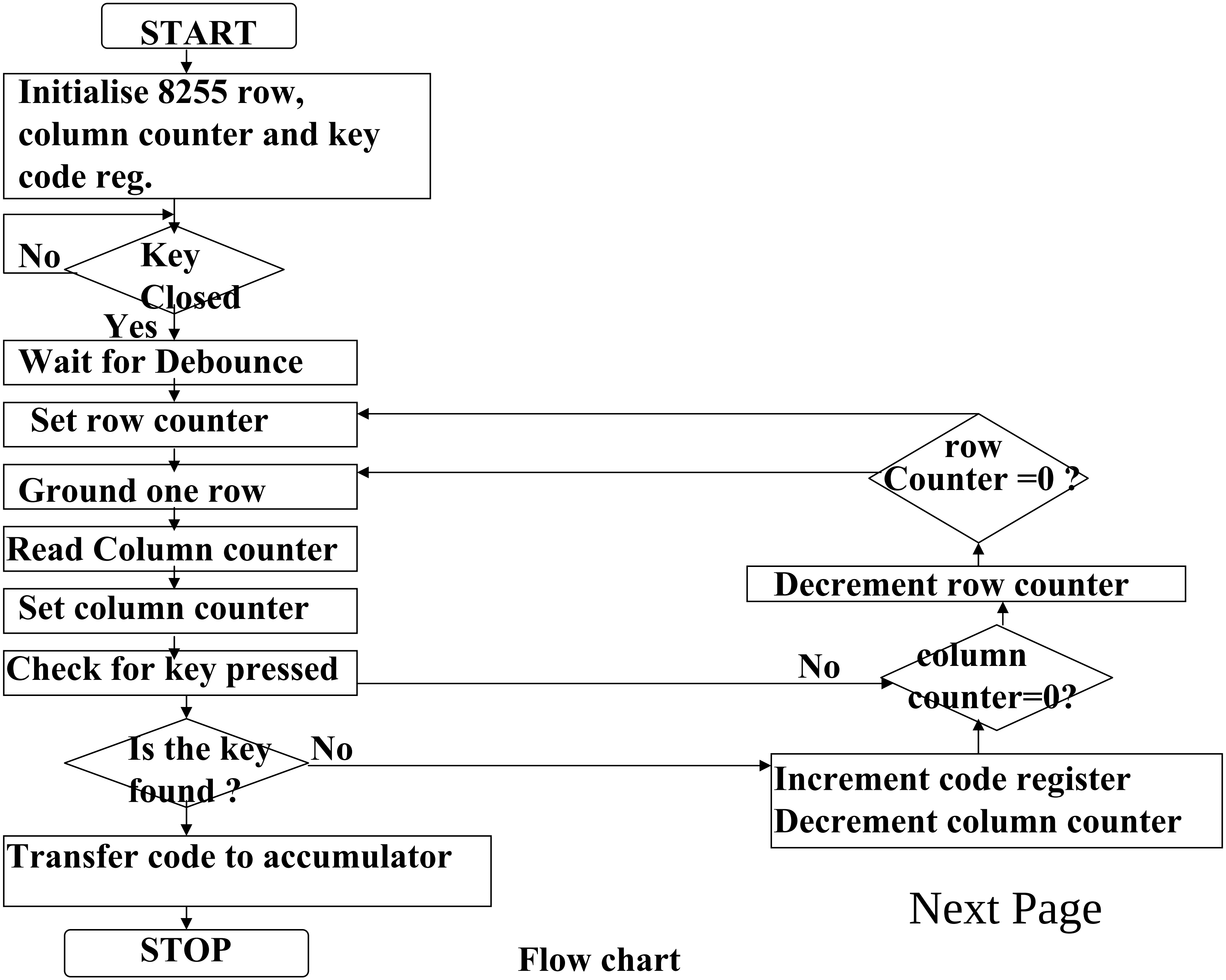


Interfacing 4 * 4 Keyboard

Next Page

- The higher order lines of port A and port B are left unused. The address of port A and port B will respectively 8000H and 8002H while address of CWR will be 8006H. The flow chart of the complete program is as given. The control word for this problem will be 82H. Code segment CS is used for storing the program code.
- **Key Debounce** : Whenever a mechanical push-button is pressed or released once, the mechanical components of the key do not change the position smoothly, rather it generates a transient response .

Next Page



START

Initialise 8255 row, column counter and key code reg.

Key Closed

No

Yes

Wait for Debounce

Set row counter

Ground one row

Read Column counter

Set column counter

Check for key pressed

Is the key found ?

No

Transfer code to accumulator

STOP

row Counter =0 ?

Decrement row counter

column counter=0?

No

Increment code register Decrement column counter

Next Page

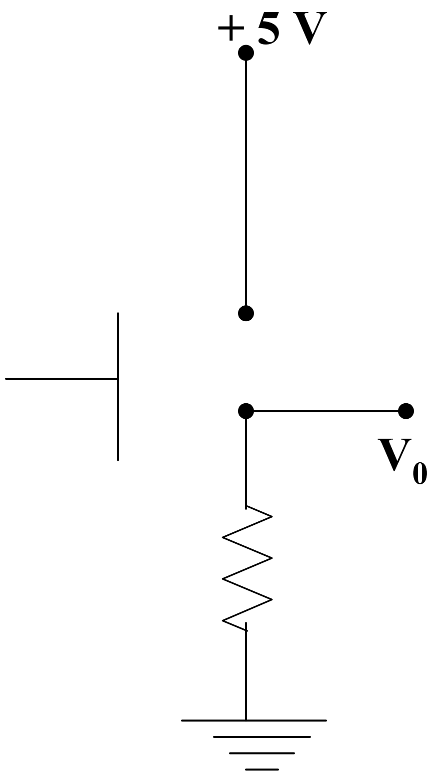
Flow chart

- These transient variations may be interpreted as the multiple key pressure and responded accordingly by the microprocessor system.
- To avoid this problem, two schemes are suggested: the first one utilizes a bistable multivibrator at the output of the key to debounce .
- The other scheme suggests that the microprocessor should be made to wait for the transient period (usually 10ms), so that the transient response settles down and reaches a steady state.

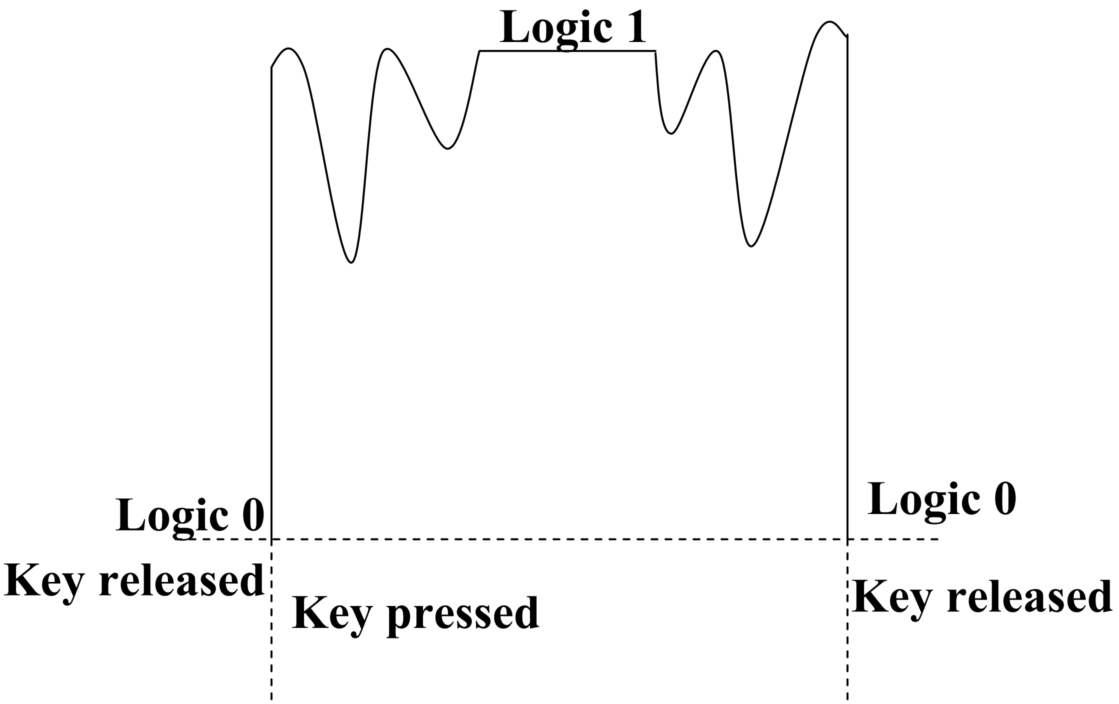
Next Page

- A logic ‘0’ will be read by the microprocessor when the key is pressed.
- In a number of high precision applications, a designer may have two options- the first is to have more than one 8-bit port, read (write) the port one by one and then from the multibyte data, the second option allows forming 16-bit ports using two 8-bit ports and use 16-bit read or write operations.

Next Page



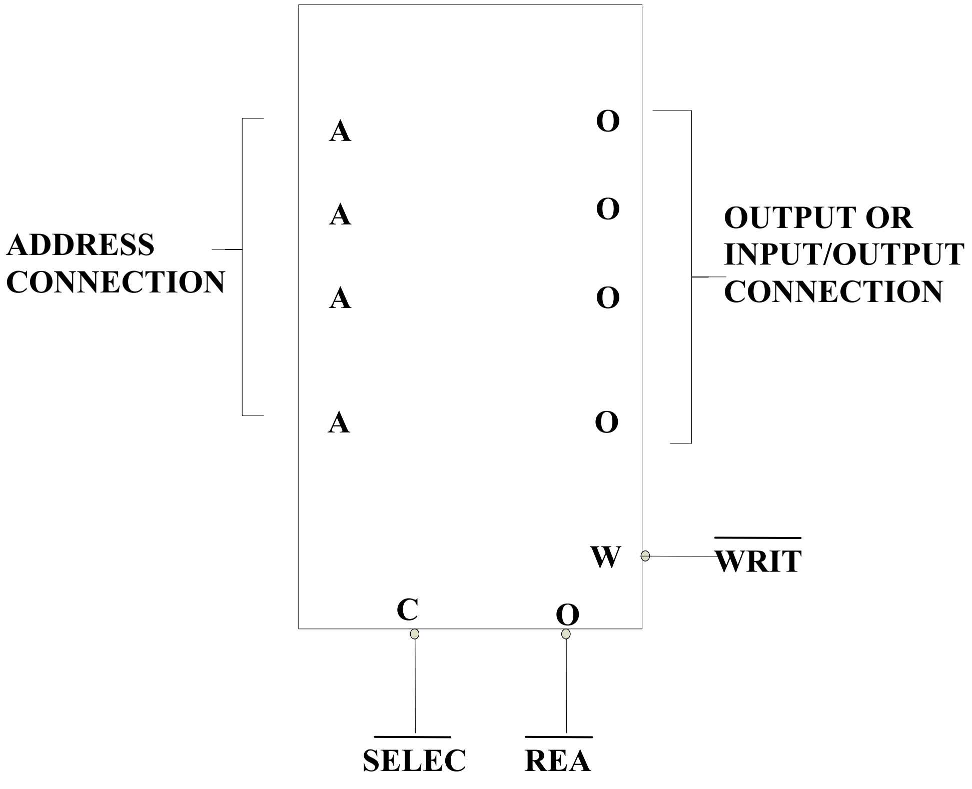
A Mechanical Key



Response

Interface

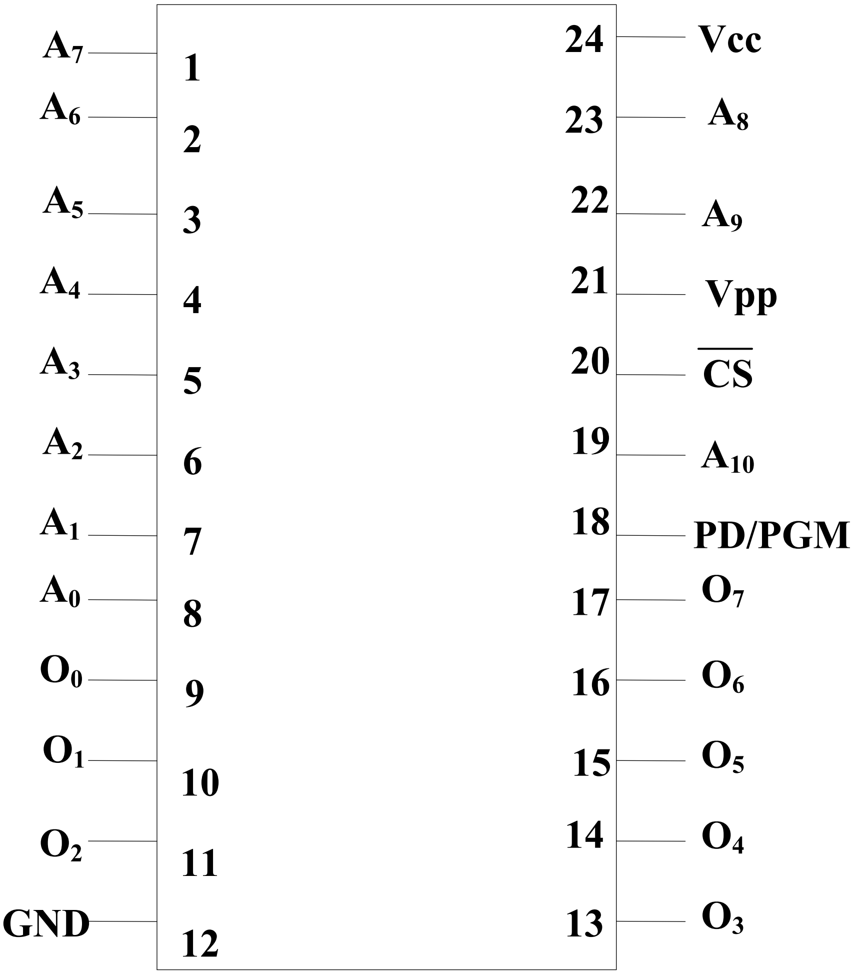
- We have four common types of memory:
- Read only memory (ROM)
- Flash memory (EEPROM)
- Static Random access memory (SARAM)
- Dynamic Random access memory (DRAM).
- Pin connections common to all memory devices are: The address input, data output or input/outputs, selection input and control input used to select a read or write operation.
- ***Address connections***: All memory devices have address inputs that select a memory location within the memory device. Address inputs are labeled from A0 to An.
- ***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 (CS), chip enable (CE) or simply select (S) input.



MEMORY COMPONENT ILLUSTRATING THE ADDRESS, DATA CONTROL CONNECTIONS

- RAM memory generally has at least one CS or S input and ROM at least one CE.
- If the CE, CS, 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 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.
- The control input most often found on the ROM is the output enable (OE) or gate (G), this allows data to flow out of the output data pins of the ROM.
- If OE and the selected input are both active, then the output is enable, if OE is inactive, the output is disabled at its high-impedance state.
- The OE connection enables and disables a set of three-state buffer located within the memory device and must be active to read data.
- A RAM memory device has either one or two control inputs. If there is one control input it is often called R/W.
- This pin selects a read operation or a write operation only if the device is selected by the selection input (CS).
- If the RAM has two control inputs, they are usually labeled WE or W and OE or G.
- (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 WE and OE are present, they must never be active at the same time.
- 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.
- 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..
- Each of these parts contains address pins, eight data connections, one or more chip selection inputs (CE) and an output enable pin (OE).
- This device contains **11** address inputs and **8** data outputs.
- If both the pin connection CE and 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.

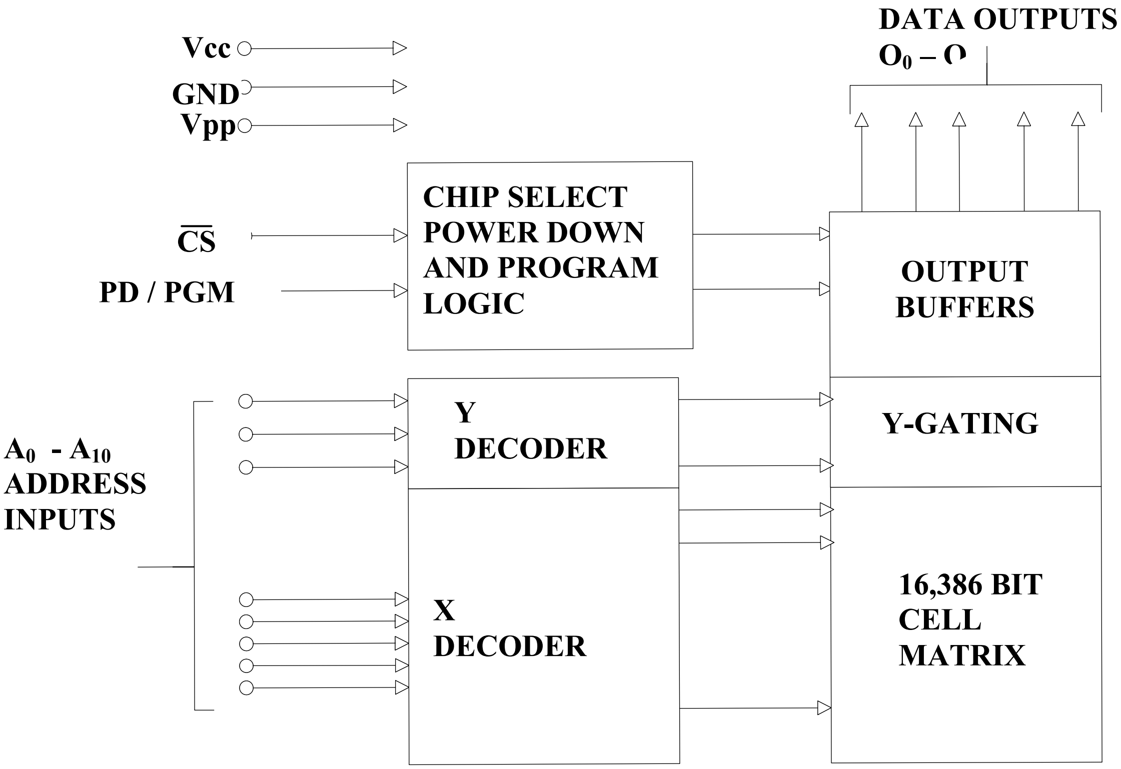
- To read data from the EPROM Vpp pin must be placed at a logic 1.



PIN CONFIGURATION OF 2716 EPROM

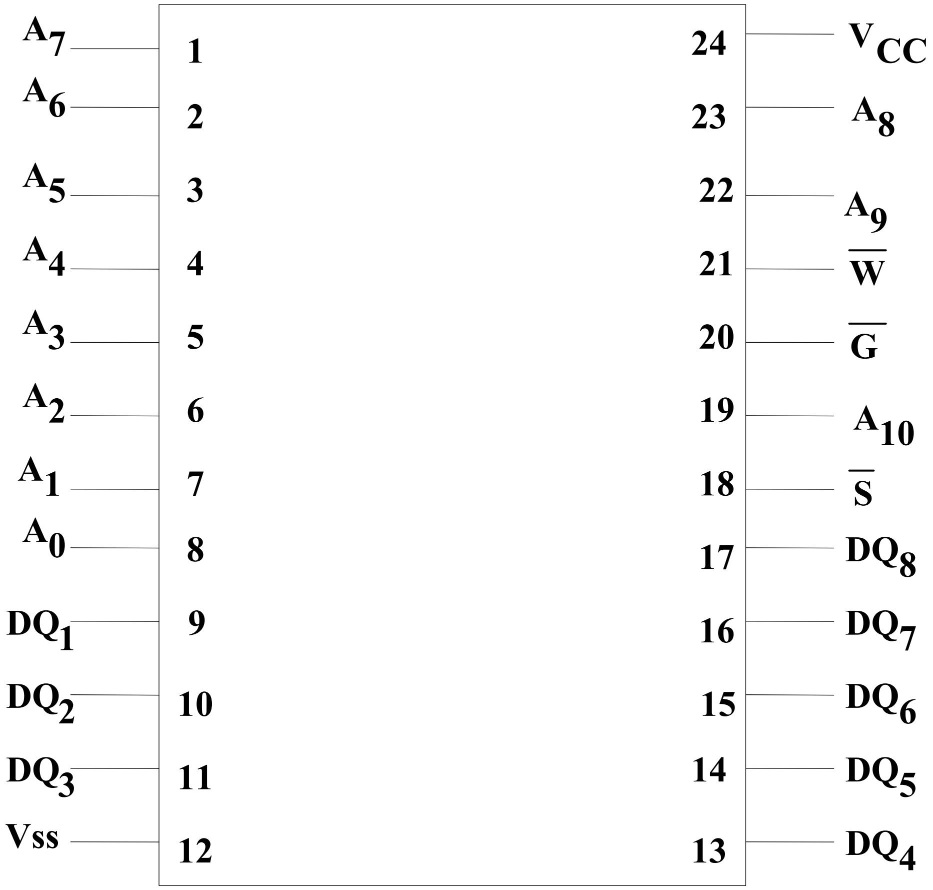
$A_0 – A_{10}$	ADDRESSES
PD/PGM	POWER DOWN PROGRAM
$\overline{C}$	CHIP
O_0-O_7	OUT

PIN



BLOCK DIAGRAM

- 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

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

Static RAM Interfacing

- 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.
- 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 = Log2 N.
- Thus if the microprocessor has n address lines, then it is able to address at the most N locations of memory, where 2n=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.

- The memory address depends upon the hardware circuit used for decoding the chip select (CS). The output of the decoding circuit is connected with the 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.
2. Connect available memory address lines of memory chip with those of the microprocessor and also connect the memory RD and 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, BHE and A0 are used for decoding the required chip select signals for the odd and even memory banks. The CS of memory is derived from the o/p of the decoding circuit.

- 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

- 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.
- 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***.
- 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

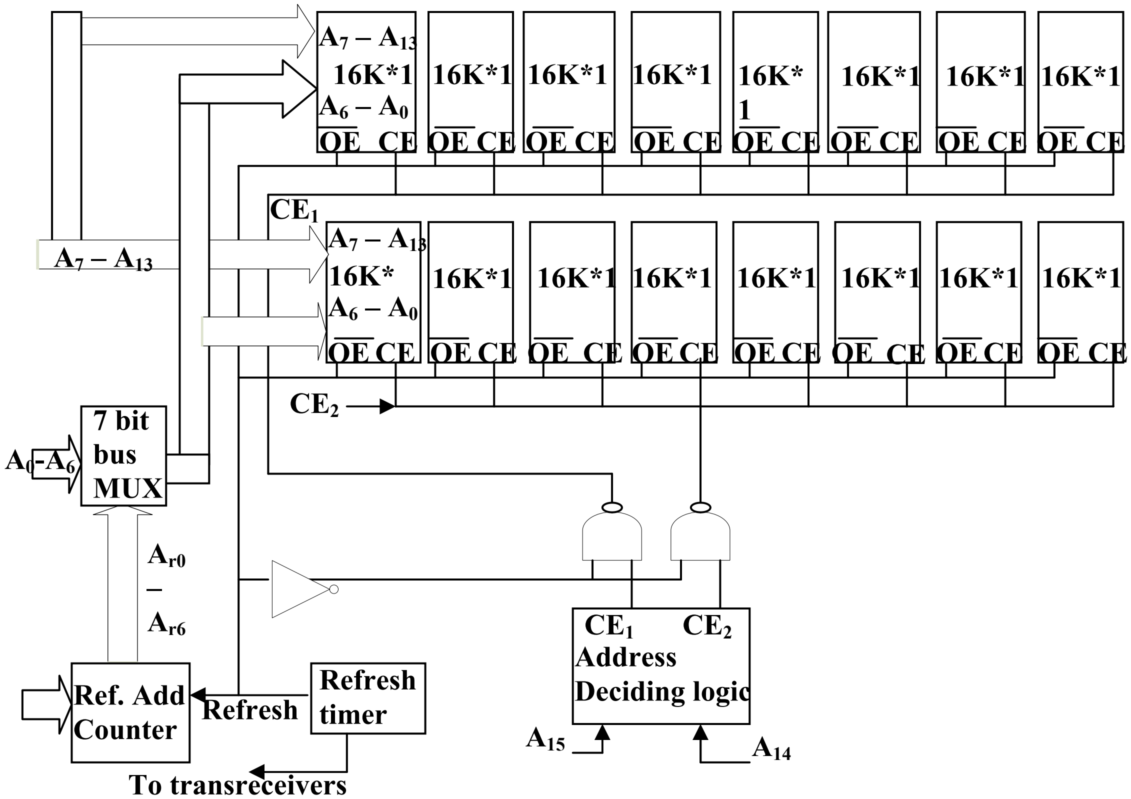
period all other operations related to the memory subsystem are suspended. Hence the refresh activity causes loss of time, resulting in reduce system performance.

- 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.
- 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.
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 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.
- 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.
- 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.

- 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) tr = (2 * 10 -3) / 64.
- Refresh Frequency fr = 64 / (2 * 10 -3) = 32 * 103 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.
- The 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, OE and 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 A0-A6 are multiplexed with the refresh counter output A10-A16.



Dynamic RAM Refreshing Logic

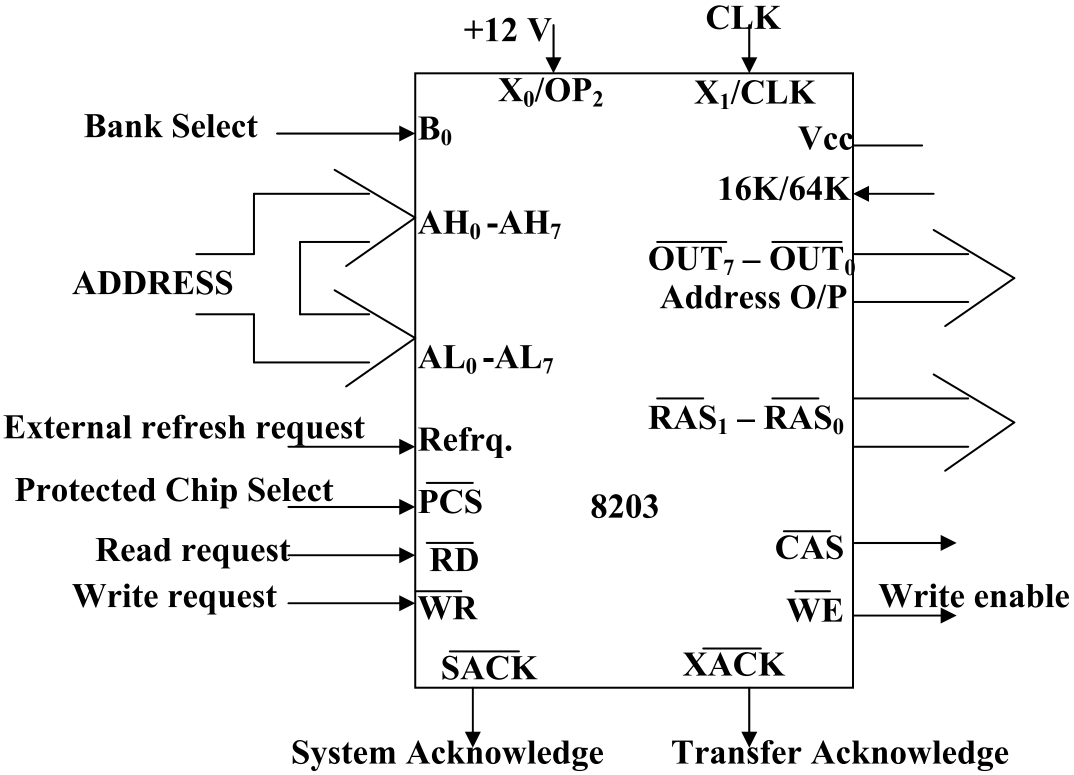


Fig : Dynamic RAM controller

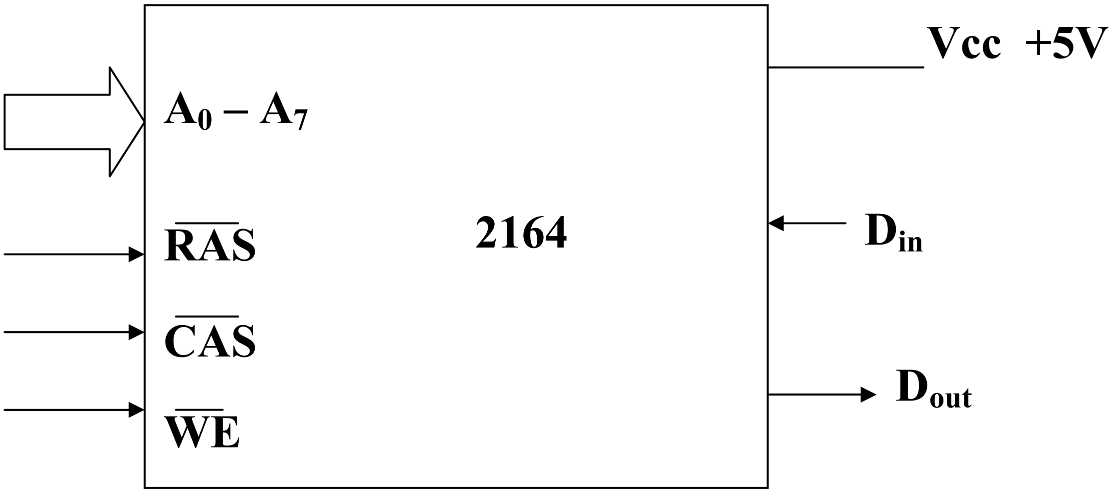


Fig : 1- bit Dynamic RAM

- The pin assignment for 2164 dynamic RAM is as in above fig.

- The RAS and CAS are row and column address strobes and are driven by the dynamic RAM controller outputs. A0 –A7 lines are the row or column address lines, driven by the OUT0 – OUT7 outputs of the controller. The WE pin indicates memory write cycles. The DIN and DOUT 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.
- 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 A1 to A16 on lines AL0-AL7 and AH0-AH7.
- The A0 lines is used to select the even or odd bank. The RD and WR signals decode whether the cycle is a memory read or memory write cycle and are accepted as inputs to 8203 from the microprocessor.
- The WE signal specifies the memory write cycle and is not output from 8203 that drives the WE input of dynamic RAM memory chip. The OUT0 – OUT7 set of eight pins is an 8-bit output bus that carries multiplexed row and column addresses are derived from the address lines A1-A16 accepted by the controller on its inputs AL0-AL7 and AH0-AH7.
- An external crystal may be applied between X0 and X1 pins, otherwise with the OP2 pin at +12V, a clock signal may be applied at pin CLK.
- The 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 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 SACK output signal marks the beginning of a memory access cycle.
- If a memory request is made during a memory refresh cycle, the 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).

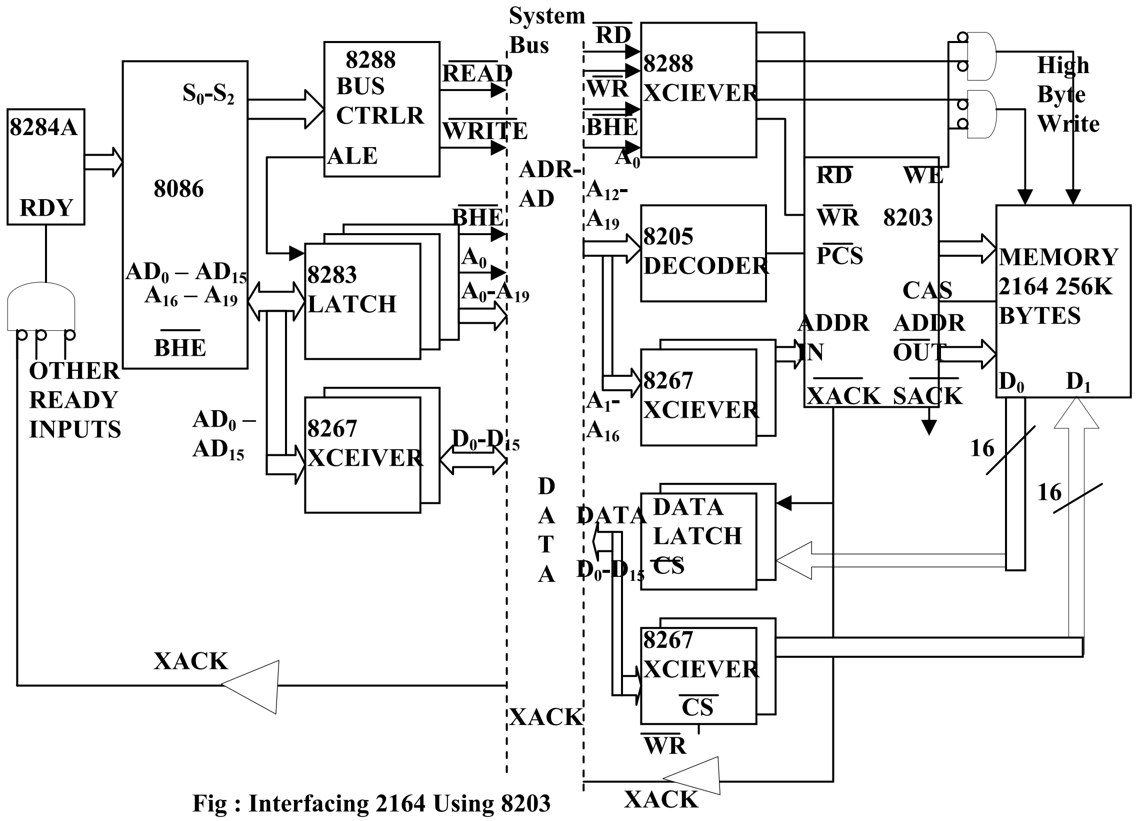


Fig : Interfacing 2164 Using 8203

- 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 AACK output of 8208 latches the A0 and BHE signals required for selecting the banks. If the latched bank select signal and the WE/PCLK output of 8208 both become low. It indicates a write operation to the respective bank.

PIO 8255

- 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.
- The port A lines are identified by symbols PA0-PA7 while the port C lines are identified as PC4-PC7. Similarly, Group B contains an 8-bit port B, containing lines PB0-PB7 and a 4-bit port C with lower bits PC0- PC3. 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).
- 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.
- RD, WR, A1, A0 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.
- 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 :
- **PA7-PA0**: 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.
- **PC7-PC4** : Upper nibble of port C lines. They may act as either output latches or input buffers lines.
- This port also can be used for generation of handshake lines in mode 1 or mode 2.
- **PC3-PC0** : These are the lower port C lines, other details are the same as PC7-PC4 lines.
- **PB0-PB7** : 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.
- **RD** : This is the input line driven by the microprocessor and should be low to indicate read operation to 8255.
- **WR** : This is an input line driven by the microprocessor. A low on this line indicates write operation.
- **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.
- **A1-A0** : These are the address input lines and are driven by the microprocessor. These lines A1-A0 with RD, WR and 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 A0 and A1 pins of 8255 are connected with A1 and A2 respectively.

- **D0-D7** : 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.

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	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

Block Diagram of 8255 (Architecture)

- 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.
- ***Read/Write control logic***: This unit accepts control signals (RD, 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) **CS** – Chipselect : A low on this PIN enables the communication between CPU and 8255.

b) **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.

c) **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) **A0** and **A1** (Address pins): These pins in conjunction with RD and 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.
- Group A - PA and PCU (PC7 –PC4)
- Group B - PCL (PC3 – PC0)
- 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.

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.

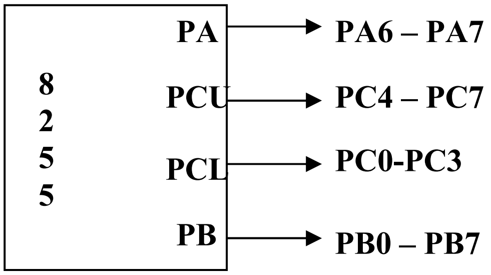
Modes of Operation of 8255

- 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 (PC0-PC7) 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.
- ***BSR Mode***: In this mode any of the 8-bits of port C can be set or reset depending on D0 of the control word. The bit to be set or reset is selected by bit select flags D3, D2 and D1 of the CWR as given in table.
- **I/O Modes** :

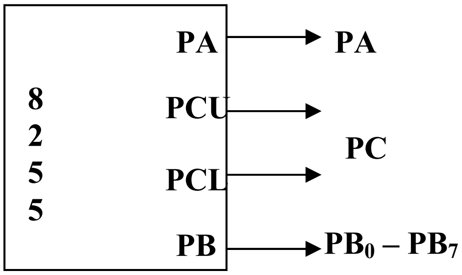
a) ***Mode 0 (Basic I/O mode):*** This mode is also called as basic input/output mode. This mode provides simple input and output capabilities using each of the three ports. Data can be simply read from and written to the input and output ports respectively, after appropriate initialisation.

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



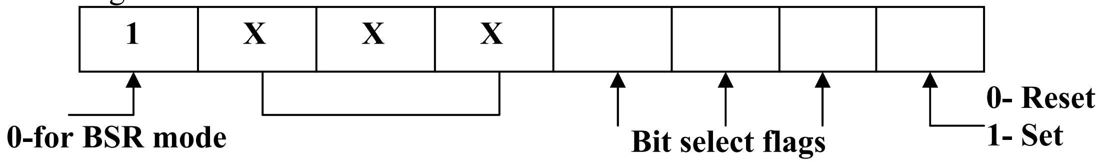
Port A and Port C acting as O/P. Port B acting as I/P

Mode 0

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

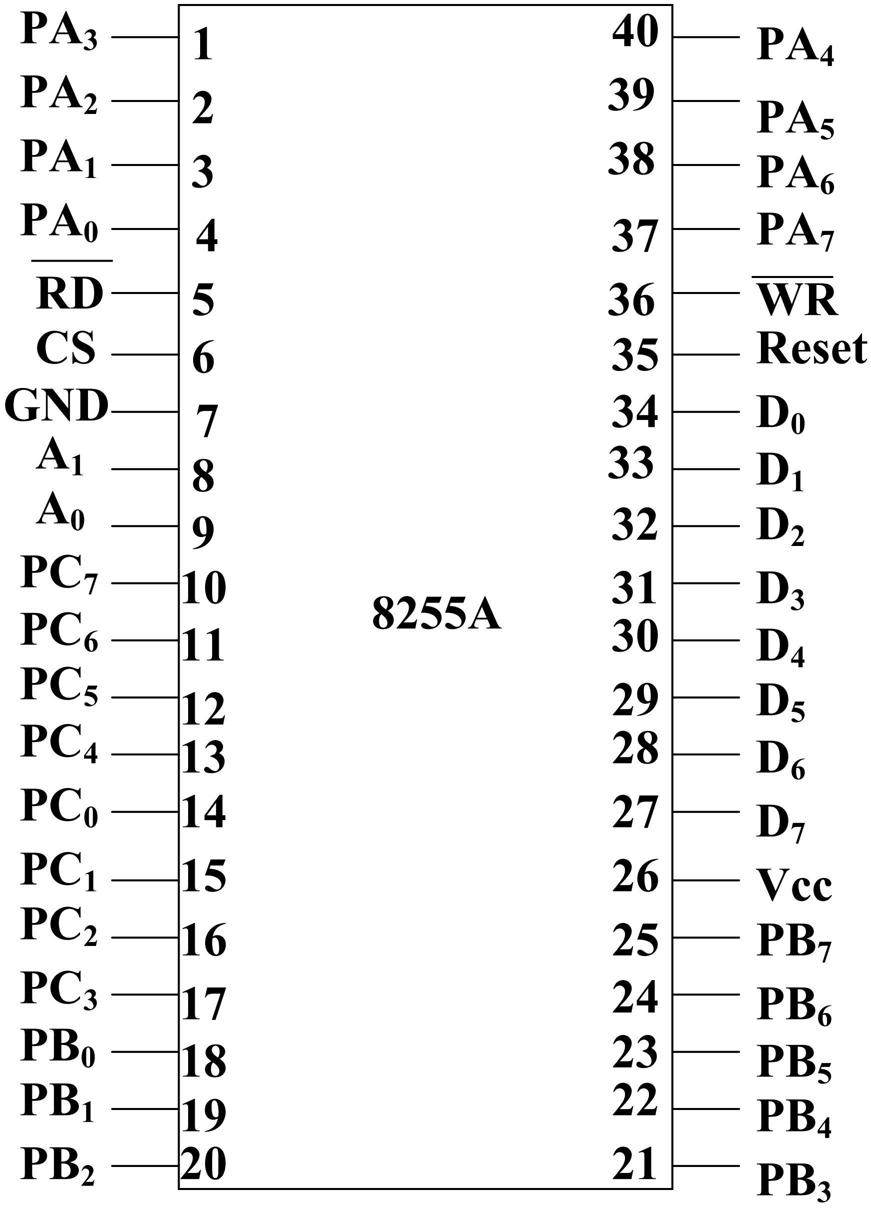
1. Two 8-bit ports (port A and port B)and two 4-bit ports (port C upper and lower) are available. The two 4-bit ports can be combinedly used as a third 8-bit port.
2. Any port can be used as an input or output port.
3. Output ports are latched. Input ports are not latched.
4. A maximum of four ports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

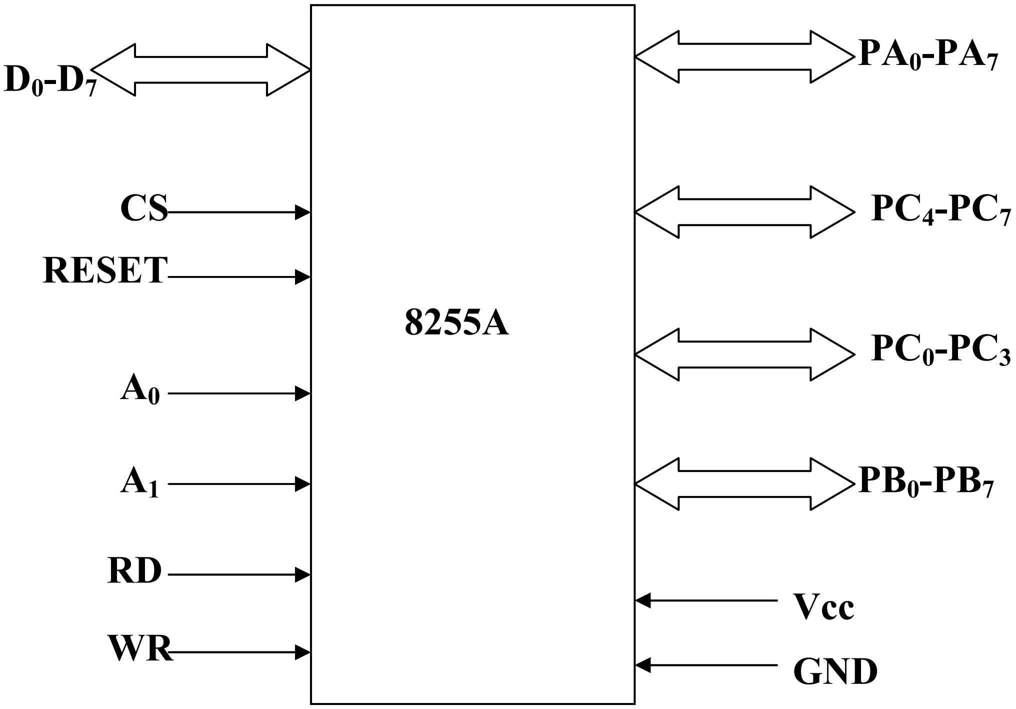


D_3, D_2, D_1 are from 000 to 111 for bits PC_0 TO PC_7

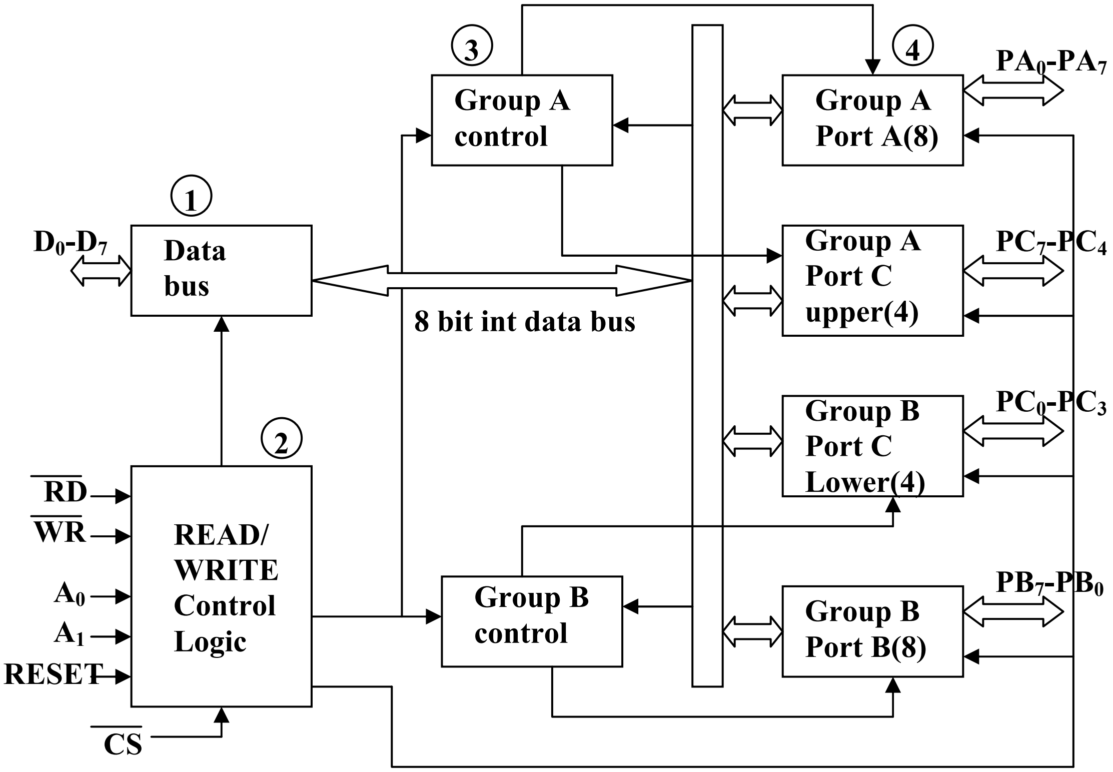
I/O Mode Control Word Register Format and
BSR Mode Control Word Register Format



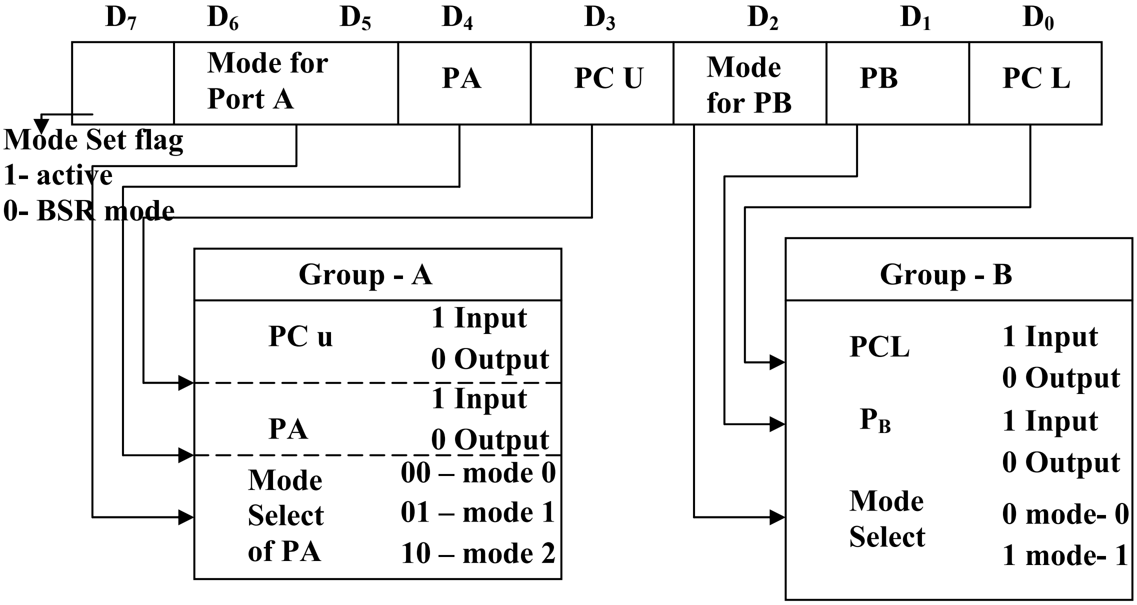
8255A Pin Configuration



Signals of 8255



Block Diagram of 8255



Control Word Format of 8255

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 PC0-PC2, provide strobe or handshake lines for port B. This group which includes port B and PC0-PC2 is called as group B for Strobed data input/output. Port C lines PC3-PC5 provide strobe lines for port A. This group including port A and PC3-PC5 from group A. Thus port C is utilized for generating handshake signals. The salient features of mode 1 are listed as follows:

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, PC0-PC2 are used to generate control signals for port B and PC3-PC5 are used to generate control signals for port A. the lines PC6, PC7 may be used as independent data lines.

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

Input control signal definitions (mode 1):

- **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 STB and is reset by the rising edge of RD input.
- **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 PC4(INTEA) or PC2(INTEB) 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.

Output control signal definitions (mode 1) :

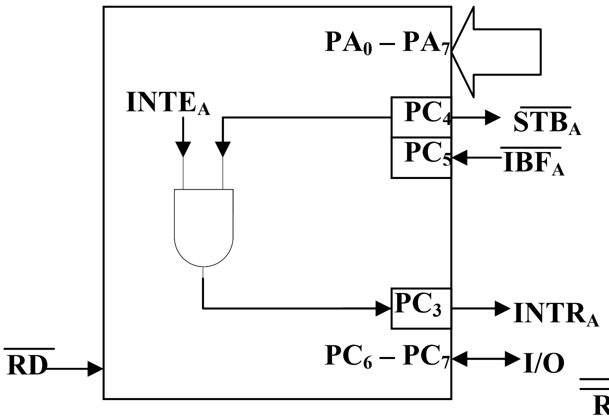
- **OBF** (Output buffer full) – This status signal, whenever falls to low, indicates that CPU has written data to the specified output port. The OBF flip-flop will be set by a rising edge of WR signal and reset by a low going edge at the ACK input.
- **ACK** (Acknowledge input) – ACK signal acts as an acknowledgement to be given by an output device. 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.
- **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 PC6 and PC2 respectively.

Input control signal definitions in Mode 1

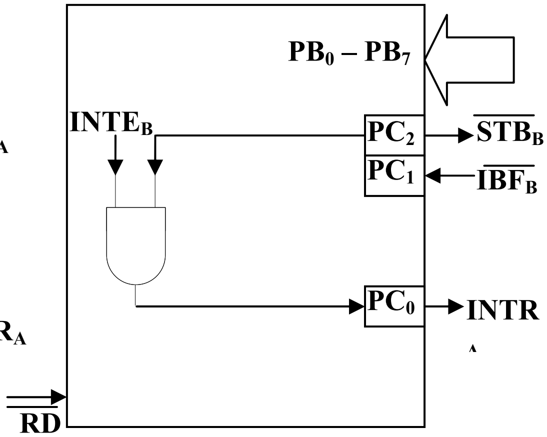
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	1	0	1/0	X	X	X

D_3: 1 - Input
0 - Output
For $PC_6 – PC_7$

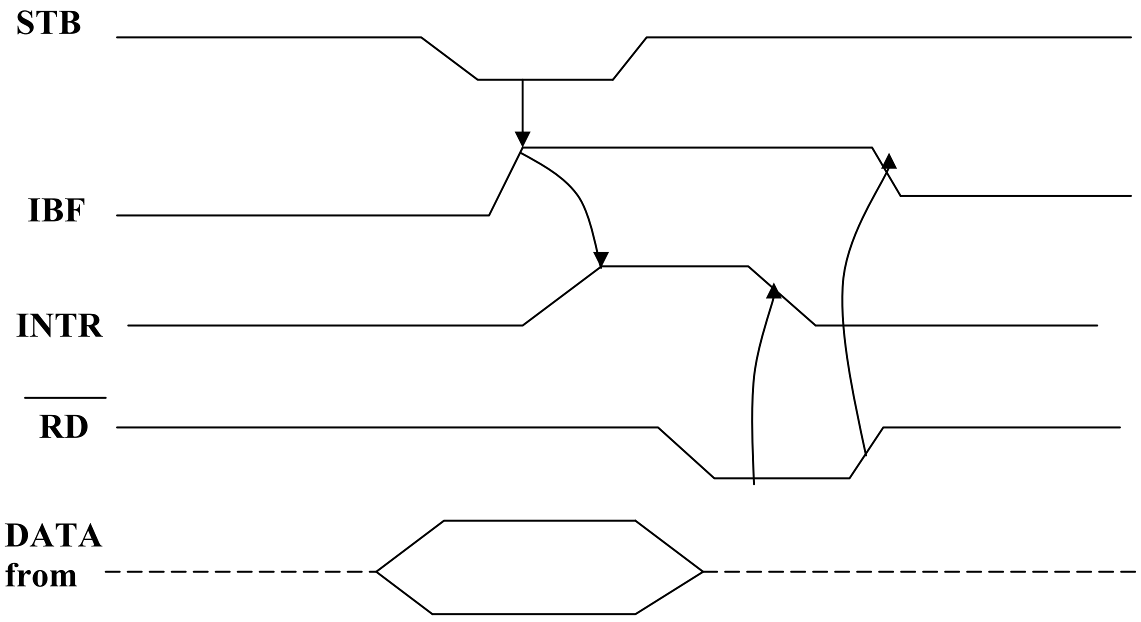
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	X	X	X	X	1	1	X



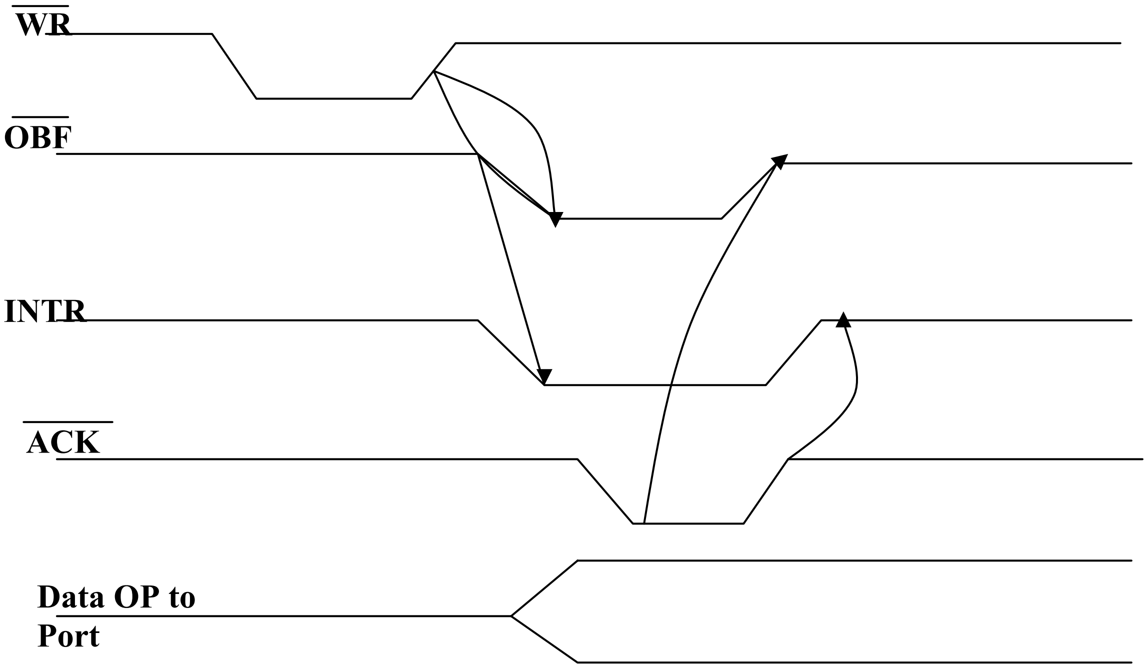
Mode 1 Control Word Group A I/P



Mode 1 Control Word Group B I/P



Mode 1 Strobed Input Data Transfer



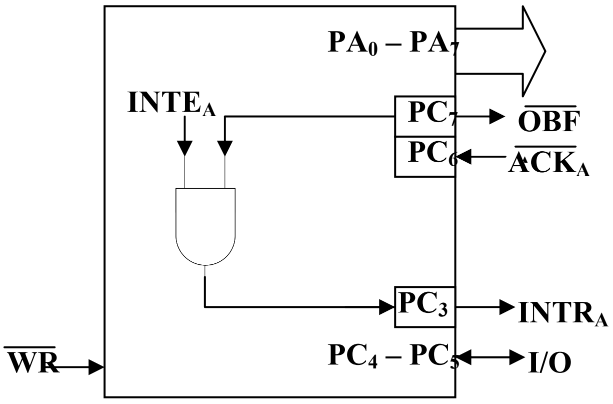
Mode 1 Strobed Data Output

Output control signal definitions Mode 1

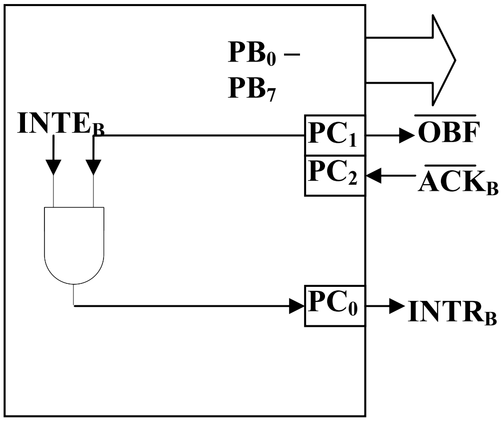
1	0	1	0	1/0	X	X	X
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

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



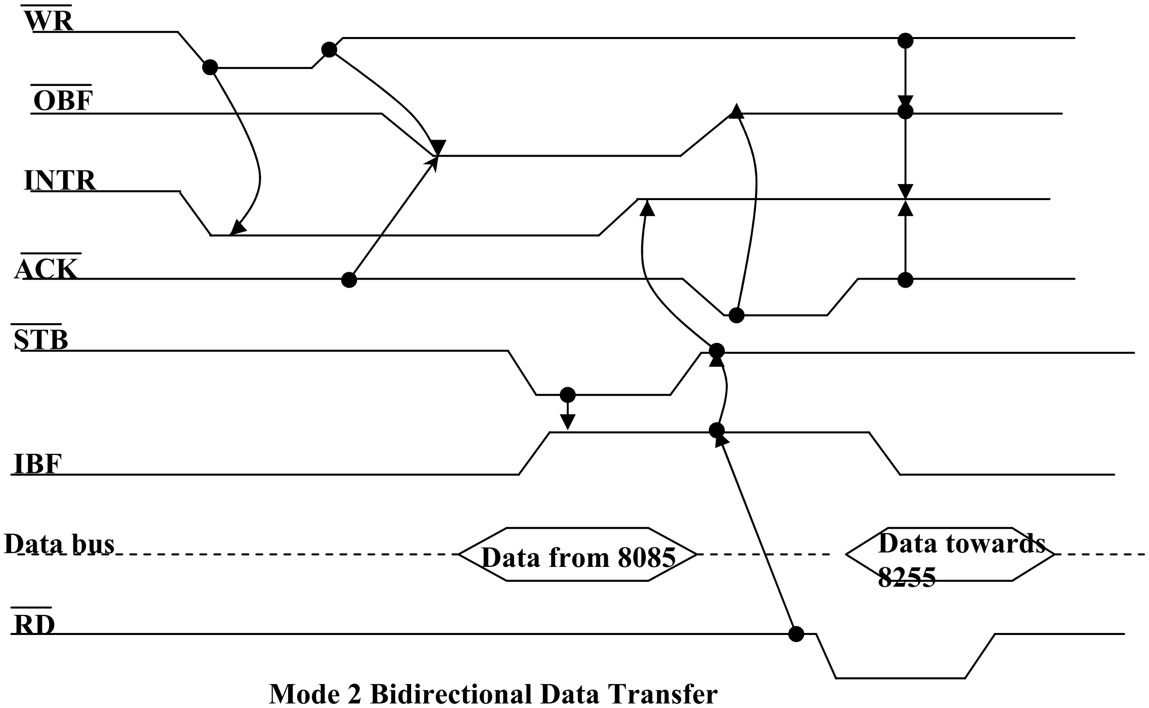
Mode 1 Control Word Group B

- **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.
- 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.(PC2 – PC0)
4. Inputs and outputs are both latched.
5. The 5-bit control port C (PC3-PC7) is used for generating / accepting handshake signals for the 8-bit data transfer on port A.

- ***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.

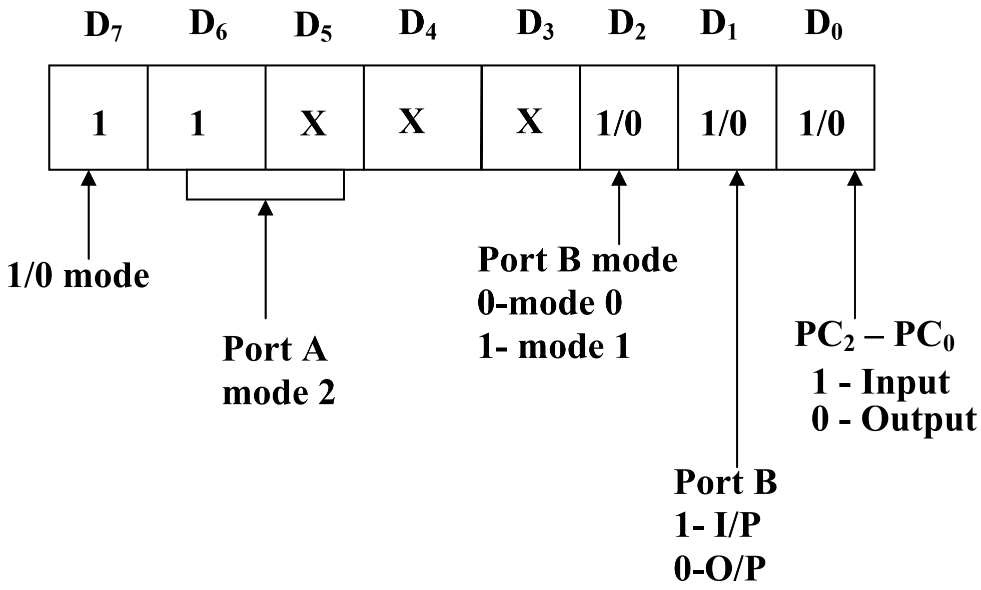
- ***Control Signals for Output operations***:
- **OBF** (Output buffer full) – This signal, when falls to low level, indicates that the CPU has written data to port A.
- **ACK** (Acknowledge) This control input, when falls to logic low level, acknowledges that the previous data byte is received by the 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 OBF) This can be controlled by bit set/reset mode with PC6.
- ***Control signals for input operations :***
- **STB** (Strobe input) A low on this line is used to strobe in the data into the input latches of 8255.
- **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: WR must occur before ACK and STB must be activated before RD.



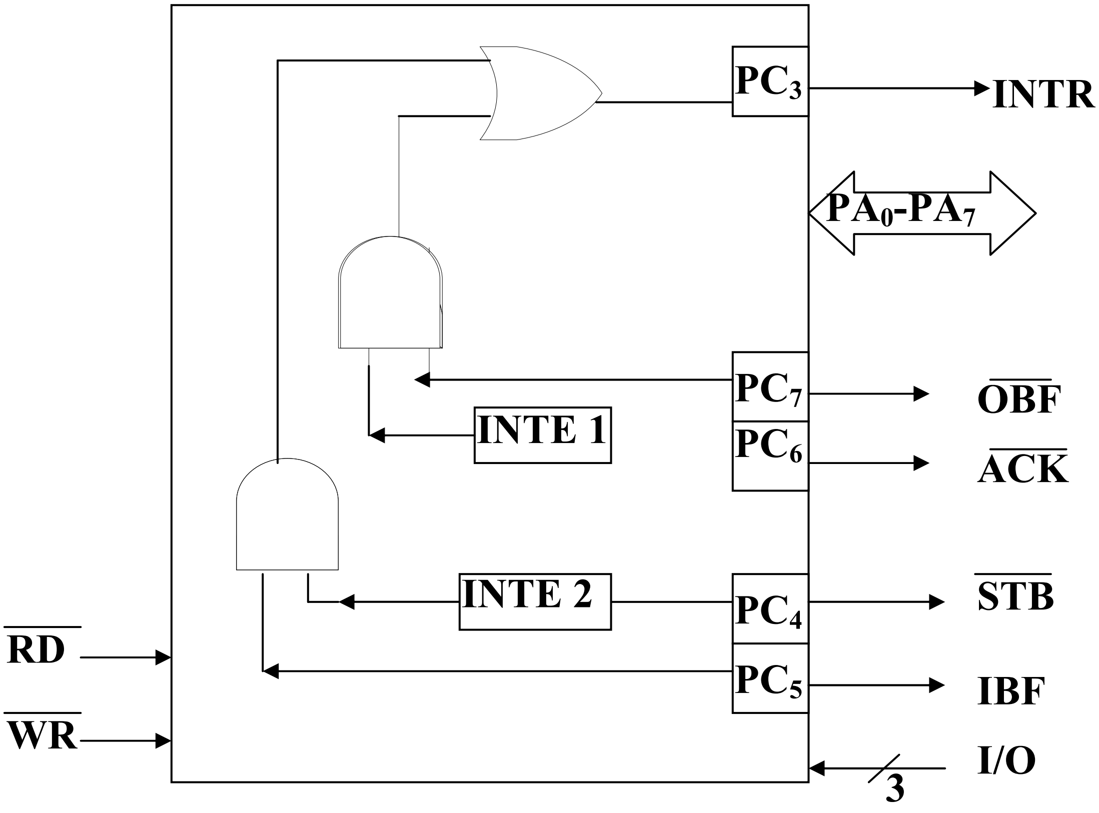
Mode 2 Bidirectional Data Transfer

- 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



Mode 2 pins

8254

- Compatible with All Intel and Most other Microprocessors
- Handles Inputs from DC to 10 MHz
- 8 MHz 8254
- 10 MHz 8254-2
- Status Read-Back Command
- Six Programmable Counter Modes
- Three Independent 16-Bit Counters
- Binary or BCD Counting
- Single a 5V Supply
- Standard Temperature Range
- The Intel 8254 is a counter/timer device designed to solve the common timing control problems in microcomputer system design.
- It provides three independent 16-bit counters, each capable of handling clock inputs up to 10 MHz.

- All modes are software programmable. The 8254 is a superset of the 8253.
- The 8254 uses HMOS technology and comes in a 24-pin plastic or CERDIP package.

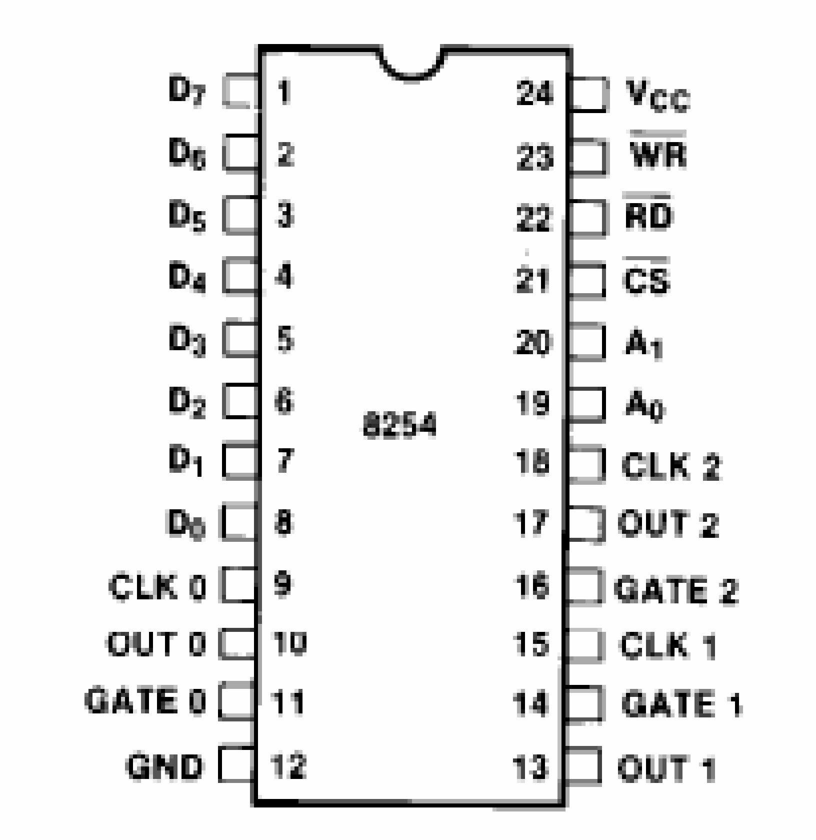


Figure 1. Pin Configuration

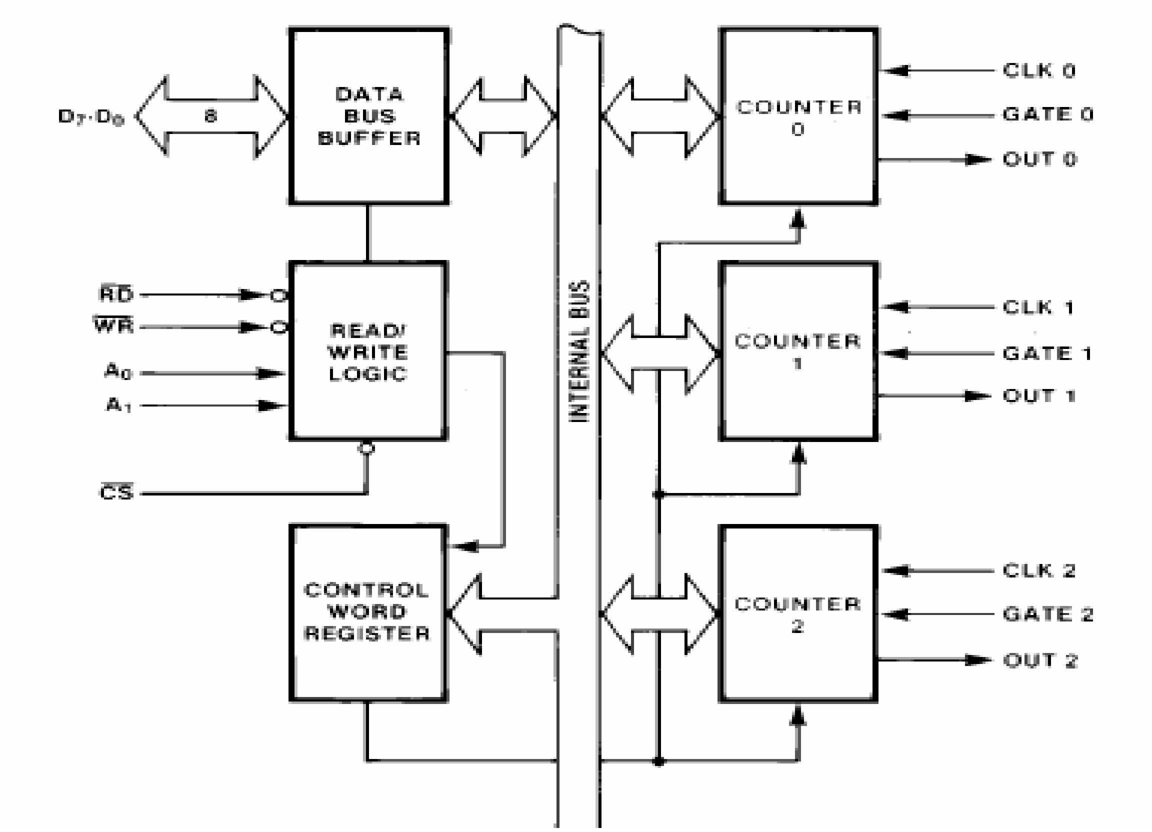


Figure 2. 8254 Block

Pin Description

Symbol	Pin No.	Type	Name and Function
D7-D0	1 - 8	I/O	DATA: Bi-directional three state data bus lines, connected to system data bus.
CLK 0	9	I	CLOCK 0: Clock input of Counter 0.
OUT 0	10	O	OUTPUT 0: Output of Counter 0.
GATE 0	11	I	GATE 0: Gate input of Counter 0.
GND	12		GROUND: Power supply connection.
VCC	24		POWER: A 5V power supply connection.
$\overline{\text{WR}}$	23	I	WRITE CONTROL: This input is low during CPU write operations.
RD	22	I	READ CONTROL: This input is low during CPU read operations.

<table>
<tr><td>CS</td><td>21</td><td>I</td><td colspan="3">CHIP SELECT: A low on this input enables the 8254 to respond to RD and WR signals. RD and WR are ignored otherwise.</td></tr>
<tr><td rowspan="3">A1, A0</td><td rowspan="3">20 – 9</td><td rowspan="3">I</td><td colspan="3">ADDRESS: Used to select one of the three Counters or the Control Word Register for read or write operations. Normally connected to the system address bus.</td></tr>
<tr><td>A1</td><td>A0</td><td>Selects</td></tr>
<tr><td>0
0
1
1</td><td>0
1
0
1</td><td>Counter 0
Counter 1
Counter 2
Control Word Register</td></tr>
<tr><td>CLK 2</td><td>18</td><td>I</td><td colspan="3">CLOCK 2: Clock input of Counter 2.</td></tr>
<tr><td>OUT 2</td><td>17</td><td>O</td><td colspan="3">OUT 2: Output of Counter 2.</td></tr>
</table>

GATE 2	16	I	GATE 2: Gate input of Counter 2.
CLK 1	15	I	CLOCK 1: Clock input of Counter 1.
GATE 1	14	I	GATE 1: Gate input of Counter 1.
OUT 1	OUT 1	O	OUT 1: Output of Counter 1.

Functional Description

- The 8254 is a programmable interval timer/counter designed for use with Intel microcomputer systems.
- It is a general purpose, multi-timing element that can be treated as an array of I/O ports in the system software.
- The 8254 solves one of the most common problems in any microcomputer system, the generation of accurate time delays under software control. Instead of setting up timing loops in software, the programmer configures the 8254 to match his requirements and programs one of the counters for the desired delay.

- After the desired delay, the 8254 will interrupt the CPU. Software overhead is minimal and variable length delays can easily be accommodated.
- Some of the other counter/timer functions common to microcomputers which can be implemented with the 8254 are:
- Real time clock
- Event-counter
- Digital one-shot
- Programmable rate generator
- Square wave generator
- Binary rate multiplier
- Complex waveform generator
- Complex motor controller

Block Diagram

- **DATA BUS BUFFER**: This 3-state, bi-directional, 8-bit buffer is used to interface the 8254 to the system bus, see the figure : Block Diagram Showing Data Bus Buffer and Read/Write Logic Functions.
- **READ/WRITE LOGIC** : The Read/Write Logic accepts inputs from the system bus and generates control signals for the other functional blocks of the 8254. A1 and A0 select one of the three counters or the Control Word Register to be read from/written into.
- A ``low" on the RD input tells the 8254 that the CPU is reading one of the counters.

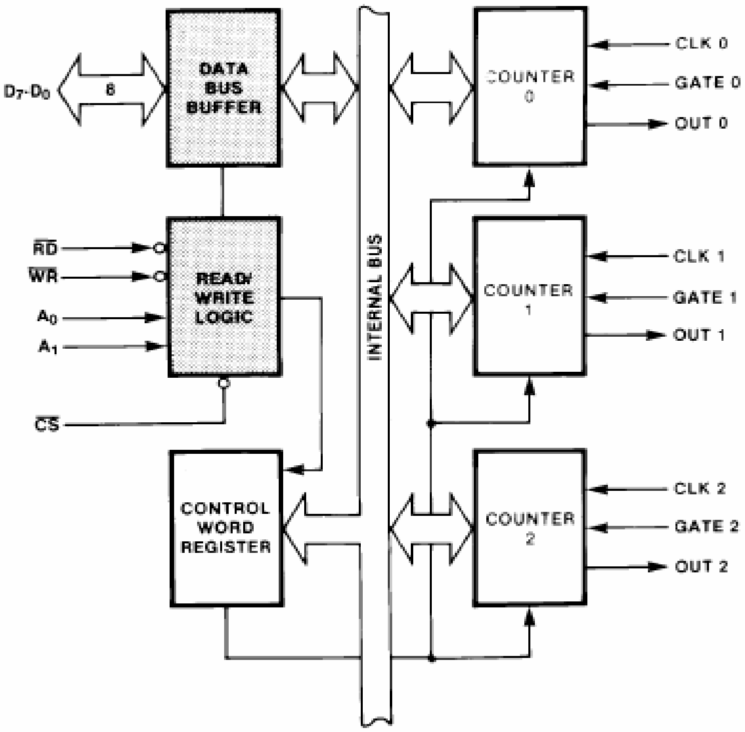


Figure 3. Block Diagram Showing Data Bus Buffer and Read/Write Logic Functions

- A ``low" on the WR input tells the 8254 that the CPU is writing either a Control Word or an initial count. Both RD and WR are qualified by CS; RD and WR are ignored unless the 8254 has been selected by holding CS low.
- **CONTROL WORD REGISTER :**The Control Word Register (see Figure 4) is selected by the Read/Write Logic when A1,A0 = 11. If the CPU then does a write operation to the 8254, the data is stored in the Control Word Register and is interpreted as a Control Word used to define the operation of the Counters.

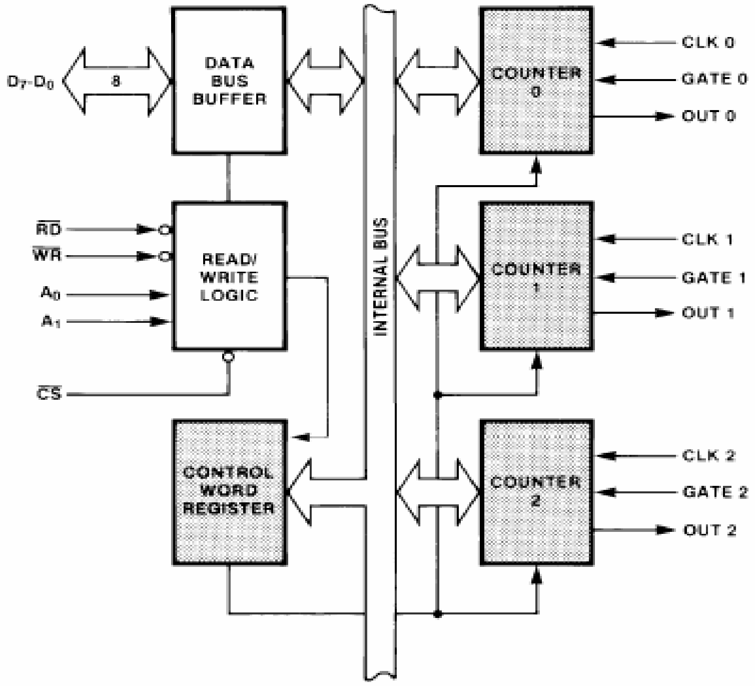


Figure 4. Block Diagram Showing Control Word Register and Counter Functions

- The Control Word Register can only be written to; status information is available with the Read-Back Command.
- **COUNTER 0, COUNTER 1, COUNTER 2** :These three functional blocks are identical in operation, so only a single Counter will be described. The internal block diagram of a single counter is shown in Figure 5.
- The Counters are fully independent. Each Counter may operate in a different Mode.
- The Control Word Register is shown in the figure; it is not part of the Counter itself, but its contents determine how the Counter operates.
- The status register, shown in Figure 5, when latched, contains the current contents of the Control Word Register and status of the output and null count flag. (See detailed explanation of the Read-Back command.)
- The actual counter is labelled CE (for ``Counting Element"). It is a 16-bit presettable synchronous down counter. OLM and OLL are two 8-bit latches. OL stands for ``Output Latch"; the subscripts M and L stand for ``Most significant byte" and ``Least significant byte' respectively.

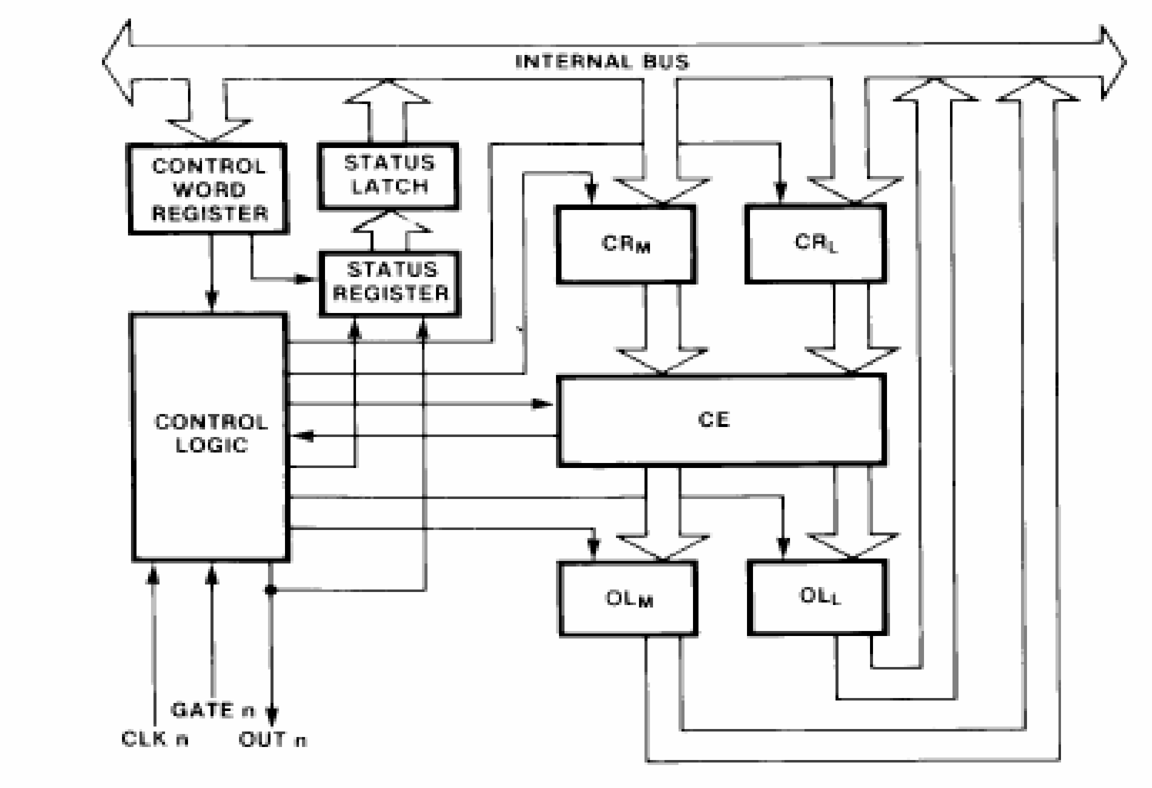


Figure 5. Internal Block Diagram of a Counter

- Both are normally referred to as one unit and called just OL. These latches normally ``follow'‘ the CE, but if a suitable Counter Latch Command is sent to the 8254, the latches ``latch" the present count until read by the CPU and then return to ``following" the CE.
- One latch at a time is enabled by the counter's Control Logic to drive the internal bus. This is how the 16-bit Counter communicates over the 8-bit internal bus. Note that the CE itself cannot be read; whenever you read the count, it is the OL that is being read.
- Similarly, there are two 8-bit registers called CRM and CRL (for ``Count Register"). Both are normally referred to as one unit and called just CR.
- When a new count is written to the Counter, the count is stored in the CR and later transferred to the CE. The Control Logic allows one register at a time to be loaded from the internal bus. Both bytes are transferred to the CE simultaneously.
- CRM and CRL are cleared when the Counter is programmed. In this way, if the Counter has been programmed for one byte counts (either most significant byte only or least significant byte only) the other byte will be zero.
- Note that the CE cannot be written into, whenever a count is written, it is written into the CR.
- The Control Logic is also shown in the diagram.
- CLK n, GATE n, and OUT n are all connected to the outside world through the Control Logic.

- **8254 SYSTEM INTERFACE :**The 8254 is a component of the Intel Microcomputer Systems and interfaces in the same manner as all other peripherals of the family.
- It is treated by the system's software as an array of peripheral I/O ports; three are counters and the fourth is a control register for MODE programming.
- Basically, the select inputs A0,A1 connect to the A0,A1 address bus signals of the CPU. The CS can be derived directly from the address bus using a linear select method. Or it can be connected to the output of a decoder, such as an Intel 8205 for larger systems.
- **Programming the 8254 :**Counters are programmed by writing a Control Word and then an initial count.
- The Control Words are written into the Control Word Register, which is selected when A1,A0 = 11. The Control Word itself specifies which Counter is being programmed.

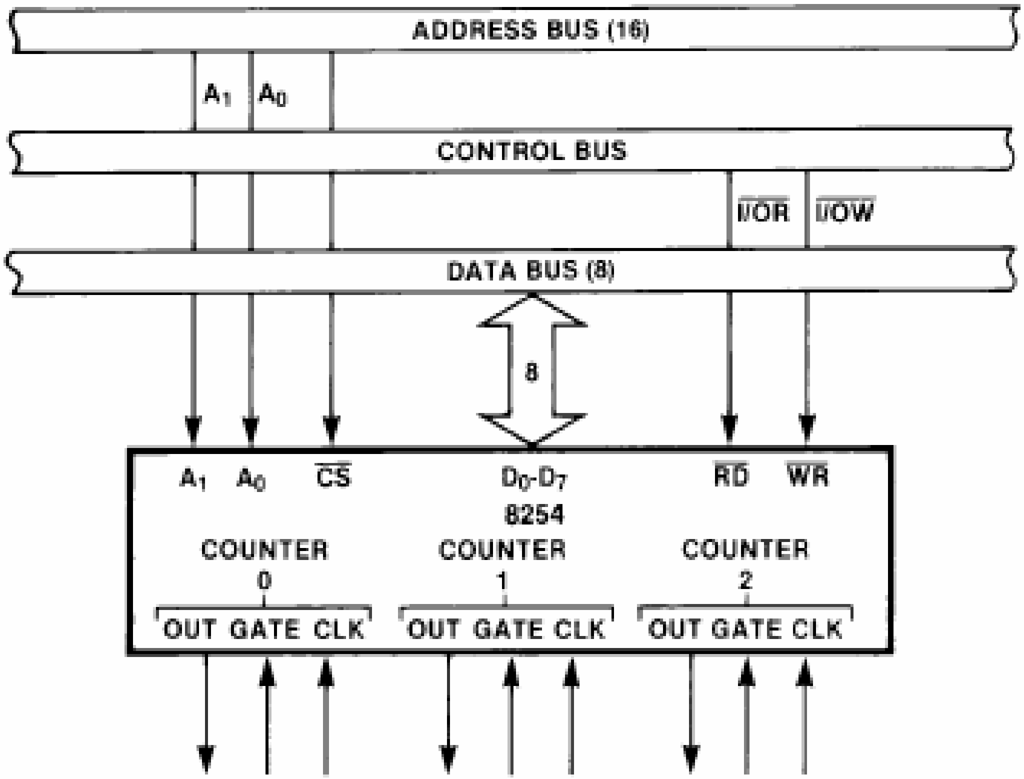


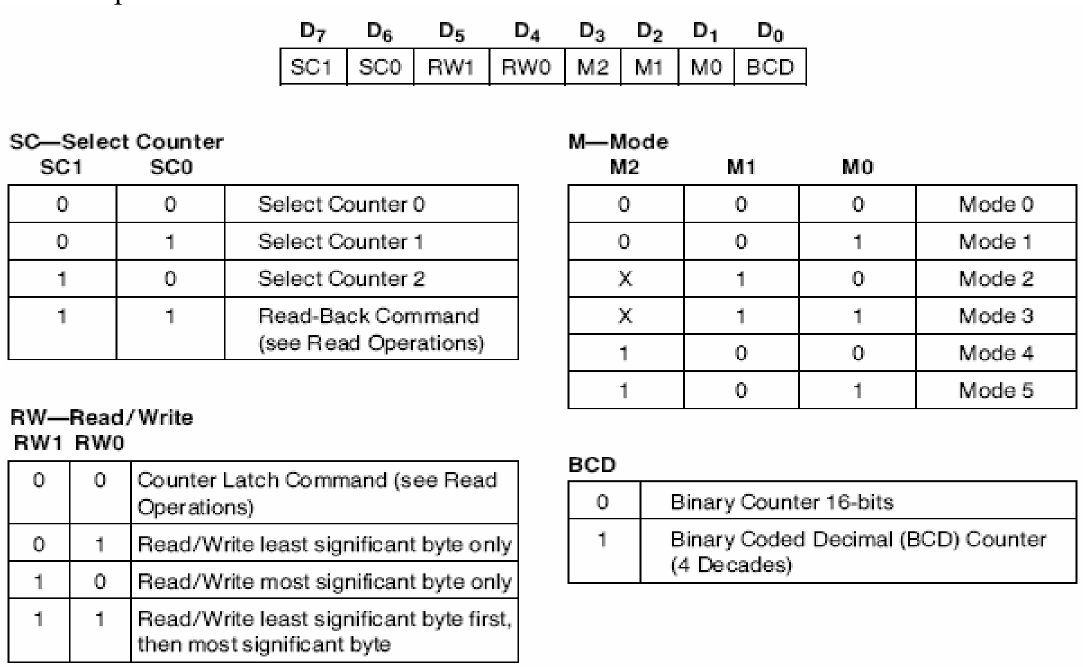
Figure 6. 8254 System Interface

- **Control Word Format:** A1,A0 = 11, CS = 0, RD = 1, WR = 0.

- By contrast, initial counts are written into the Counters, not the Control Word Register. The A1,A0 inputs are used to select the Counter to be written into. The format of the initial count is determined by the Control Word used.
- **Write Operations**: The programming procedure for the 8254 is very flexible. Only two conventions need to be remembered:

1) For each Counter, the Control Word must be written before the initial count is written.
2) The initial count must follow the count format specified in the Control Word (least significant byte only, most significant byte only, or least significant byte and then most significant byte).

- Since the Control Word Register and the three Counters have separate addresses (selected by the A1,A0 inputs), and each Control Word specifies the Counter it applies to (SC0,SC1 bits), no special instruction sequence is required.
- Any programming sequence that follows the conventions in Figure 7 is acceptable.



D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

SC—Select Counter

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Read-Back Command (see Read Operations)

M—Mode

M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

RW—Read/Write

RW1	RW0	
0	0	Counter Latch Command (see Read Operations)
0	1	Read/Write least significant byte only
1	0	Read/Write most significant byte only
1	1	Read/Write least significant byte first, then most significant byte

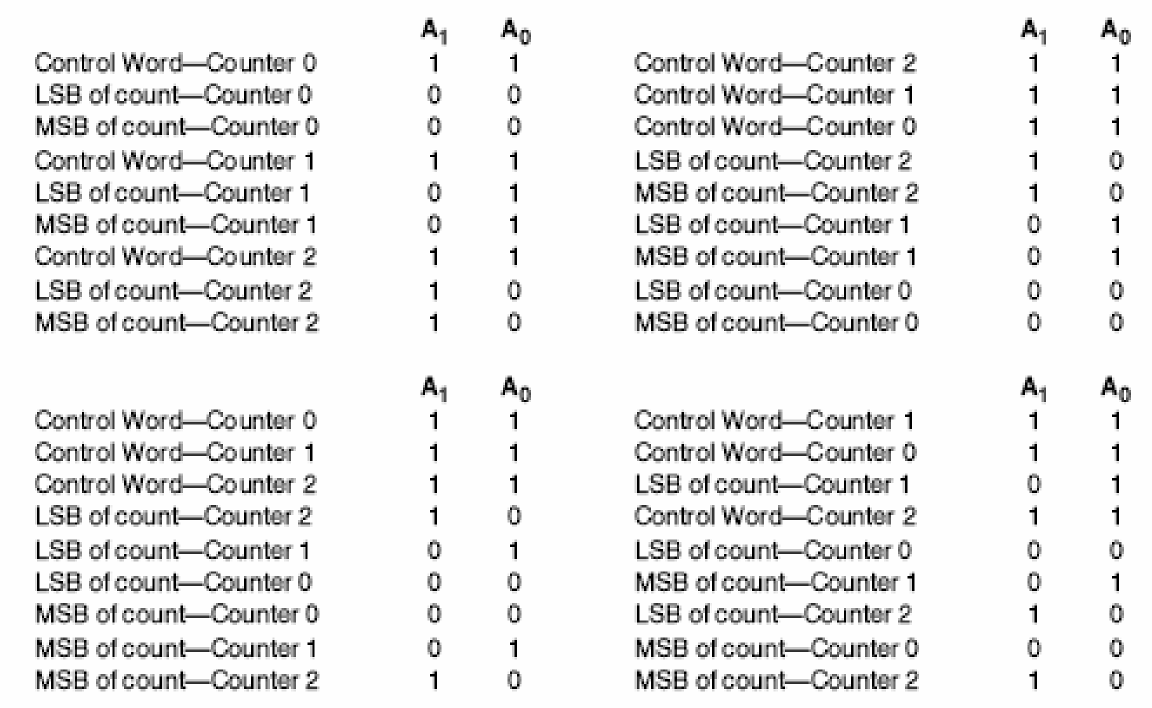
BCD

0	Binary Counter 16-bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

NOTE: Don't care bits (X) should be 0 to insure compatibility with future Intel products.

Figure 7. Control Word Format

- A new initial count may be written to a Counter at any time without affecting the Counter's programmed Mode in any way. Counting will be affected as described in the Mode definitions. The new count must follow the programmed count format.
- If a Counter is programmed to read/write two-byte counts, the following precaution applies: A program must not transfer control between writing the first and second byte to another routine which also writes into that same Counter. Otherwise, the Counter will be loaded with an incorrect count.



	A_1	A_0
Control Word—Counter 0	1	1
LSB of count—Counter 0	0	0
MSB of count—Counter 0	0	0
Control Word—Counter 1	1	1
LSB of count—Counter 1	0	1
MSB of count—Counter 1	0	1
Control Word—Counter 2	1	1
LSB of count—Counter 2	1	0
MSB of count—Counter 2	1	0

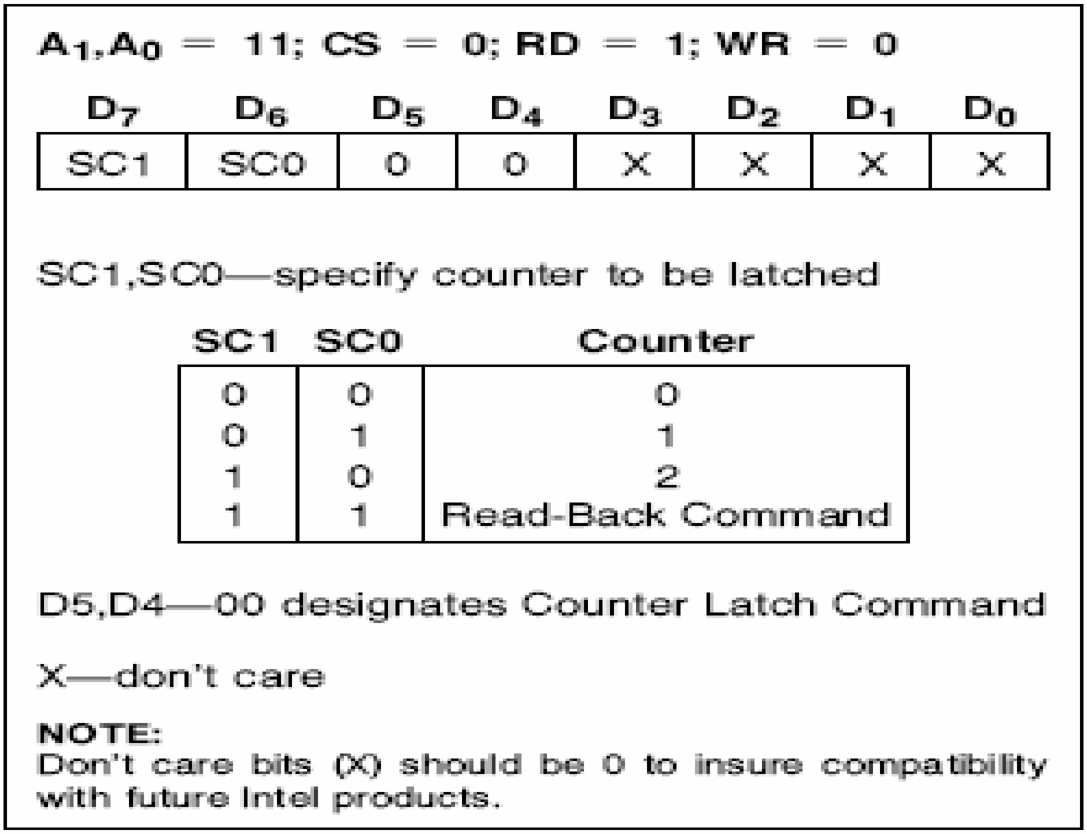
	A_1	A_0
Control Word—Counter 2	1	1
Control Word—Counter 1	1	1
Control Word—Counter 0	1	1
LSB of count—Counter 2	1	0
MSB of count—Counter 2	1	0
LSB of count—Counter 1	0	1
MSB of count—Counter 1	0	1
LSB of count—Counter 0	0	0
MSB of count—Counter 0	0	0

	A_1	A_0
Control Word—Counter 0	1	1
Control Word—Counter 1	1	1
Control Word—Counter 2	1	1
LSB of count—Counter 2	1	0
LSB of count—Counter 1	0	1
LSB of count—Counter 0	0	0
MSB of count—Counter 0	0	0
MSB of count—Counter 1	0	1
MSB of count—Counter 2	1	0

	A_1	A_0
Control Word—Counter 1	1	1
Control Word—Counter 0	1	1
LSB of count—Counter 1	0	1
Control Word—Counter 2	1	1
LSB of count—Counter 0	0	0
MSB of count—Counter 1	0	1
LSB of count—Counter 2	1	0
MSB of count—Counter 0	0	0
MSB of count—Counter 2	1	0

Figure 8. A Few Possible Programming Sequences

- **Read Operations**: It is often desirable to read the value of a Counter without disturbing the count in progress. This is easily done in the 8254.
- There are three possible methods for reading the counters: a simple read operation, the Counter Latch Command, and the Read-Back Command.
- Each is explained below. The first method is to perform a simple read operation. To read the Counter, which is selected with the A1, A0 inputs, the CLK input of the selected Counter must be inhibited by using either the GATE input or external logic.
- Otherwise, the count may be in the process of changing when it is read, giving an undefined result.
- **COUNTER LATCH COMMAND**: The second method uses the ``Counter Latch Command".
- Like a Control Word, this command is written to the Control Word Register, which is selected when A1,A0 = 11. Also like a Control Word, the SC0, SC1 bits select one of the three Counters, but two other bits, D5 and D4, distinguish this command from a Control Word.
- The selected Counter's output latch (OL) latches the count at the time the Counter Latch Command is received. This count is held in the latch until it is read by the CPU (or until the Counter is reprogrammed).



A_1,A_0 = 11; CS = 0; RD = 1; WR = 0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	0	0	X	X	X	X

SC1,SC0—specify counter to be latched

SC1	SC0	Counter
0	0	0
0	1	1
1	0	2
1	1	Read-Back Command

D5,D4—00 designates Counter Latch Command

X—don't care

NOTE:
Don't care bits (X) should be 0 to insure compatibility with future Intel products.

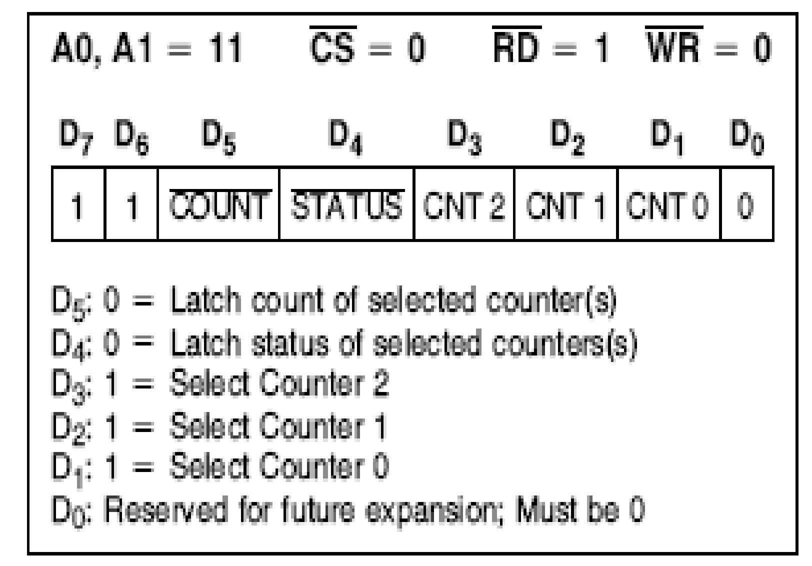
Figure 9. Counter Latching Command Format

- The count is then unlatched automatically and the OL returns to ``following" the counting element (CE).
- This allows reading the contents of the Counters ``on the fly" without affecting counting in progress.
- Multiple Counter Latch Commands may be used to latch more than one Counter. Each latched Counter's OL holds its count until it is read.
- Counter Latch Commands do not affect the programmed Mode of the Counter in any way.
- If a Counter is latched and then, some time later, latched again before the count is read, the second Counter Latch Command is ignored. The count read will be the count at the time the first Counter Latch Command was issued.
- With either method, the count must be read according to the programmed format; specifically, if the Counter is programmed for two byte counts, two bytes must be read. The two bytes do not have to be read one right after the other, read or write or programming operations of other Counters may be inserted between them.

- Another feature of the 8254 is that reads and writes of the same Counter may be interleaved.
- **Example**: If the Counter is programmed for two byte counts, the following sequence is valid.

1) Read least significant byte.
2) Write new least significant byte.
3) Read most significant byte.
4) Write new most significant byte.

- If a Counter is programmed to read/write two-byte counts, the following precaution applies: A program must not transfer control between reading the first and second byte to another routine which also reads from that same Counter. Otherwise, an incorrect count will be read.
- **READ-BACK COMMAND:** The third method uses the Read-Back Command. This command allows the user to check the count value, programmed Mode, and current states of the OUT pin and Null Count flag of the selected counter (s).
- The command is written into the Control Word Register and has the format shown in Figure 10. The command applies to the counters selected by setting their corresponding bits D3, D2, D1 = 1.
- The read-back command may be used to latch multiple counter output latches (OL) by setting the COUNT bit D5 = 0 and selecting the desired counter (s). This single command is functionally equivalent to several counter latch commands, one for each counter latched.



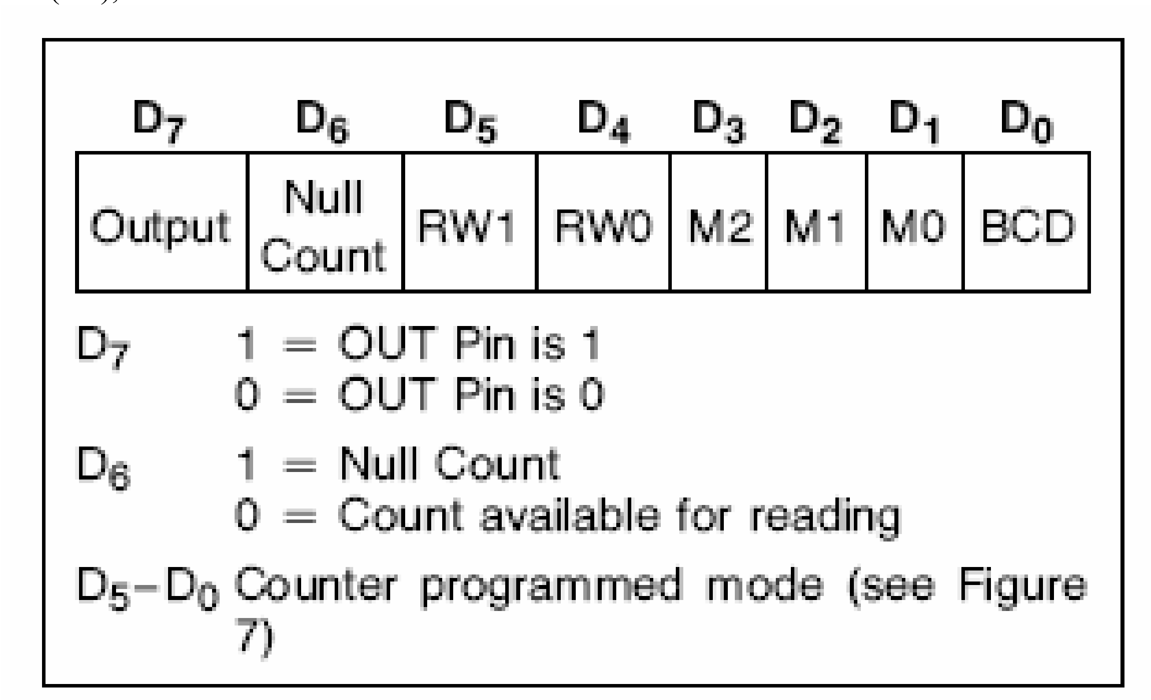
A0, A1 = 11 $\overline{CS}$ = 0 $\overline{RD}$ = 1 $\overline{WR}$ = 0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	$\overline{\text{COUNT}}$	$\overline{\text{STATUS}}$	CNT 2	CNT 1	CNT 0	0

D_5: 0 = Latch count of selected counter(s)
D_4: 0 = Latch status of selected counters(s)
D_3: 1 = Select Counter 2
D_2: 1 = Select Counter 1
D_1: 1 = Select Counter 0
D_0: Reserved for future expansion; Must be 0

- Each counter's latched count is held until it is read (or the counter is reprogrammed).
- The counter is automatically unlatched when read, but other counters remain latched until they are read. If multiple count read-back commands are issued to the same counter without reading the count, all but the first are ignored; i.e., the count which will be read is the count at the time the first read-back command was issued.

- The read-back command may also be used to latch status information of selected counter (s) by setting STATUS bit D4 = 0. Status must be latched to be read; status of a counter is accessed by a read from that counter.
- The counter status format is shown in Figure 11.
- Bits D5 through D0 contain the counter's programmed Mode exactly as written in the last Mode Control Word. OUTPUT bit D7 contains the current state of the OUT pin.
- This allows the user to monitor the counter's output via software, possibly eliminating some hardware from a system. NULL COUNT bit D6 indicates when the last count written to the counter register (CR) has been loaded into the counting element (CE).
- The exact time this happens depends on the Mode of the counter and is described in the Mode Definitions, but until the count is loaded into the counting element (CE), it can't be read from the counter.



D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Output	Null Count	RW1	RW0	M2	M1	M0	BCD

D_7 1 = OUT Pin is 1
0 = OUT Pin is 0

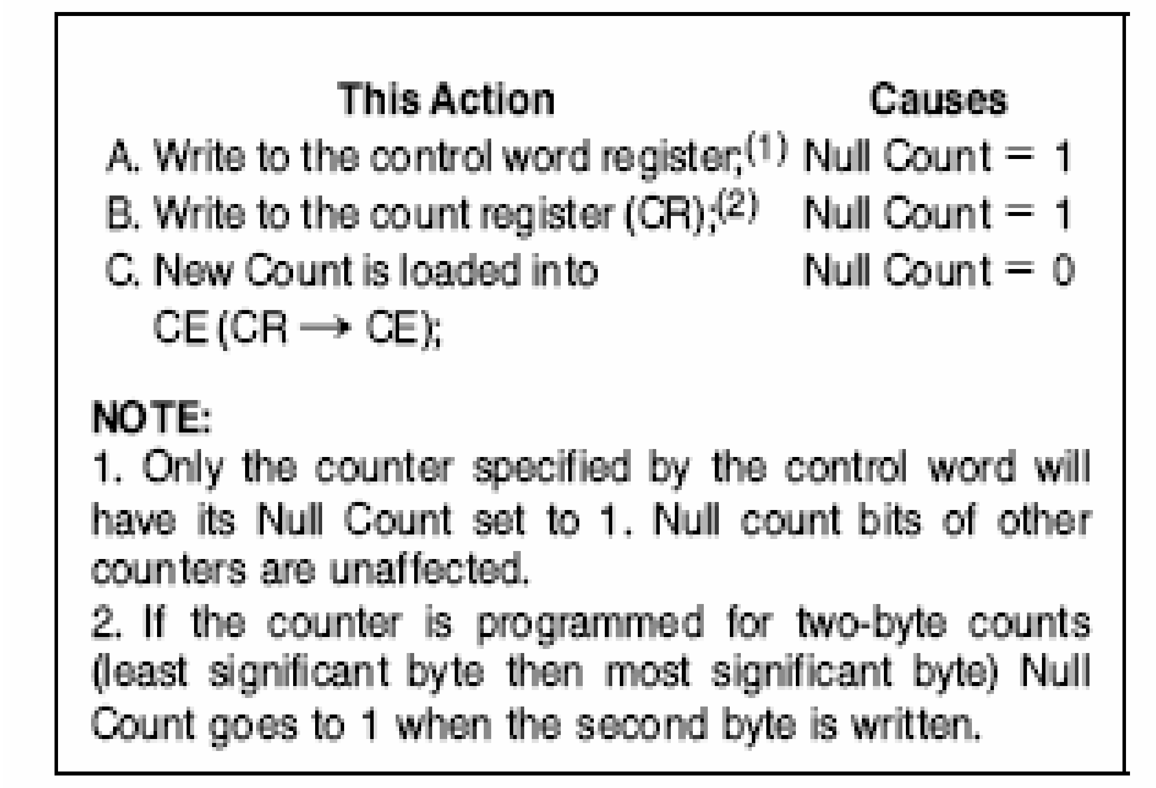
D_6 1 = Null Count
0 = Count available for reading

D_5-D_0 Counter programmed mode (see Figure 7)

Figure 11. Status Byte

- If the count is latched or read before this time, the count value will not reflect the new count just written. The operation of Null Count is shown in Figure 12.
- If multiple status latch operations of the counter (s) are performed without reading the status, all but the first are ignored; i.e., the status that will be read is the status of the counter at the time the first status read-back command was issued.

- Both count and status of the selected counter (s) may be latched simultaneously by setting both COUNT and STATUS bits D5,D4 = 0. This is functionally the same as issuing two separate read-back commands at once, and the above discussions apply here also.



This Action	Causes
A. Write to the control word register;[1]	Null Count = 1
B. Write to the count register (CR);[2]	Null Count = 1
C. New Count is loaded into CE (CR → CE);	Null Count = 0

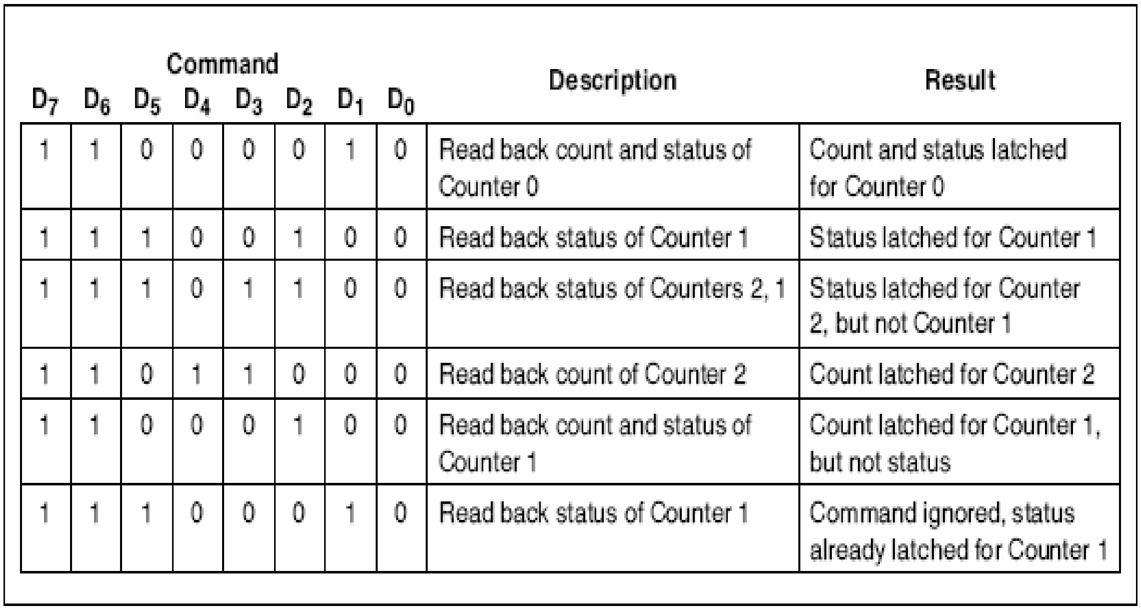
NOTE:

1. Only the counter specified by the control word will have its Null Count set to 1. Null count bits of other counters are unaffected.

2. If the counter is programmed for two-byte counts (least significant byte then most significant byte) Null Count goes to 1 when the second byte is written.

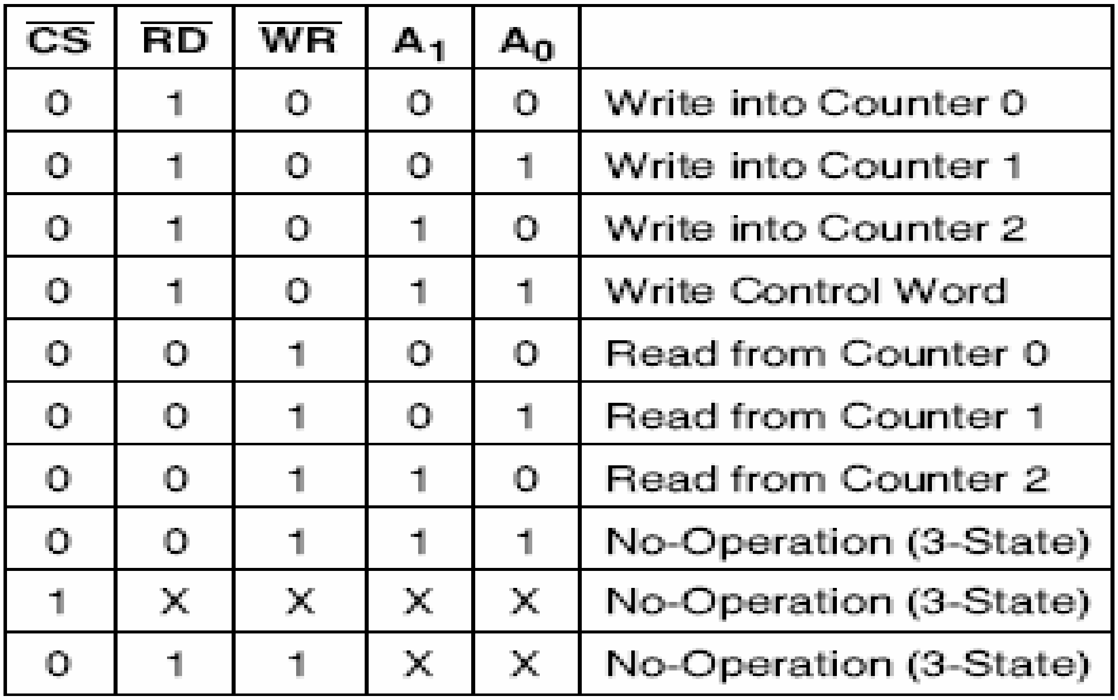
Figure 12. Null Count Operation

- Specifically, if multiple count and/or status read-back commands are issued to the same counter (s) without any intervening reads, all but the first are ignored. This is illustrated in Figure 13.
- If both count and status of a counter are latched, the first read operation of that counter will return latched status, regardless of which was latched first. The next one or two reads (depending on whether the counter is programmed for one or two type counts) return latched count. Subsequent reads return unlatched count.



Command								Description	Result
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0		
1	1	0	0	0	0	1	0	Read back count and status of Counter 0	Count and status latched for Counter 0
1	1	1	0	0	1	0	0	Read back status of Counter 1	Status latched for Counter 1
1	1	1	0	1	1	0	0	Read back status of Counters 2, 1	Status latched for Counter 2, but not Counter 1
1	1	0	1	1	0	0	0	Read back count of Counter 2	Count latched for Counter 2
1	1	0	0	0	1	0	0	Read back count and status of Counter 1	Count latched for Counter 1, but not status
1	1	1	0	0	0	1	0	Read back status of Counter 1	Command ignored, status already latched for Counter 1

Figure 13. Read-Back Command Example



$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_1	A_0	
0	1	0	0	0	Write into Counter 0
0	1	0	0	1	Write into Counter 1
0	1	0	1	0	Write into Counter 2
0	1	0	1	1	Write Control Word
0	0	1	0	0	Read from Counter 0
0	0	1	0	1	Read from Counter 1
0	0	1	1	0	Read from Counter 2
0	0	1	1	1	No-Operation (3-State)
1	X	X	X	X	No-Operation (3-State)
0	1	1	X	X	No-Operation (3-State)

Figure 14. Read/Write Operations Summary

- **Mode Definitions** :The following are defined for use in describing the operation of the 8254.
- **CLK Pulse**: A rising edge, then a falling edge, in that order, of a Counter's CLK input.
- **Trigger**: A rising edge of a Counter's GATE input.
- **Counter loading**: The transfer of a count from the CR to the CE (refer to the ``Functional Description").

- **MODE 0**: ***INTERRUPT ON TERMINAL COUNT*** :

- Mode 0 is typically used for event counting. After the Control Word is written, OUT is initially low, and will remain low until the Counter reaches zero.
- OUT then goes high and remains high until a new count or a new Mode 0 Control Word is written into the Counter.
- GATE = 1 enables counting; GATE = 0 disables counting. GATE has no effect on OUT.
- After the Control Word and initial count are written to a Counter, the initial count will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not go high until N a 1 CLK pulses after the initial count is written.
- If a new count is written to the Counter, it will be loaded on the next CLK pulse and counting will continue from the new count. If a two-byte count is written, the following happens:

1) Writing the first byte disables counting. OUT is set low immediately (no clock pulse required).
2) Writing the second byte allows the new count to be loaded on the next CLK pulse.

- This allows the counting sequence to be synchronized by software. Again, OUT does not go high until Na1 CLK pulses after the new count of N is written.
- If an initial count is written while GATE e 0, it will still be loaded on the next CLK pulse. When GATE goes high, OUT will go high N CLK pulses later; no CLK pulse is needed to load the Counter as this has already been done.

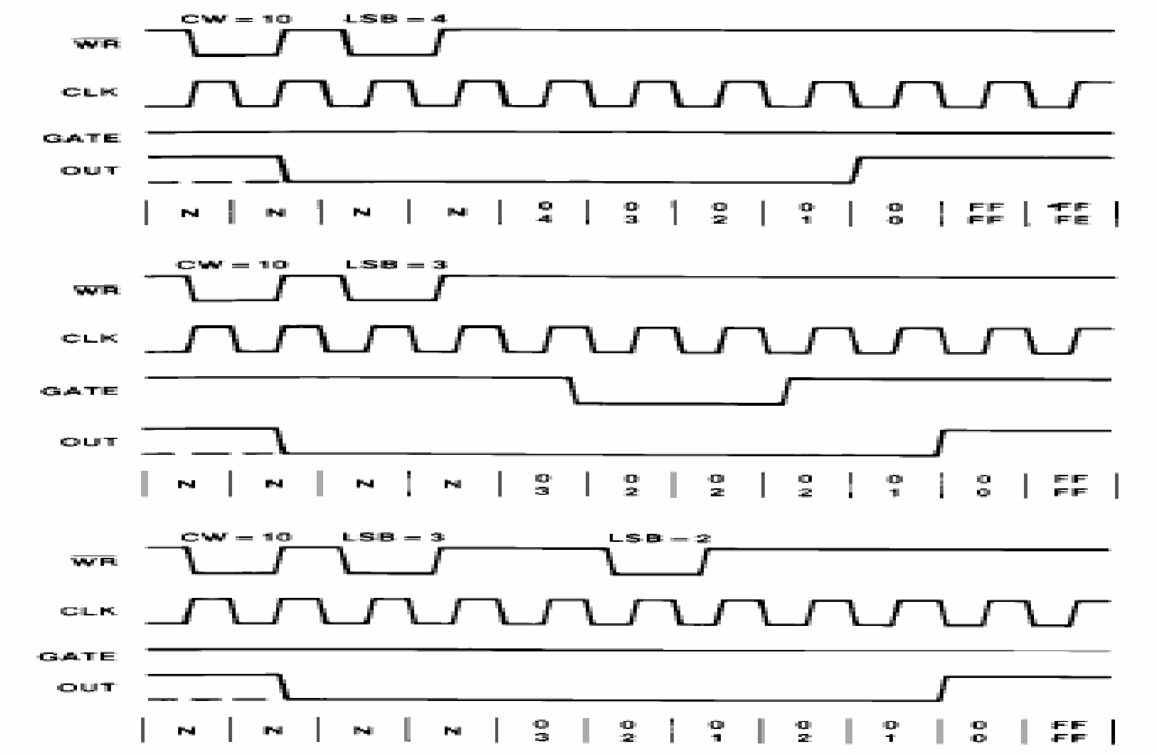

CW = 10
LSB = 4
WR
CLK
GATE
OUT
N N N N 0 4 0 3 0 2 0 1 0 0 FF FF FF FE
CW = 10
LSB = 3
WR
CLK
GATE
OUT
N N N N 0 3 0 2 0 2 0 2 0 1 0 0 FF FF
CW = 10
LSB = 3
LSB = 2
WR
CLK
GATE
OUT
N N N N 0 3 0 2 0 1 0 2 0 1 0 0 FF FF

Figure 15. Mode 0

Note:

1. Counters are programmed for binary (not BCD) counting and for reading/writing least significant byte (LSB) only.

2. The counter is always selected (CS always low).

3. CW stands for ``Control Word"; CW = 10 means a control word of 10 HEX is written to the counter.

4. LSB stands for ``Least Significant Byte" of count.

5. Numbers below diagrams are count values. The lower number is the least significant byte. The upper number is the most significant byte. Since the counter is programmed to read/write LSB only, the most significant byte cannot be read. N stands for an undefined count. Vertical lines show transitions between count values.

- **MODE 1**: ***HARDWARE RETRIGGERABLE ONE-SHOT***: OUT will be initially high.
- OUT will go low on the CLK pulse following a trigger to begin the one-shot pulse, and will remain low until the Counter reaches zero.
- OUT will then go high and remain high until the CLK pulse after the next trigger.
- After writing the Control Word and initial count, the Counter is armed. A trigger results in loading the Counter and setting OUT low on the next CLK pulse, thus starting the one-shot pulse. An initial count of N will result in a one-shot pulse N CLK cycles in duration.

- The one-shot is retriggerable, hence OUT will remain low for N CLK pulses after any trigger. The one-shot pulse can be repeated without rewriting the same count into the counter. GATE has no effect on OUT.
- If a new count is written to the Counter during a oneshot pulse, the current one-shot is not affected unless the counter is retriggered. In that case, the Counter is loaded with the new count and the oneshot pulse continues until the new count expires.

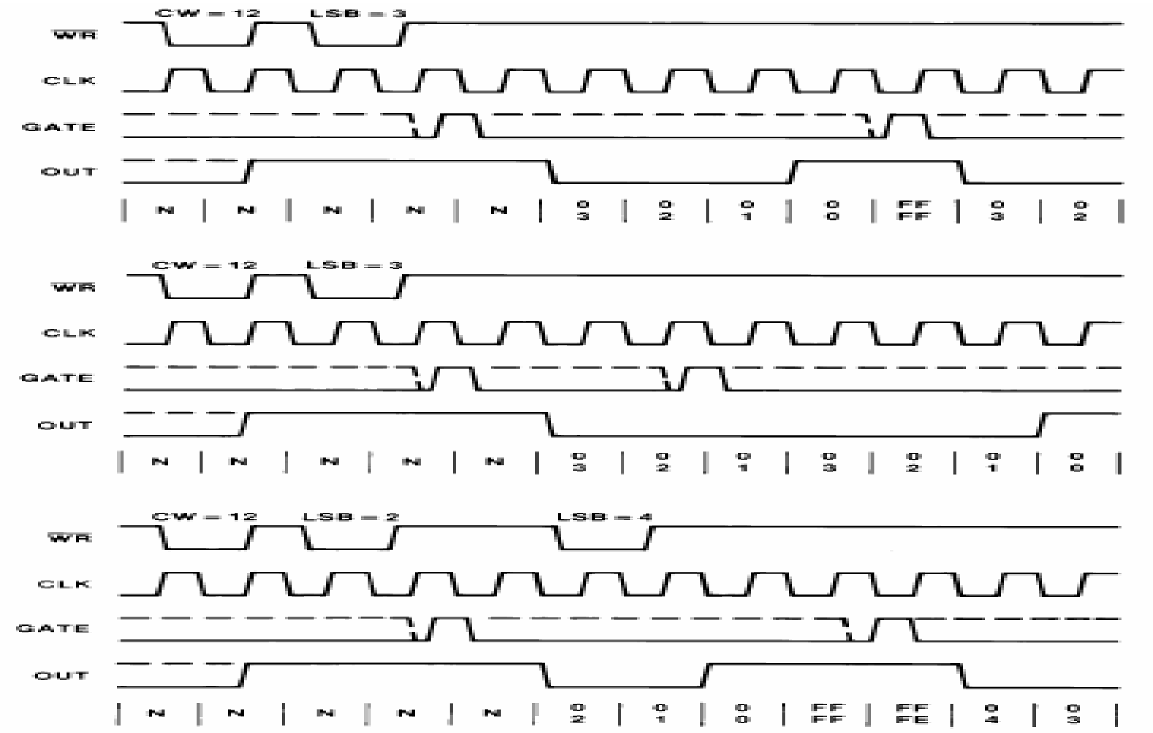


Figure 16. Mode 1

- **MODE 2**: RATE GENERATOR: This Mode functions like a divide-by-N counter. It is typically used to generate a Real Time Clock interrupt.
- OUT will initially be high. When the initial count has decremented to 1, OUT goes low for one CLK pulse. OUT then goes high again, the Counter reloads the initial count and the process is repeated.
- Mode 2 is periodic, the same sequence is repeated indefinitely. For an initial count of N, the sequence repeats every N CLK cycles.
- GATE = 1 enables counting; GATE = 0 disables counting. If GATE goes low during an output pulse, OUT is set high immediately.
- A trigger reloads the Counter with the initial count on the next CLK pulse, OUT goes low N CLK pulses after the trigger. Thus the GATE input can be used to synchronize the Counter.
- After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. OUT goes low N CLK Pulses after the initial count is written.
- This allows the Counter to be synchronized by software also. Writing a new count while counting does not affect the current counting sequence.

- If a trigger is received after writing a new count but before the end of the current period, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from the new count.
- Otherwise, the new count will be loaded at the end of the current counting cycle. In mode 2, a COUNT of 1 is illegal.
- **MODE 3**: SQUARE WAVE MODE :Mode 3 is typically used for Baud rate generation. Mode 3 is similar to Mode 2 except for the duty cycle of OUT. OUT will initially be high.

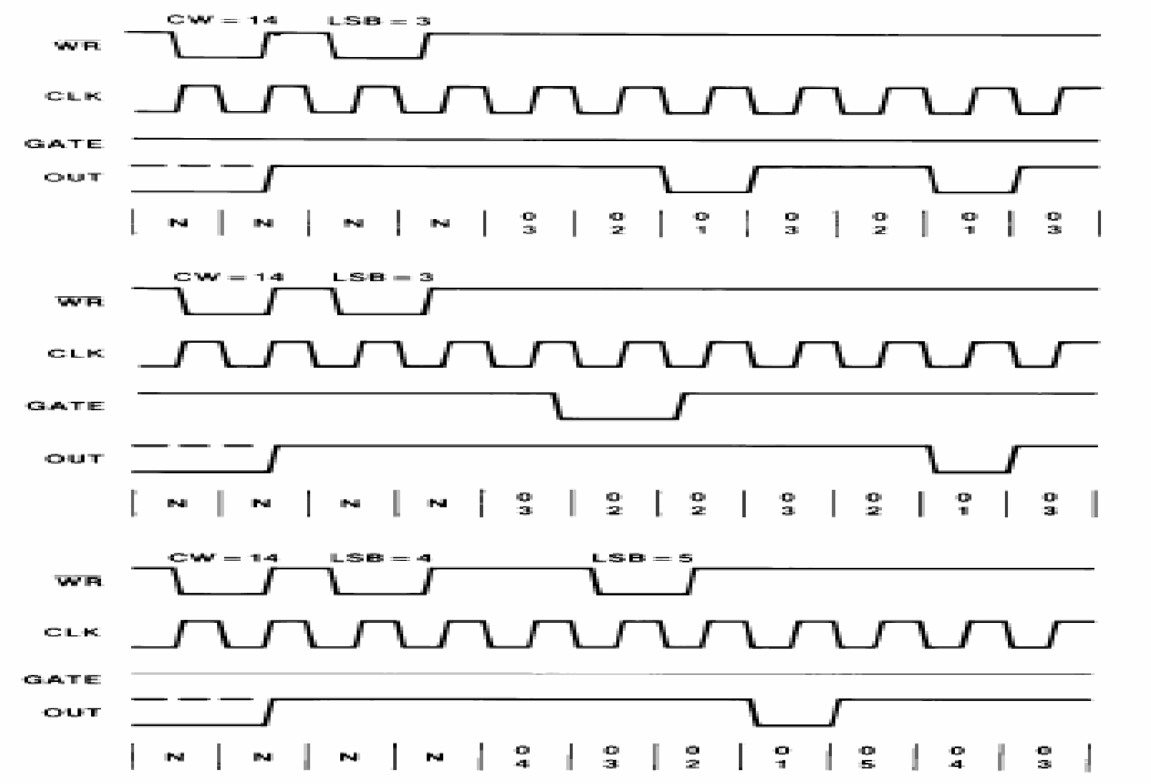


Figure 17. Mode 2

- When half the initial count has expired, OUT goes low for the remainder of the count. Mode 3 is periodic; the sequence above is repeated indefinitely.
- An initial count of N results in a square wave with a period of N CLK cycles. GATE = 1 enables counting; GATE = 0 disables counting. If GATE goes low while OUT is low, OUT is set high immediately; no CLK pulse is required.
- A trigger reloads the Counter with the initial count on the next CLK pulse. Thus the GATE input can be used to synchronize the Counter.
- After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. This allows the Counter to be synchronized by software also.
- Writing a new count while counting does not affect the current counting sequence. If a trigger is received after writing a new count but before the end of the current half-cycle of the square wave, the Counter will be loaded with the new

count on the next CLK pulse and counting will continue from the new count. Otherwise, the new count will be loaded at the end of the current half-cycle.

- **Mode 3:Even counts**: OUT is initially high. The initial count is loaded on one CLK pulse and then is decremented by two on succeeding CLK pulses.
- When the count expires OUT changes value and the Counter is reloaded with the initial count. The above process is repeated indefinitely.
- **Odd counts**: OUT is initially high. The initial count minus one (an even number) is loaded on one CLK pulse and then is decremented by two on succeeding CLK pulses.

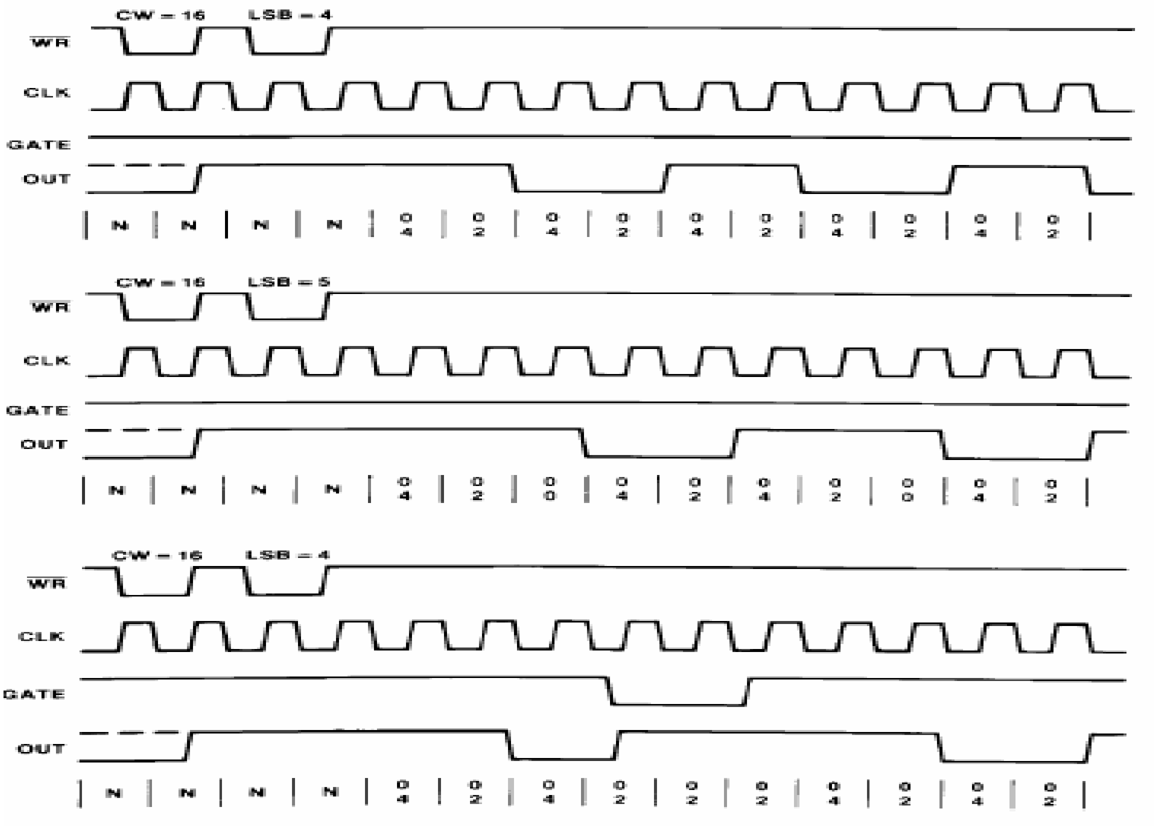


Figure 18. Mode 3

- One CLK pulse after the count expires, OUT goes low and the Counter is reloaded with the initial count minus one.
- Succeeding CLK pulses decrement the count by two.
- When the count expires, OUT goes high again and the Counter is reloaded with the initial count minus one. The above process is repeated indefinitely.
- So for odd counts, OUT will be high for (N - 1)/2 counts and low for (N - 1)/2 counts.
- **MODE 4: SOFTWARE TRIGGERED STROBE :**

- OUT will be initially high. When the initial count expires, OUT will go low for one CLK pulse and then go high again. The counting sequence is ``triggered'' by writing the initial count.
- GATE = 1 enables counting; GATE = 0 disables counting. GATE has no effect on OUT. After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse.
- This CLK pulse does not decrement the count, so for an initial count of N, OUT does not strobe low until N + 1 CLK pulses after the initial count is written.
- If a new count is written during counting, it will be loaded on the next CLK pulse and counting will continue from the new count. If a two-byte count is written, the following happens:

1) Writing the first byte has no effect on counting.
2) Writing the second byte allows the new count to be loaded on the next CLK pulse.

- This allows the sequence to be ``retriggered" by software. OUT strobes low N a 1 CLK pulses after the new count of N is written.

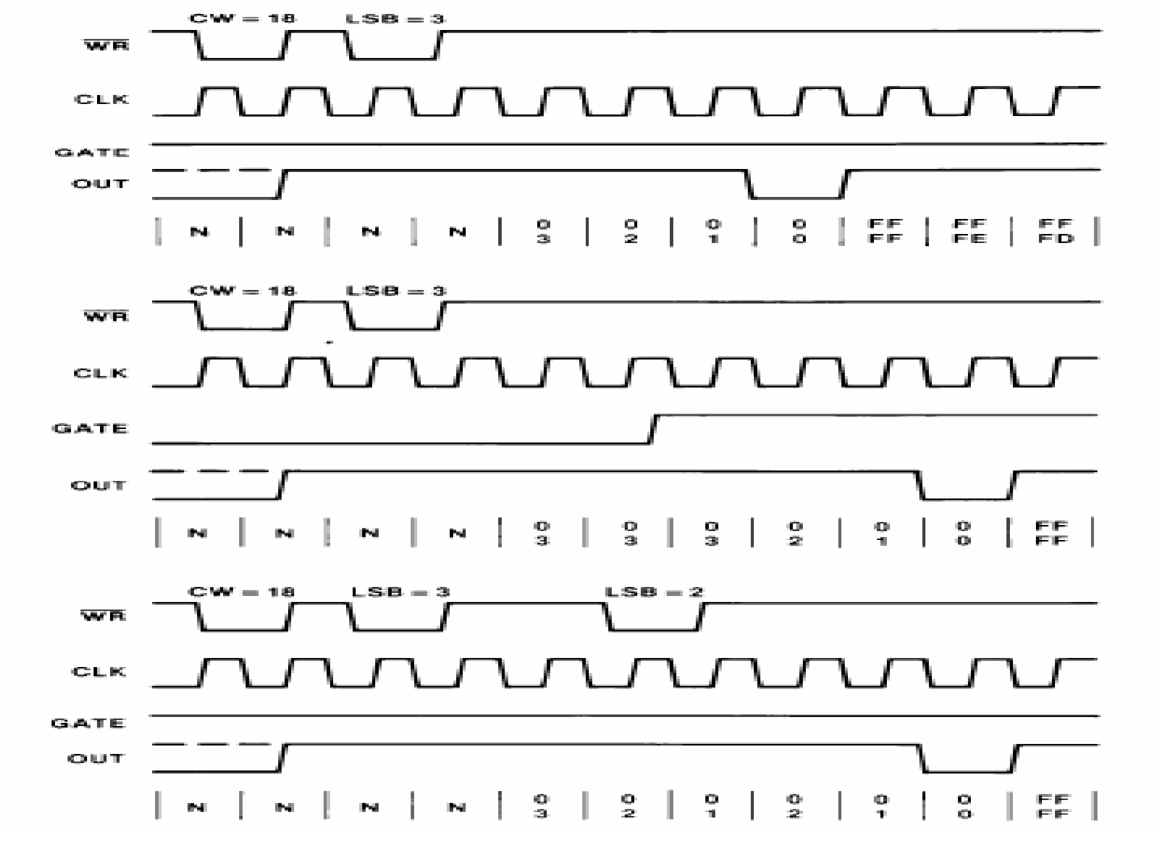


Figure 19. Mode 4

- **MODE 5: HARDWARE TRIGGERED STROBE (RETRIGGERABLE):** OUT will initially be high. Counting is triggered by a rising edge of GATE. When the initial count has expired, OUT will go low for one CLK pulse and then go high again.

- After writing the Control Word and initial count, the counter will not be loaded until the CLK pulse after a trigger. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not strobe low until N = 1 CLK pulses after a trigger.
- A trigger results in the Counter being loaded with the initial count on the next CLK pulse. The counting sequence is retriggerable. OUT will not strobe low for N a 1 CLK pulses after any trigger. GATE has no effect on OUT.
- If a new count is written during counting, the current counting sequence will not be affected. If a trigger occurs after the new count is written but before the current count expires, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from there.

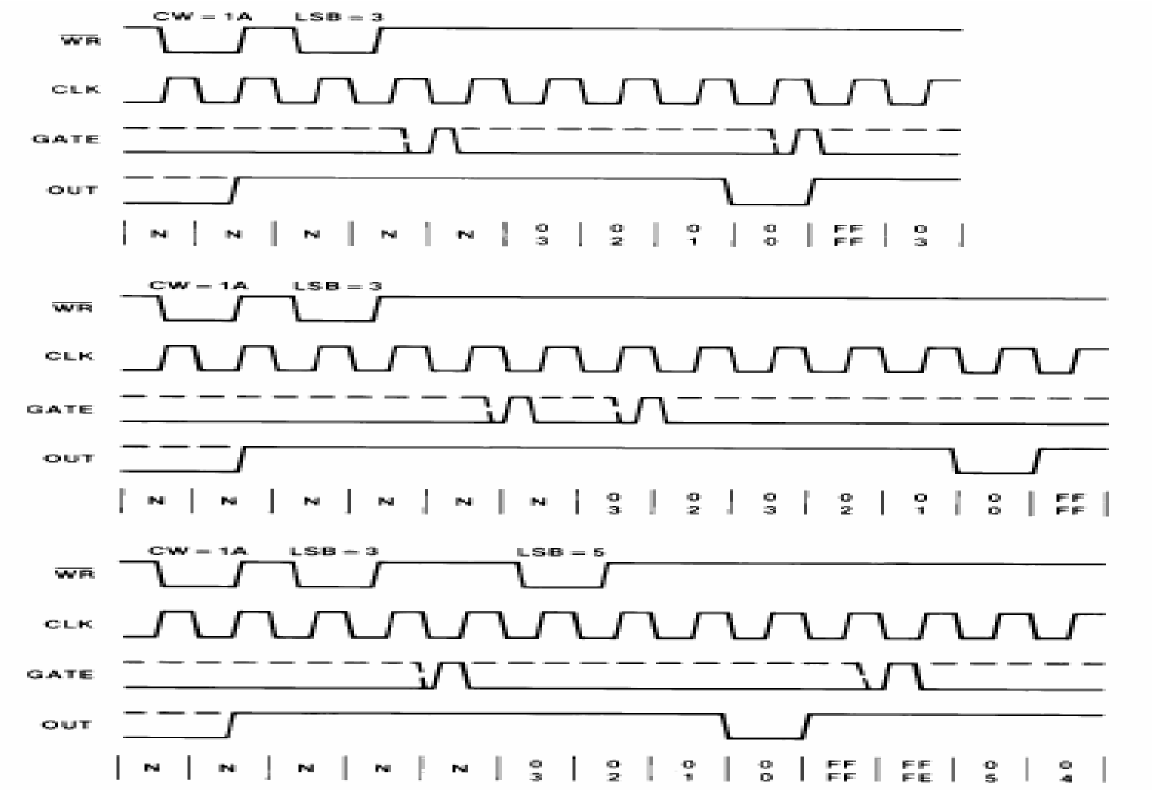
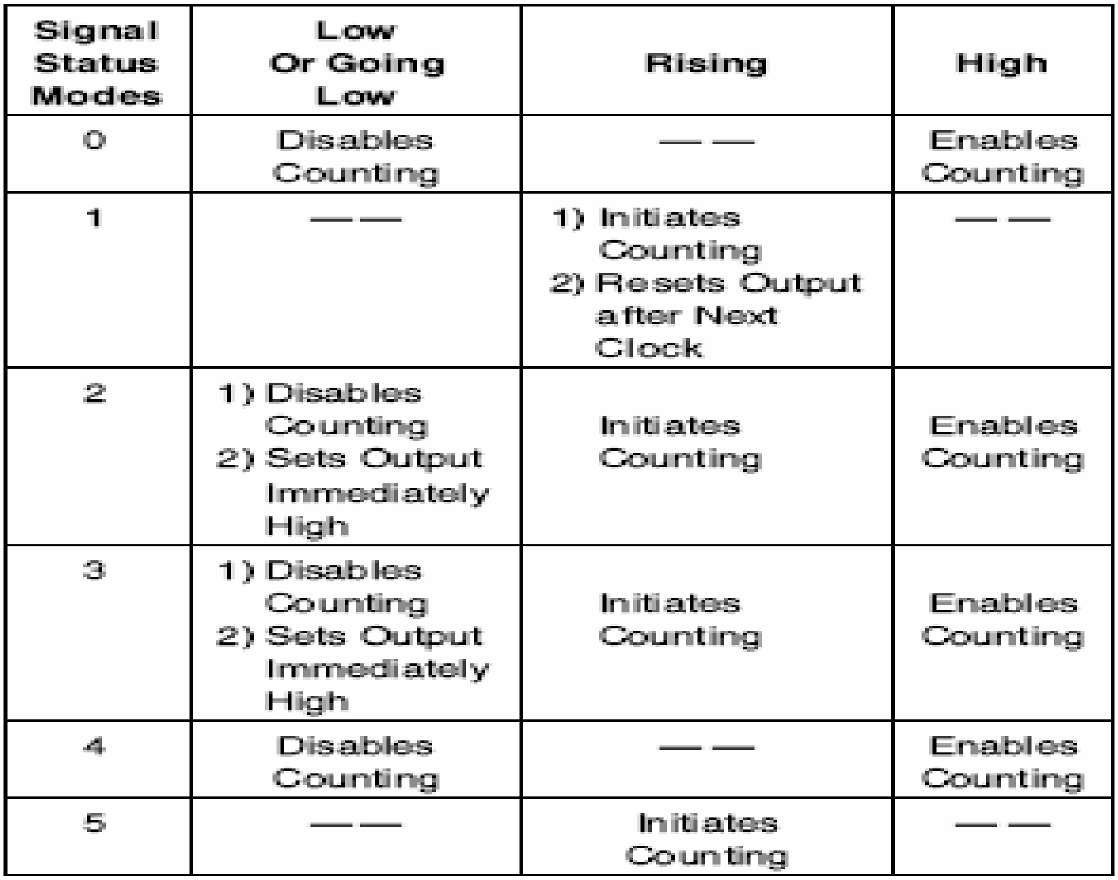


Figure 20. Mode 5

- **Operation Common to All Modes**:
- **PROGRAMMING:** When a Control Word is written to a Counter, all Control Logic is immediately reset and OUT goes to a known initial state; no CLK pulses are required for this.
- **GATE**: The GATE input is always sampled on the rising edge of CLK. In Modes 0, 2, 3, and 4 the GATE input is level sensitive, and the logic level is sampled on the rising edge of CLK. In Modes 1, 2, 3, and 5 the GATE input is rising-edge sensitive.

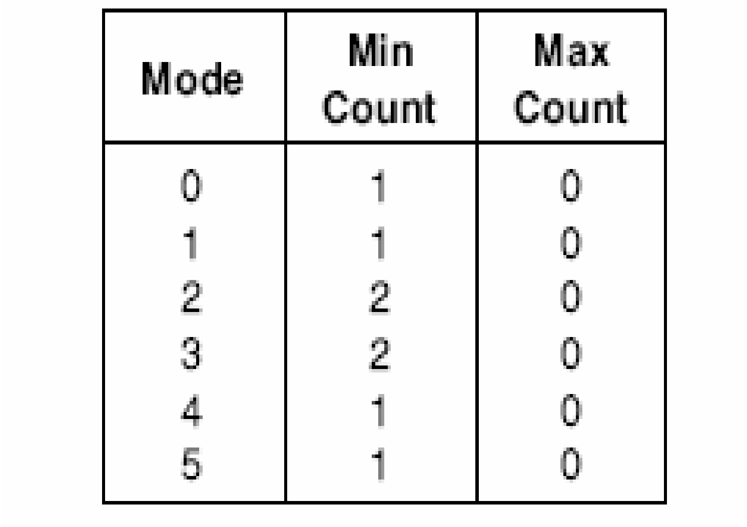
- In these Modes, a rising edge of GATE (trigger) sets an edge-sensitive flip-flop in the Counter. This flip-flop is then sampled on the next rising edge of CLK; the flip-flop is reset immediately after it is sampled. In this way, a trigger will be detected no matter when it occurs-a high logic level does not have to be maintained until the next rising edge of CLK.
- Note that in Modes 2 and 3, the GATE input is both edge- and level-sensitive. In Modes 2 and 3, if a CLK source other than the system clock is used, GATE should be pulsed immediately following WR of a new count value.



Signal Status Modes	Low Or Going Low	Rising	High
0	Disables Counting	— —	Enables Counting
1	— —	1) Initiates Counting 2) Resets Output after Next Clock	— —
2	1) Disables Counting 2) Sets Output Immediately High	Initiates Counting	Enables Counting
3	1) Disables Counting 2) Sets Output Immediately High	Initiates Counting	Enables Counting
4	Disables Counting	— —	Enables Counting
5	— —	Initiates Counting	— —

Figure 21. Gate Pin Operations Summary

- **COUNTER**: New counts are loaded and Counters are decremented on the falling edge of CLK.
- The largest possible initial count is 0, this is equivalent to 216 for binary counting and 104 for BCD counting. The Counter does not stop when it reaches zero.
- In Modes 0, 1, 4, and 5 the Counter ``wraps around" to the highest count, either FFFF hex for binary counting or 9999 for BCD counting, and continues counting.
- Modes 2 and 3 are periodic; the Counter reloads itself with the initial count and continues counting from there.



Mode	Min Count	Max Count
0	1	0
1	1	0
2	2	0
3	2	0
4	1	0
5	1	0

NOTE: 0 is equivalent to 2^{16} for binary counting and 10^{4} for BCD counting.

Figure 22. Minimum and Maximum Initial Counts

8279

- While studying 8255, we have explained the use of 8255 in interfacing keyboards and displays with 8086. The disadvantages of this method of interfacing keyboard and display with 8086 is that the processor has to refresh the display and check the status of the keyboard periodically using polling technique. Thus a considerable amount of CPU time is wasted, reducing the system operating speed.
- Intel's 8279 is a general purpose keyboard display controller that simultaneously drives the display of a system and interfaces a keyboard with the CPU, leaving it free for its routine task.

Architecture and Signal Descriptions of 8279

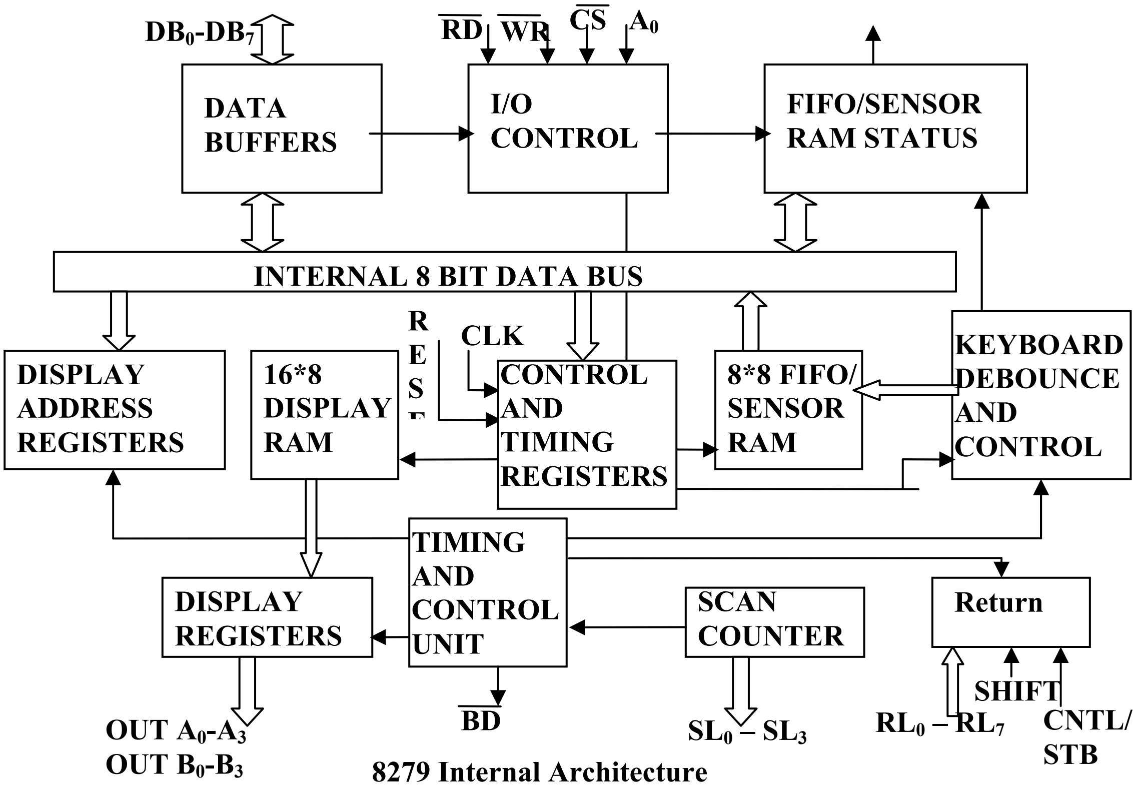
- The keyboard display controller chip 8279 provides:

a) a set of four scan lines and eight return lines for interfacing keyboards

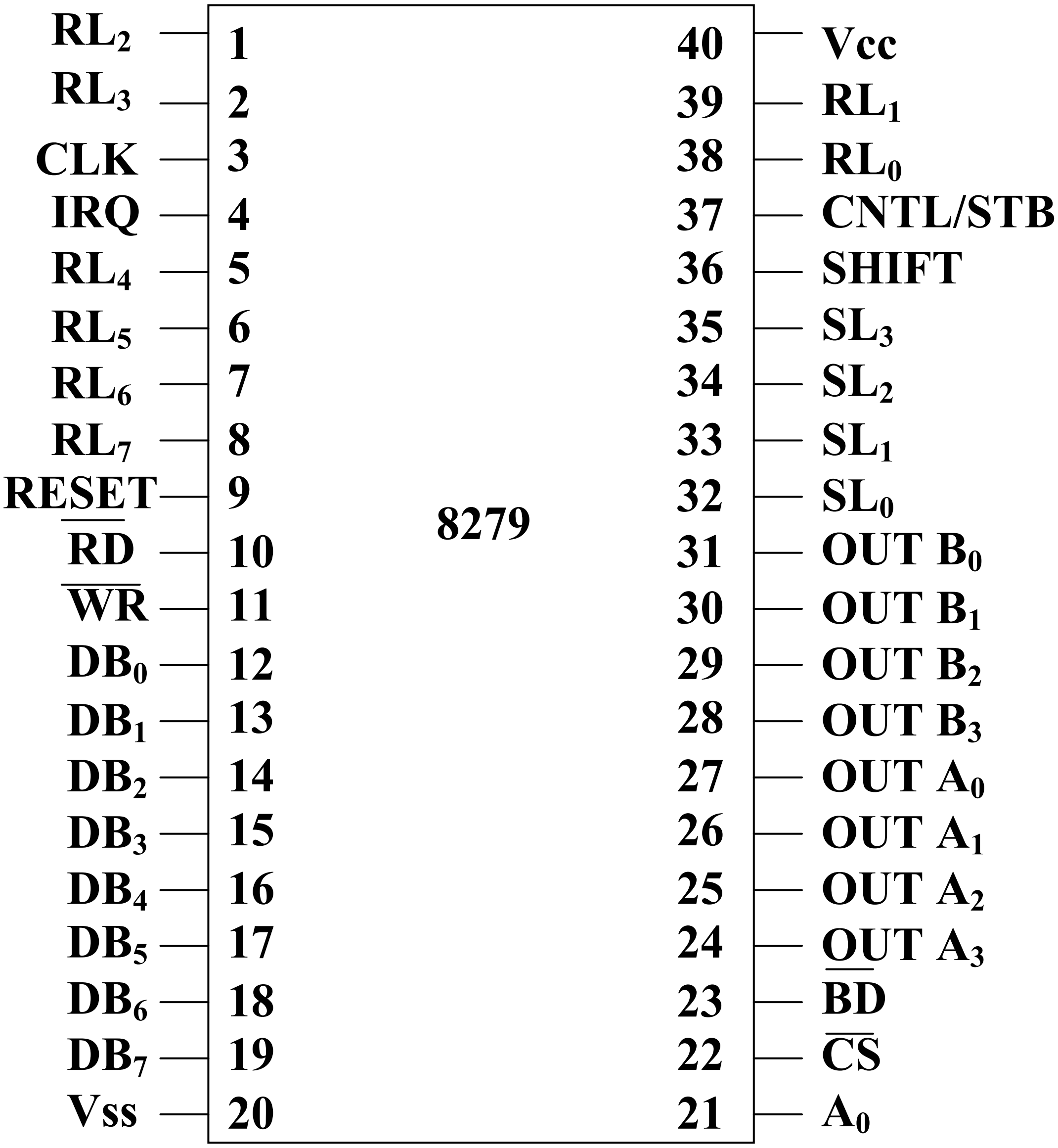
b) A set of eight output lines for interfacing display.

- Fig shows the functional block diagram of 8279 followed by its brief description.
- **I/O Control and Data Buffers** : The I/O control section controls the flow of data to/from the 8279. The data buffers interface the external bus of the system with internal bus of 8279.
- The I/O section is enabled only if CS is low. The pins A0, RD and WR select the command, status or data read/write operations carried out by the CPU with 8279.
- **Control and Timing Register and Timing Control** : These registers store the keyboard and display modes and other operating conditions programmed by CPU. The registers are written with A0=1 and WR=0. The Timing and control unit controls the basic timings for the operation of the circuit. Scan counter divide down the operating frequency of 8279 to derive scan keyboard and scan display frequencies.
- **Scan Counter** : The scan counter has two modes to scan the key matrix and refresh the display. In the encoded mode, the counter provides binary count that is to be externally decoded to provide the scan lines for keyboard and display (Four externally decoded scan lines may drive upto 16 displays). In the decode scan mode, the counter internally decodes the least significant 2 bits and provides a decoded 1 out of 4 scan on SL0-SL3(Four internally decoded scan lines may drive upto 4 displays). The keyboard and display both are in the same mode at a time.
- **Return Buffers and Keyboard Debounce and Control**: This section for a key closure row wise. If a key closer is detected, the keyboard debounce unit debounces the key entry (i.e. wait for 10 ms). After the debounce period, if the key continues to be detected. The code of key is directly transferred to the sensor RAM along with SHIFT and CONTROL key status.

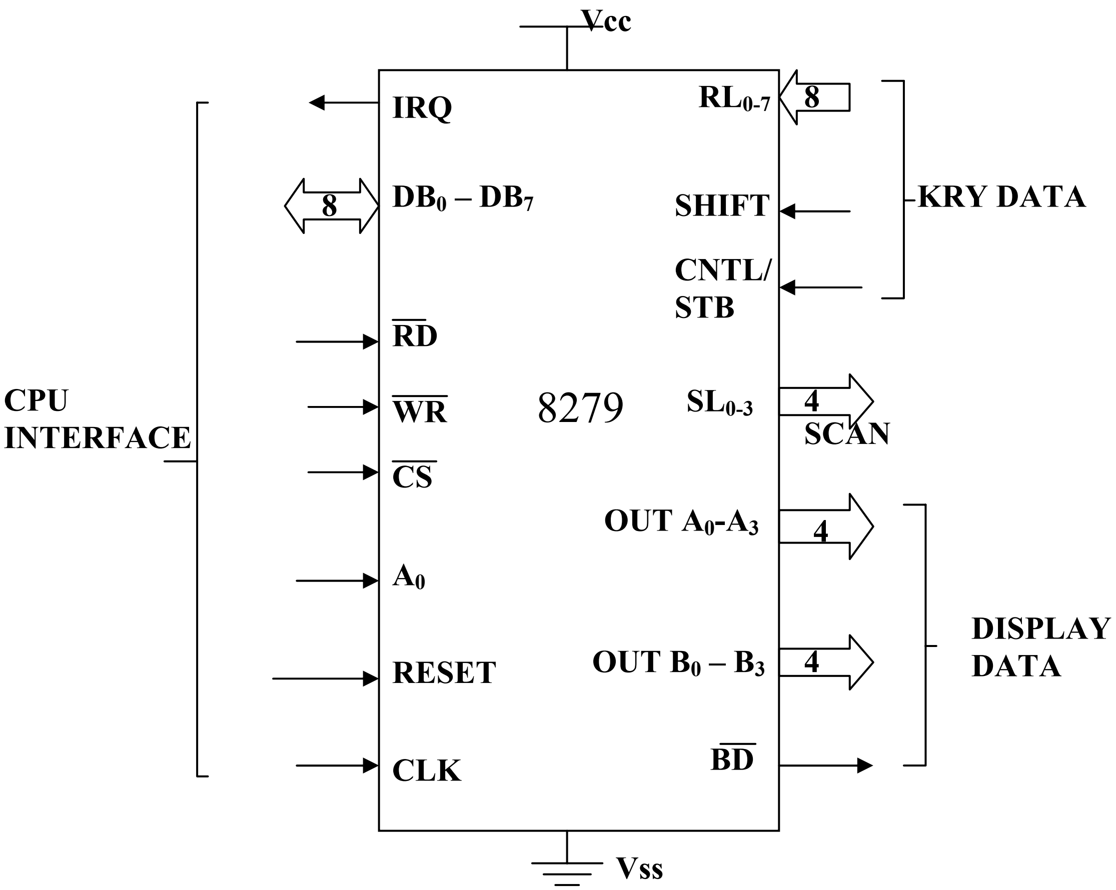
- **FIFO/Sensor RAM and Status Logic**: In keyboard or strobed input mode, this block acts as 8-byte first-in-first-out (FIFO) RAM. Each key code of the pressed key is entered in the order of the entry and in the mean time read by the CPU, till the RAM become empty.
- The status logic generates an interrupt after each FIFO read operation till the FIFO is empty. In scanned sensor matrix mode, this unit acts as sensor RAM. Each row of the sensor RAM is loaded with the status of the corresponding row of sensors in the matrix. If a sensor changes its state, the IRQ line goes high to interrupt the CPU.
- **Display Address Registers and Display RAM** : The display address register holds the address of the word currently being written or read by the CPU to or from the display RAM. The contents of the registers are automatically updated by 8279 to accept the next data entry by CPU.



8279 Internal Architecture



8279 Pin Configuration



- The signal discription of each of the pins of 8279 as follows :
- **DB0-DB7** : These are bidirectional data bus lines. The data and command words to and from the CPU are transferred on these lines.
- **CLK** : This is a clock input used to generate internal timing required by 8279.
- **RESET** : This pin is used to reset 8279. A high on this line reset 8279. After resetting 8279, its in sixteen 8-bit display, left entry encoded scan, 2-key lock out mode. The clock prescaler is set to 31.
- **CS** : Chip Select – A low on this line enables 8279 for normal read or write operations. Other wise, this pin should remain high.
- **A0** : A high on this line indicates the transfer of a command or status information. A low on this line indicates the transfer of data. This is used to select one of the internal registers of 8279.
- **RD, WR (Input/Output) READ/WRITE** – These input pins enable the data buffers to receive or send data over the data bus.
- **IRQ** : This interrupt output lines goes high when there is a data in the FIFO sensor RAM. The interrupt lines goes low with each FIFO RAM read operation but if the FIFO RAM further contains any key-code entry to be read by the CPU, this pin again goes high to generate an interrupt to the CPU.
- **Vss, Vcc** : These are the ground and power supply lines for the circuit.
- **SL0-SL3-Scan Lines** : These lines are used to scan the key board matrix and display digits. These lines can be programmed as encoded or decoded, using the mode control register.

- **RL0 - RL7 - Return Lines** : These are the input lines which are connected to one terminal of keys, while the other terminal of the keys are connected to the decoded scan lines. These are normally high, but pulled low when a key is pressed.
- **SHIFT** : The status of the shift input lines is stored along with each key code in FIFO, in scanned keyboard mode. It is pulled up internally to keep it high, till it is pulled low with a key closure.
- **BD – Blank Display** : This output pin is used to blank the display during digit switching or by a blanking closure.
- **OUT A0 – OUT A3 and OUT B0 – OUT B3** – These are the output ports for two 16*4 or 16*8 internal display refresh registers. The data from these lines is synchronized with the scan lines to scan the display and keyboard. The two 4-bit ports may also as one 8-bit port.
- **CNTL/STB- CONTROL/STROBED I/P Mode** : In keyboard mode, this lines is used as a control input and stored in FIFO on a key closure. The line is a strobed lines that enters the data into FIFO RAM, in strobed input mode. It has an interrupt pull up. The lines is pulled down with a key closer.

Modes of Operation of 8279

- The modes of operation of 8279 are as follows :

1. Input (Keyboard) modes.
2. Output (Display) modes.

- **Input (Keyboard) Modes :** 8279 provides three input modes. These modes are as follows:

1. **Scanned Keyboard Mode** : This mode allows a key matrix to be interfaced using either encoded or decoded scans. In encoded scan, an 8*8 keyboard or in decoded scan, a 4*8 keyboard can be interfaced. The code of key pressed with SHIFT and CONTROL status is stored into the FIFO RAM.
2. **Scanned Sensor Matrix** : In this mode, a sensor array can be interfaced with 8279 using either encoded or decoded scans. With encoded scan 8*8 sensor matrix or with decoded scan 4*8 sensor matrix can be interfaced. The sensor codes are stored in the CPU addressable sensor RAM.
3. **Strobed input**: In this mode, if the control lines goes low, the data on return lines, is stored in the FIFO byte by byte.

- **Output (Display) Modes** : 8279 provides two output modes for selecting the display options. These are discussed briefly.

1. **Display Scan** : In this mode 8279 provides 8 or 16 character multiplexed displays those can be organized as dual 4- bit or single 8-bit display units.
2. **Display Entry** : (right entry or left entry mode) 8279 allows options for data entry on the displays. The display data is entered for display either from the right side or from the left side.

Keyboard Modes

i. **Scanned Keyboard mode with 2 Key Lockout** : In this mode of operation, when a key is pressed, a debounce logic comes into operation. During the next two scans, other keys are checked for closure and if no other key is pressed the first pressed key is identified.

- The key code of the identified key is entered into the FIFO with SHIFT and CNTL status, provided the FIFO is not full, i.e. it has at least one byte free. If the FIFO does not have any free byte, naturally the key data will not be entered and the error flag is set.
- If FIFO has at least one byte free, the above code is entered into it and the 8279 generates an interrupt on IRQ line to the CPU to inform about the previous key closures. If another key is found closed during the first key, the keycode is entered in FIFO.
- If the first pressed key is released before the others, the first will be ignored. A key code is entered to FIFO only once for each valid depression, independent of other keys pressed along with it, or released before it.
- If two keys are pressed within a debounce cycle (simultaneously), no key is recognized till one of them remains closed and the other is released. The last key, that remains depressed is considered as single valid key depression.

ii. **Scanned Keyboard with N-Key Rollover** : In this mode, each key depression is treated independently. When a key is pressed, the debounce circuit waits for 2 keyboards scans and then checks whether the key is still depressed. If it is still depressed, the code is entered in FIFO RAM.

Any number of keys can be pressed simultaneously and recognized in the order, the keyboard scan recorded them. All the codes of such keys are entered into FIFO.

In this mode, the first pressed key need not be released before the second is pressed. All the keys are sensed in the order of their depression, rather in the order the keyboard scan senses them, and independent of the order of their release.

iii. **Scanned Keyboard Special Error Mode :** This mode is valid only under the N-Key rollover mode. This mode is programmed using end interrupt / error mode set command. If during a single debounce period (two keyboard scans) two keys are found pressed , this is considered a simultaneous depression and an error flag is set**.**

- This flag, if set, prevents further writing in FIFO but allows the generation of further interrupts to the CPU for FIFO read. The error flag can be read by reading the FIFO status word. The error Flag is set by sending normal clear command with CF = 1.

iv. **Sensor Matrix Mode** : In the sensor matrix mode, the debounce logic is inhibited. The 8-byte FIFO RAM now acts as 8 * 8 bit memory matrix. The status of the sensor switch matrix is fed directly to sensor RAM matrix. Thus the sensor RAM bits contains the row-wise and column wise status of the sensors in the sensor matrix.

- The IRQ line goes high, if any change in sensor value is detected at the end of a sensor matrix scan or the sensor RAM has a previous entry to be read by the CPU. The IRQ line is reset by the first data read operation, if AI = 0, otherwise, by issuing the end interrupt command. AI is a bit in read sensor RAM word.

Display Modes

- There are various options of data display. For example, the command number of characters can be 8 or 16, with each character organised as single 8-bit or dual 4-bit codes. Similarly there are two display formats.
- The first one is known as left entry mode or type writer mode, since in a type writer the first character typed appears at the left-most position, while the subsequent characters appear successively to the right of the first one. The other display format is known as right entry mode, or calculator mode, since in a calculator the first character entered appears at the rightmost position and this character is shifted one position left when the next characters is entered.
- Thus all the previously entered characters are shifted left by one position when a new characters is entered.

i. **Left Entry Mode** : In the left entry mode, the data is entered from left side of the display unit. Address 0 of the display RAM contains the leftmost display characters and address 15 of the RAM contains the right most display characters. It is just like writing in our address is automatically updated with successive reads or writes. The first entry is displayed on the leftmost display and the sixteenth entry on the rightmost display. The seventeenth entry is again displayed at the leftmost display position.

ii. **Right Entry Mode** : In this right entry mode, the first entry to be displayed is entered on the rightmost display. The next entry is also placed in the right most display but after the previous display is shifted left by one display position. The leftmost characters is shifted out of that display at the seventeenth entry and is lost, i.e. it is pushed out of the display RAM.

Command Words of 8279

- All the command words or status words are written or read with A0 = 1 and CS = 0 to or from 8279. This section describes the various command available in 8279.

a) **Keyboard Display Mode Set** – The format of the command word to select different modes of operation of 8279 is given below with its bit definitions.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
0	0	D	D	D	K	K	K	1

D	**D**	**Display modes**
0	**0**	**Eight 8-bit character Left entry**
0	**1**	**Sixteen 8-bit character left entry**
1	**0**	**Eight 8-bit character Right entry**
1	**1**	**Sixteen 8-bit character Right entry**

K	**K**	**K**	**Keyboard modes**
0	**0**	**0**	**Encoded Scan, 2 key lockout (Default after reset)**
0	**0**	**1**	**Decoded Scan, 2 key lockout**
0	**1**	**0**	**Encoded Scan, N- key Roll over**
0	**1**	**1**	**Decoded Scan, N- key Roll over**
1	**0**	**0**	**Encode Scan, N- key Roll over**
1	**0**	**1**	**Decoded Scan, N- key Roll over**
1	**1**	**0**	**Strobed Input Encoded Scan**
1	**1**	**1**	**Strobed Input Decoded Scan**

b) **Programmable clock** : The clock for operation of 8279 is obtained by dividing the external clock input signal by a programmable constant called prescaler.

- PPPPP is a 5-bit binary constant. The input frequency is divided by a decimal constant ranging from 2 to 31, decided by the bits of an internal prescaler, PPPPP.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
0	0	1	P	P	P	P	P	1

c) **Read FIFO / Sensor RAM** : The format of this command is given below.

- This word is written to set up 8279 for reading FIFO/ sensor RAM. In scanned keyboard mode, AI and AAA bits are of no use. The 8279 will automatically drive data bus for each subsequent read, in the same sequence, in which the data was entered.
- In sensor matrix mode, the bits AAA select one of the 8 rows of RAM. If AI flag is set, each successive read will be from the subsequent RAM location.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
0	1	0	AI	X	A	A	A	1

X – don’t care
AI – Auto Increment Flag
AAA – Address pointer to 8 bit FIFO RAM

d) **Read Display RAM** : This command enables a programmer to read the display RAM data. The CPU writes this command word to 8279 to prepare it for display RAM read operation. AI is auto increment flag and AAAA, the 4-bit address points to the 16-byte display RAM that is to be read. If AI=1, the address will be automatically, incremented after each read or write to the Display RAM. The same address counter is used for reading and writing.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
0	1	1	AI	A	A	A	A	1

e) **Write Display RAM** :

AI – Auto increment Flag.
AAAA – 4 bit address for 16-bit display RAM to be written.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
1	0	0	AI	A	A	A	A	1

f) **Display Write Inhibit/Blanking** : The IW (inhibit write flag) bits are used to mask the individual nibble as shown in the below command word. The output lines are divided into two nibbles (OUTA0 – OUTA3) and (OUTB0 – OUTB3), those can be masked by setting the corresponding IW bit to 1.

- Once a nibble is masked by setting the corresponding IW bit to 1, the entry to display RAM does not affect the nibble even though it may change the unmasked nibble. The blank display bit flags (BL) are used for blanking A and B nibbles.
- Here D0, D2 corresponds to OUTB0 – OUTB3 while D1 and D3 corresponds to OUTA0-OUTA3 for blanking and masking.
- If the user wants to clear the display, blank (BL) bits are available for each nibble as shown in format. Both BL bits will have to be cleared for blanking both the nibbles.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
1	0	1	X	IW	IW	BL	BL	1

g) **Clear Display RAM** : The CD2, CD1, CD0 is a selectable blanking code to clear all the rows of the display RAM as given below. The characters A and B represents the output nibbles.

- CD2 must be 1 for enabling the clear display command. If CD2 = 0, the clear display command is invoked by setting CA=1 and maintaining CD1, CD0 bits exactly same as above. If CF=1, FIFO status is cleared and IRQ line is pulled down.
- Also the sensor RAM pointer is set to row 0. if CA=1, this combines the effect of CD and CF bits. Here, CA represents Clear All and CF as Clear FIFO RAM.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
1	1	0	CD_2	CD_1	CD_0	CF	CA	1

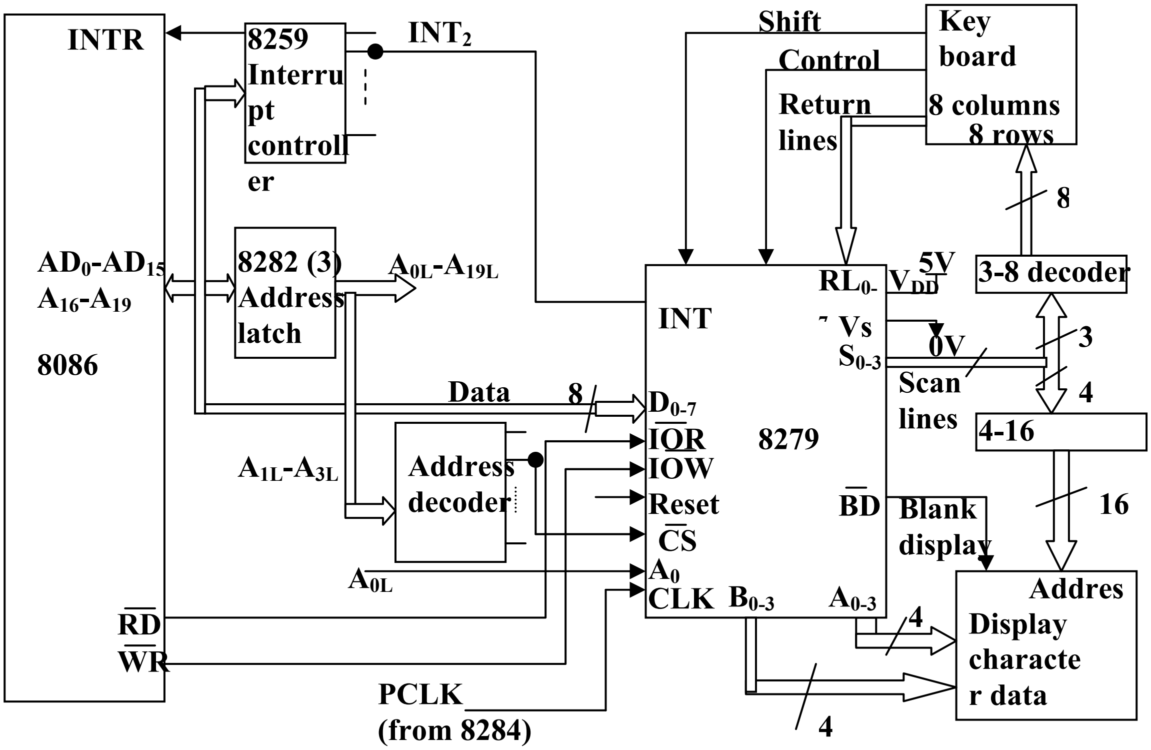
CD_2	CD_1	CD_0
1	0	X
1	1	0
1	1	1

All zeros (x don't care) AB=00
A3-A0 =2 (0010) and B3-B0=00 (0000)
All ones (AB =FF), i.e. clear RAM

h) **End Interrupt / Error mode Set** : For the sensor matrix mode, this command lowers the IRQ line and enables further writing into the RAM. Otherwise, if a change in sensor value is detected, IRQ goes high that inhibits writing in the sensor RAM.

- For N-Key roll over mode, if the E bit is programmed to be '1', the 8279 operates in special Error mode. Details of this mode are described in scanned keyboard special error mode. **X**- don't care.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
1	1	1	E	X	X	X	X	1


INTR
8259 Interrupt controller
INT₂
Shift
Control
Return lines
Key board
8 columns
8 rows
8
AD₀-AD₁₅
A₁₆-A₁₉
8086
8282 (3) Address latch
A₀L-A₁₉L
INT
RL₀-₇
5V
V_DD
Vs
0V
S₀-₃
3-8 decoder
3
4
Scan lines
4-16
Data
8
D₀-₇
IOR
IOW
Reset
CS
A₀
CLK
8279
A₁L-A₃L
Address decoder
A₀L
BD
Blank display
16
B₀-₃
A₀-₃
4
Addres Display characte r data
4
RD
WR
PCLK (from 8284)

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.

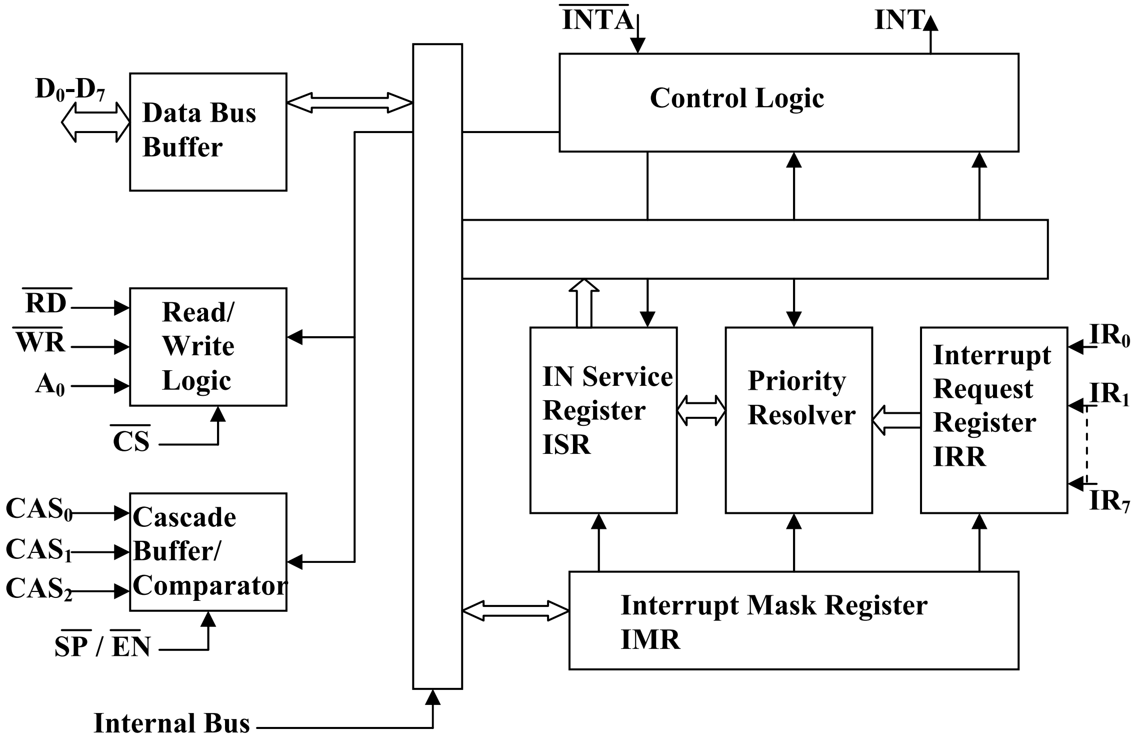


Fig:1 8259A Block Diagram

- **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.

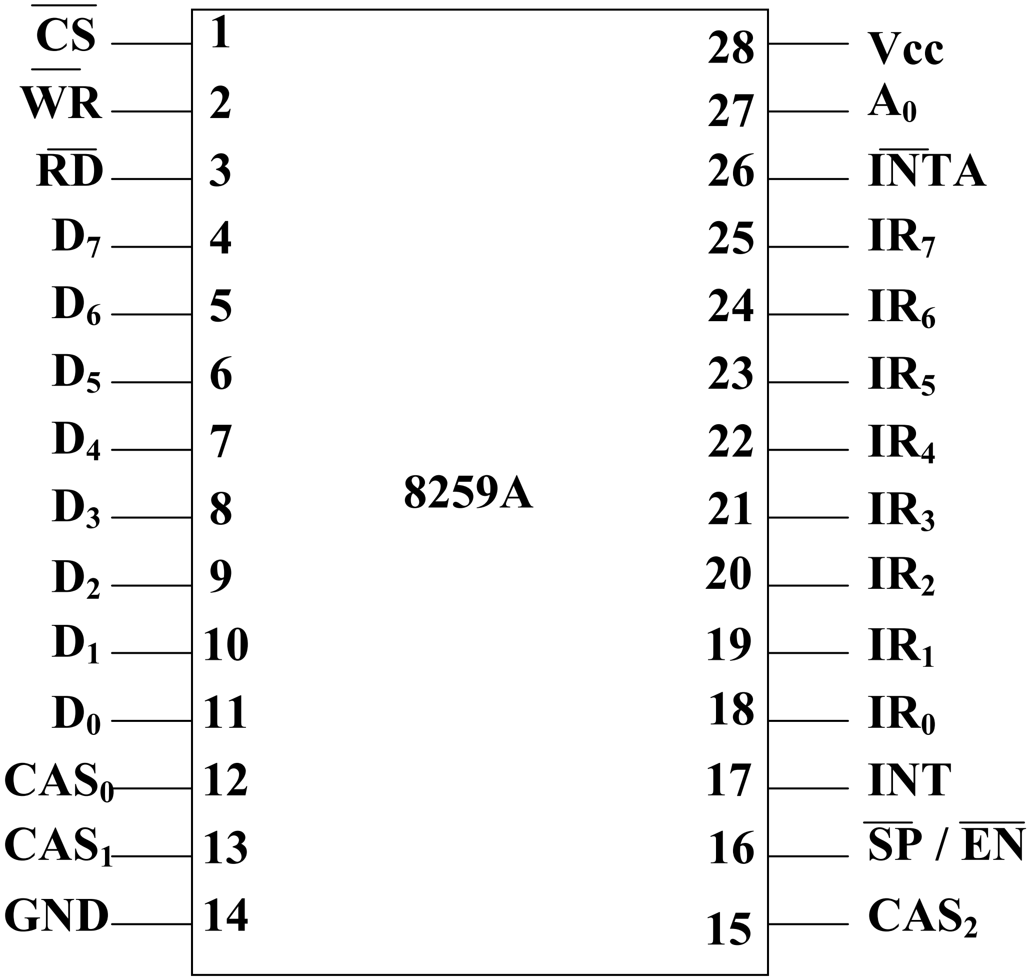


Fig : 8259 Pin Diagram

- **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 all the 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.

Interrupt Sequence in an 8086 system

- The Interrupt sequence in an 8086-8259A system is described as follows:

1. One or more IR lines are raised high that set corresponding IRR bits.
2. 8259A resolves priority and sends an INT signal to CPU.
3. The CPU acknowledge with INTA pulse.
4. Upon receiving an INTA signal from the CPU, the highest priority ISR bit is set and the corresponding IRR bit is reset. The 8259A does not drive data during this period.
5. The 8086 will initiate a second INTA pulse. During this period 8259A releases an 8-bit pointer on to a data bus from where it is read by the CPU.
6. This completes the interrupt cycle. The ISR bit is reset at the end of the second INTA pulse if automatic end of interrupt (AEOI) mode is programmed. Otherwise ISR bit remains set until an appropriate EOI command is issued at the end of interrupt subroutine.

Command Words of 8259A

- The command words of 8259A are classified in two groups

1. Initialization command words (ICW) and
2. Operation command words (OCW).

- Initialization Command Words (ICW): Before it starts functioning, the 8259A must be initialized by writing two to four command words into the respective command word registers. These are called as initialized command words.
- If A0 = 0 and D4 = 1, the control word is recognized as ICW1. It contains the control bits for edge/level triggered mode, single/cascade mode, call address interval and whether ICW4 is required or not.
- If A0=1, the control word is recognized as ICW2. The ICW2 stores details regarding interrupt vector addresses. The initialisation sequence of 8259A is described in form of a flow chart in fig 3 below.
- The bit functions of the ICW1 and ICW2 are self explanatory as shown in fig below.

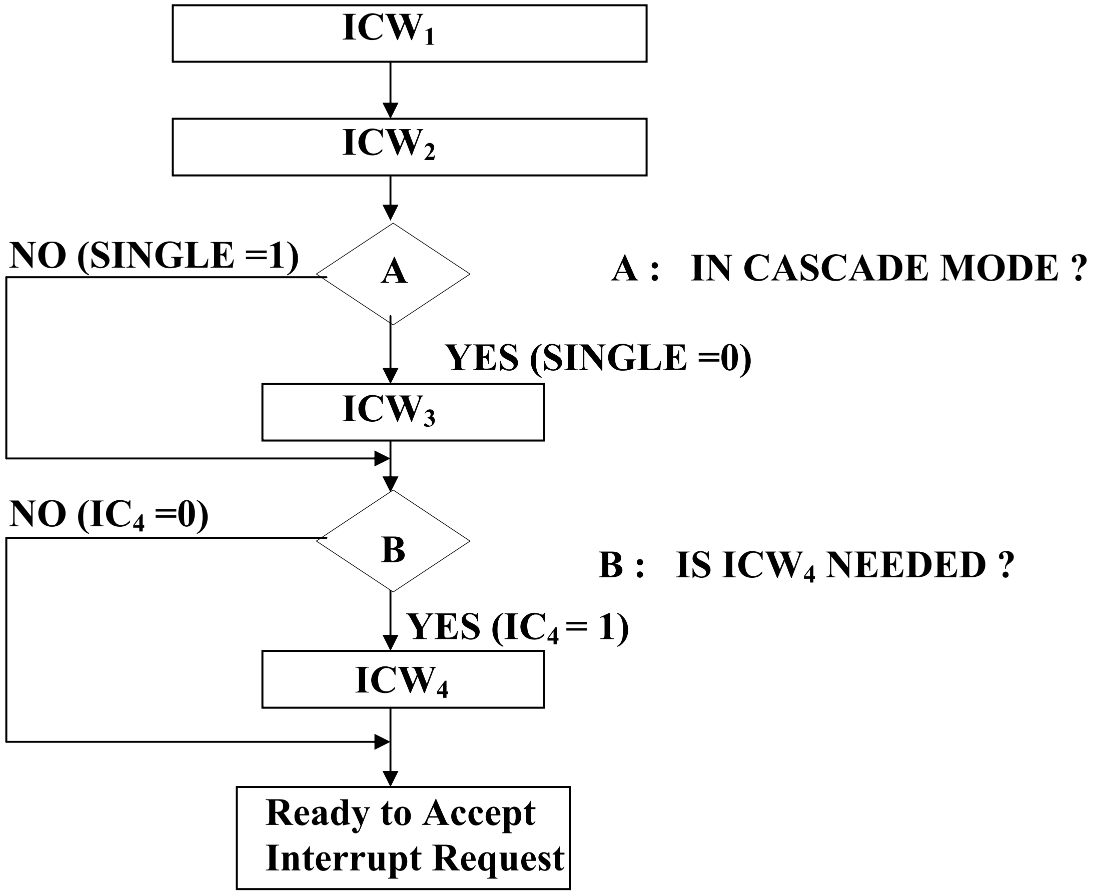
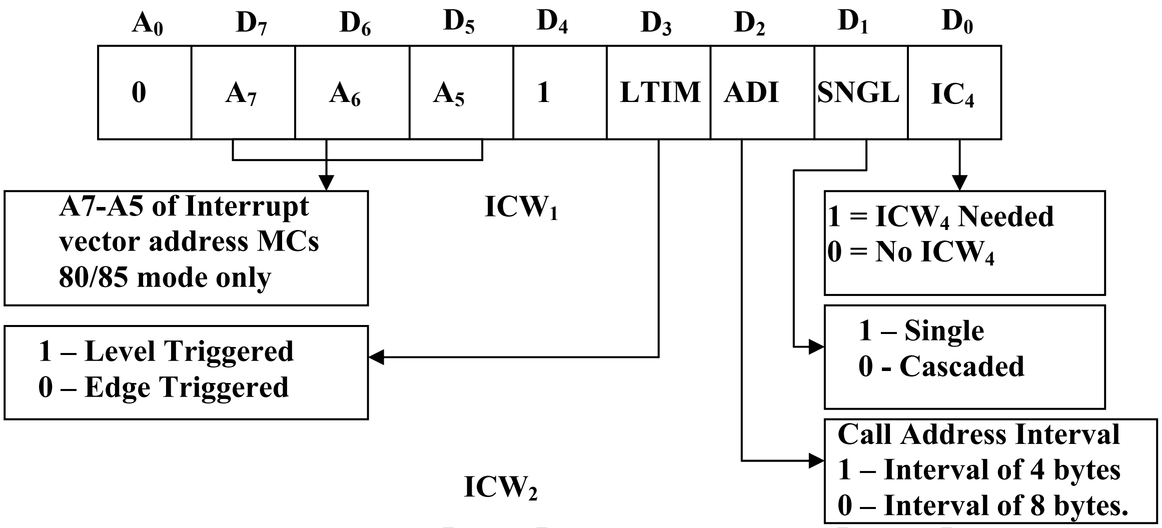


Fig 3: Initialisation Sequence of 8259A



ICW$_2$

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	T_7	T_6	T_5	T_4	T_3	A_{10}	A_9	A_8

- **T_7 – T3 are A3 – A0 of interrupt address**
- **A_{10} – A_9, A_8 – Selected according to interrupt request level. They are not the address lines of Microprocessor**
- **A0 =1 selects ICW$_2$**

Fig 4 : Instruction Command Words ICW$_1$ and ICW$_2$

- Once ICW1 is loaded, the following initialization procedure is carried out internally.

a. The edge sense circuit is reset, i.e. by default 8259A interrupts are edge sensitive.
b. IMR is cleared.
c. IR7 input is assigned the lowest priority.
d. Slave mode address is set to 7.
e. Special mask mode is cleared and status read is set to IRR.
f. If IC4 = 0, all the functions of ICW4 are set to zero. Master/Slave bit in ICW4 is used in the buffered mode only.
g. In an 8085 based system A15-A8 of the interrupt vector address are the respective bits of ICW2.
h. In 8086 based system A15-A11 of the interrupt vector address are inserted in place of T7 – T3 respectively and the remaining three bits A8, A9, A10 are selected depending upon the interrupt level, i.e. from 000 to 111 for IR0 to IR7.
i. ICW1 and ICW2 are compulsory command words in initialization sequence of 8259A as is evident from fig, while ICW3 and ICW4 are optional. The ICW3 is read only when there are more than one 8259A in the system, cascading is used (SNGL=0).
j. The SNGL bit in ICW1 indicates whether the 8259A in the cascade mode or not. The ICW3 loads an 8-bit slave register. It detailed functions are as follows.

k. In master mode [SP = 1 or in buffer mode M/S = 1 in ICW4], the 8-bit slave register will be set bit-wise to 1 for each slave in the system as in fig 5.

l. The requesting slave will then release the second byte of a CALL sequence. In slave mode [SP=0 or if BUF =1 and M/S = 0 in ICW4] bits D2 to D0 identify the slave, i.e. 000 to 111 for slave 1 to slave 8. The slave compares the cascade inputs with these bits and if they are equal, the second byte of the CALL sequence is released by it on the data bus.

Master mode ICW_3

A_0	**D_7**	**D_6**	**D_5**	**D_4**	**D_3**	**D_2**	**D_1**	**D_0**
1	**S_7**	**S_6**	**S_5**	**S_4**	**S_3**	**S_2**	**S_1**	**S_0**

Sn = 1-IRn Input has a slave
= 0 – IRn Input does not have a slave

Slave mode ICW_3

A_0	**D_7**	**D_6**	**D_5**	**D_4**	**D_3**	**D_2**	**D_1**	**D_0**
1	**0**	**0**	**0**	**0**	**0**	**ID_2**	**ID_1**	**ID_0**

$D_2D_1D_0$ – 000 to 111 for IR_0 to IR_7 or slave 1 to slave 8

ICW_4

A_0	**D_7**	**D_6**	**D_5**	**D_4**	**D_3**	**D_2**	**D_1**	**D_0**
1	**0**	**0**	**0**	**SFNM**	**BUF**	**M/S**	**AEOI**	**μPM**

Fig : ICW_3 in Master and Slave Mode, ICW_4 Bit Functions

- **ICW4**: The use of this command word depends on the IC4 bit of ICW1. If IC4=1, IC4 is used, otherwise it is neglected. The bit functions of ICW4 are described as follow:
- **SFNM**: If BUF = 1, the buffered mode is selected. In the buffered mode, SP/EN acts as enable output and the master/slave is determined using the M/S bit of ICW4.
- **M/S**: If M/S = 1, 8259A is a master. If M/S =0, 8259A is slave. If BUF = 0, M/S is to be neglected.
- **AEOI**: If AEOI = 1, the automatic end of interrupt mode is selected.

- **µPM** : If the µPM bit is 0, the Mcs-85 system operation is selected and if µPM=1, 8086/88 operation is selected.
- **Operation Command Words:** Once 8259A is initialized using the previously discussed command words for initialisation, it is ready for its normal function, i.e. for accepting the interrupts but 8259A has its own way of handling the received interrupts called as modes of operation. These modes of operations can be selected by programming, i.e. writing three internal registers called as operation command words.
- In the three operation command words OCW1, OCW2 and OCW3 every bit corresponds to some operational feature of the mode selected, except for a few bits those are either 1 or 0. The three operation command words are shown in fig with the bit selection details.
- OCW1 is used to mask the masked and if it is 0 the request is enabled. In OCW2 the three bits, R, SL and EOI control the end of interrupt, the rotate mode and their combinations as shown in fig below.
- The three bits L2, L1 and L0 in OCW2 determine the interrupt level to be selected for operation, if SL bit is active i.e. 1.
- The details of OCW2 are shown in fig.
- In operation command word 3 (OCW3), if the ESMM bit, i.e. enable special mask mode bit is set to 1, the SMM bit is neglected. If the SMM bit, i.e. special mask mode. When ESMM bit is 0 the SMM bit is neglected. If the SMM bit. i.e. special mask mode bit is 1, the 8259A will enter special mask mode provided ESMM=1.
- If ESMM=1 and SMM=0, the 8259A will return to the normal mask mode. The details of bits of OCW3 are given in fig along with their bit definitions.

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	M_7	M_6	M_5	M_4	M_3	M_2	M_1	M_0

1 – Mask Set
0 – Mask Reset

Fig (a) : OCW_1

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	ESMM	SMM	0	1	P	RR	RIS

Fig (b) :

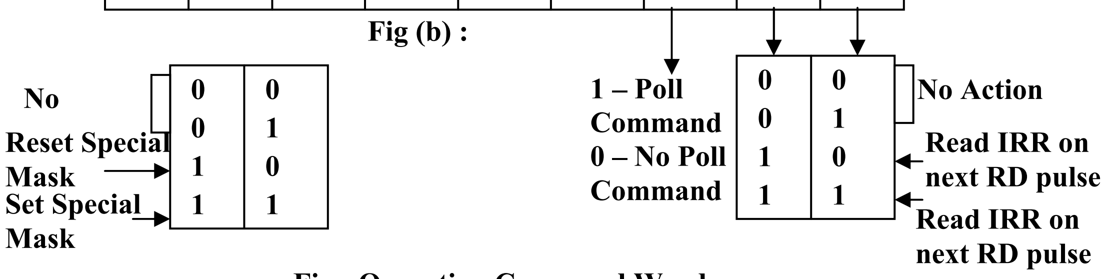
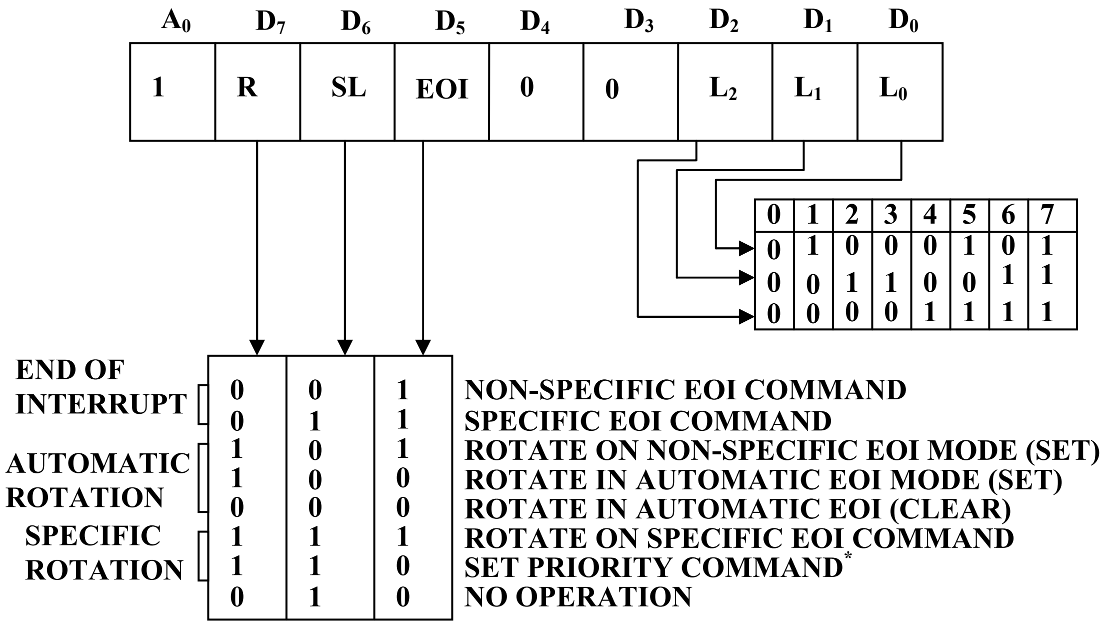


Fig : Operation Command Words

Fig (c) :OCW_2



A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	R	SL	EOI	0	0	L_2	L_1	L_0

0	1	2	3	4	5	6	7
0	1	0	0	0	1	0	1
0	0	1	1	0	0	1	1
0	0	0	0	1	1	1	1

END OF INTERRUPT	0	0	1	NON-SPECIFIC EOI COMMAND
	0	1	1	SPECIFIC EOI COMMAND
AUTOMATIC ROTATION	1	0	1	ROTATE ON NON-SPECIFIC EOI MODE (SET)
	1	0	0	ROTATE IN AUTOMATIC EOI MODE (SET)
	0	0	0	ROTATE IN AUTOMATIC EOI (CLEAR)
SPECIFIC ROTATION	1	1	1	ROTATE ON SPECIFIC EOI COMMAND
	1	1	0	SET PRIORITY COMMAND*
	0	1	0	NO OPERATION

* - In this Mode $L_0 – L_2$ are used

Fig : Operation Command Word

Operating Modes of 8259

- The different modes of operation of 8259A can be programmed by setting or resting the appropriate bits of the ICW or OCW as discussed previously. The different modes of operation of 8259A are explained in the following.
- **Fully Nested Mode** : This is the default mode of operation of 8259A. IR0 has the highest priority and IR7 has the lowest one. When interrupt request are noticed, the highest priority request amongst them is determined and the vector is placed on the data bus. The corresponding bit of ISR is set and remains set till the microprocessor issues an EOI command just before returning from the service routine or the AEOI bit is set.
- If the ISR (in service) bit is set, all the same or lower priority interrupts are inhibited but higher levels will generate an interrupt, that will be acknowledge only if the microprocessor interrupt enable flag IF is set. The priorities can afterwards be changed by programming the rotating priority modes.
- **End of Interrupt (EOI)** : The ISR bit can be reset either with AEOI bit of ICW1 or by EOI command, issued before returning from the interrupt service routine. There are two types of EOI commands specific and non-specific. When 8259A is operated in the modes that preserve fully nested structure, it can determine which ISR bit is to be reset on EOI.
- When non-specific EOI command is issued to 8259A it will be automatically reset the highest ISR bit out of those already set.

- When a mode that may disturb the fully nested structure is used, the 8259A is no longer able to determine the last level acknowledged. In this case a specific EOI command is issued to reset a particular ISR bit. An ISR bit that is masked by the corresponding IMR bit, will not be cleared by non-specific EOI of 8259A, if it is in special mask mode.
- **Automatic Rotation** : This is used in the applications where all the interrupting devices are of equal priority.
- In this mode, an interrupt request IR level receives priority after it is served while the next device to be served gets the highest priority in sequence. Once all the device are served like this, the first device again receives highest priority.
- **Automatic EOI Mode** : Till AEOI=1 in ICW4, the 8259A operates in AEOI mode. In this mode, the 8259A performs a non-specific EOI operation at the trailing edge of the last INTA pulse automatically. This mode should be used only when a nested multilevel interrupt structure is not required with a single 8259A.
- **Specific Rotation** : In this mode a bottom priority level can be selected, using L2, L1 and L0 in OCW2 and R=1, SL=1, EOI=0.
- The selected bottom priority fixes other priorities. If IR5 is selected as a bottom priority, then IR5 will have least priority and IR4 will have a next higher priority. Thus IR6 will have the highest priority.
- These priorities can be changed during an EOI command by programming the rotate on specific EOI command in OCW2.
- **Specific Mask Mode**: In specific mask mode, when a mask bit is set in OCW1, it inhibits further interrupts at that level and enables interrupt from other levels, which are not masked.
- **Edge and Level Triggered Mode** : This mode decides whether the interrupt should be edge triggered or level triggered. If bit LTIM of ICW1 =0 they are edge triggered, otherwise the interrupts are level triggered.
- **Reading 8259 Status** : The status of the internal registers of 8259A can be read using this mode. The OCW3 is used to read IRR and ISR while OCW1 is used to read IMR. Reading is possible only in no polled mode.
- **Poll Command** : In polled mode of operation, the INT output of 8259A is neglected, though it functions normally, by not connecting INT output or by masking INT input of the microprocessor. The poll mode is entered by setting P=1 in OCW3.
- The 8259A is polled by using software execution by microprocessor instead of the requests on INT input. The 8259A treats the next RD pulse to the 8259A as an interrupt acknowledge. An appropriate ISR bit is set, if there is a request. The priority level is read and a data word is placed on to data bus, after RD is activated. A poll command may give more than 64 priority levels.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	x	x	x	x	w_2	w_1	w_0

If = 1, there is an interrupt

Binary code of highest priority level

Fig : Data Word of 8259

- **Special Fully Nested Mode** : This mode is used in more complicated system, where cascading is used and the priority has to be programmed in the master using ICW4. this is somewhat similar to the normal nested mode.
- In this mode, when an interrupt request from a certain slave is in service, this slave can further send request to the master, if the requesting device connected to the slave has higher priority than the one being currently served. In this mode, the master interrupt the CPU only when the interrupting device has a higher or the same priority than the one current being served. In normal mode, other requests than the one being served are masked out.
- When entering the interrupt service routine the software has to check whether this is the only request from the slave. This is done by sending a non-specific EOI can be sent to the master, otherwise no EOI should be sent. This mode is important, since in the absence of this mode, the slave would interrupt the master only once and hence the priorities of the slave inputs would have been disturbed.
- **Buffered Mode**: When the 83259A is used in the systems where bus driving buffers are used on data buses. The problem of enabling the buffers exists. The 8259A sends buffer enable signal on SP/ EN pin, whenever data is placed on the bus.
- **Cascade Mode** : The 8259A can be connected in a system containing one master and eight slaves (maximum) to handle upto 64 priority levels. The master controls the slaves using CAS0-CAS2 which act as chip select inputs (encoded) for slaves.
- In this mode, the slave INT outputs are connected with master IR inputs. When a slave request line is activated and acknowledged, the master will enable the slave to release the vector address during second pulse of INTA sequence.
- The cascade lines are normally low and contain slave address codes from the trailing edge of the first INTA pulse to the trailing edge of the second INTA pulse. Each 8259A in the system must be separately initialized and programmed to work in different modes. The EOI command must be issued twice, one for master and the other for the slave.
- A separate address decoder is used to activate the chip select line of each 8259A.
- Following Fig shows the details of the circuit connections of 8259A in cascade scheme.

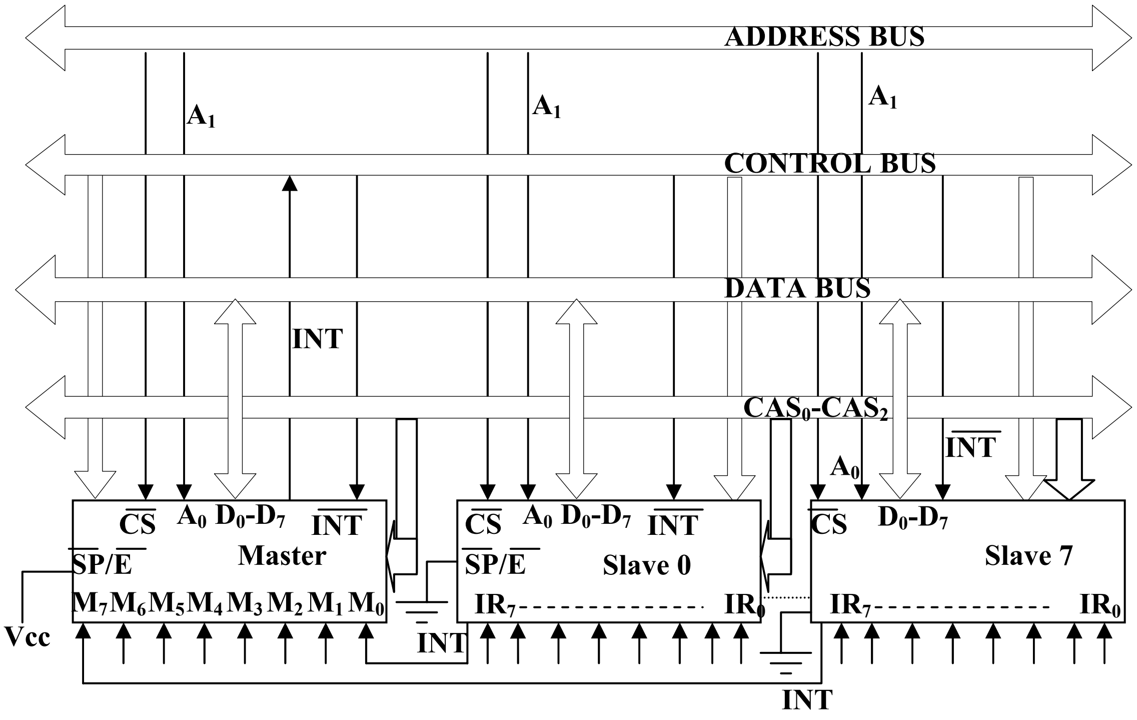


Fig : 8259A in Cascade Mode

Interfacing a Microprocessor To Keyboard

- When you press a key on your computer, you are activating a switch. There are many different ways of making these switches. An overview of the construction and operation of some of the most common types.

1. ***Mechanical key switches:*** In mechanical-switch keys, two pieces of metal are pushed together when you press the key. The actual switch elements are often made of a phosphor-bronze alloy with gold platting on the contact areas. The key switch usually contains a spring to return the key to the nonpressed position and perhaps a small piece of foam to help damp out bouncing.
2. Some mechanical key switches now consist of a molded silicon dome with a small piece of conductive rubber foam short two trace on the printed-circuit board to produce the key pressed signal.
3. Mechanical switches are relatively inexpensive but they have several disadvantages. First, they suffer from contact bounce. A pressed key may make and break contact several times before it makes solid contact.
4. Second, the contacts may become oxidized or dirty with age so they no longer make a dependable connection.

- Higher-quality mechanical switches typically have a rated life time of about 1 million keystrokes. The silicone dome type typically last 25 million keystrokes.

2. ***Membrane key switches:*** These switches are really a special type of mechanical switches. They consist of a three-layer plastic or rubber sandwich.

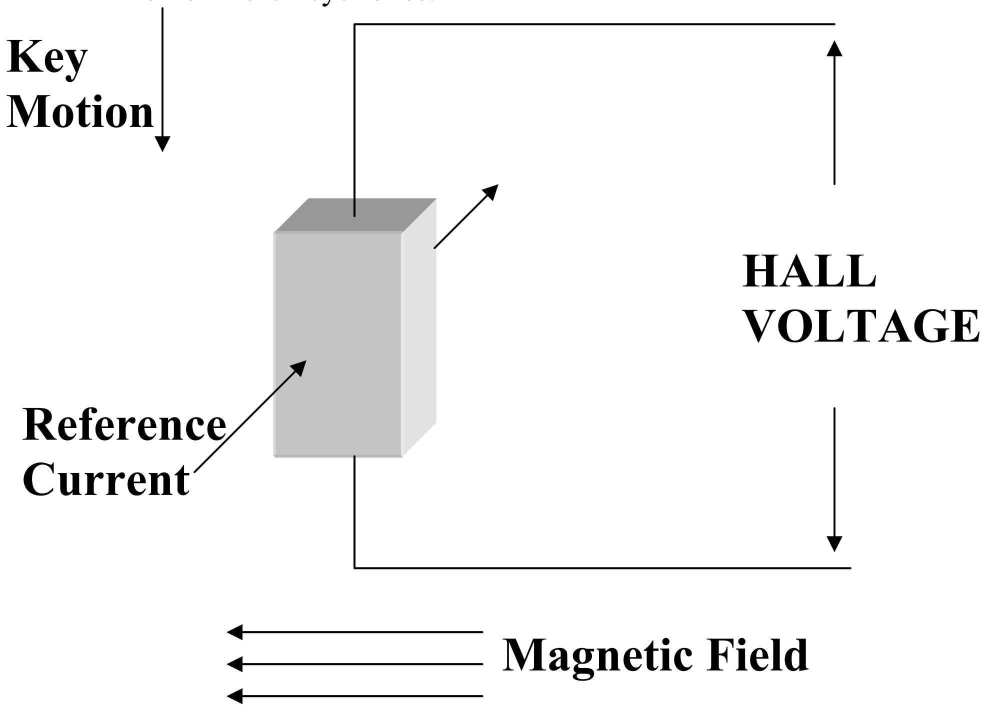
- The top layer has a conductive line of silver ink running under each key position. The bottom layer has a conductive line of silver ink running under each column of keys.
- When u press a key, you push the top ink line through the hole to contact the bottom ink line.
- The advantages of membrane keyboards is that they can be made as very thin, sealed units.
- They are often used on cash registers in fast food restaurants. The lifetime of membrane keyboards varies over a wide range.

3. Capacitive key switches: A capacitive keyswitch has two small metal plates on the printed circuit board and another metal plate on the bottom of a piece of foam.

- When u press the key, the movable plate is pushed closer to fixed plate. This changes the capacitance between the fixed plates. Sense amplifier circuitry detects this change in capacitance and produce a logic level signal that indicates a key has been pressed.
- The big advantages of a capacitive switch is that it has no mechanical contacts to become oxidized or dirty.
- A small disadvantage is the specified circuitry needed to detect the change in capacitance.
- Capacitive keyswitches typically have a rated lifetime of about 20 million keystrokes.

4. ***Hall effect keyswitches:*** This is another type of switch which has no mechanical contact. It takes advantage of the deflection of a moving charge by a magnetic field.

- A reference current is passed through a semiconductor crystal between two opposing faces. When a key is pressed, the crystal is moved through a magnetic field which has its flux lines perpendicular to the direction of current flow in the crystal.
- Moving the crystal through the magnetic field causes a small voltage to be developed between two of the other opposing faces of the crystal.
- This voltage is amplified and used to indicate that a key has been pressed. Hall effect sensors are also used to detect motion in many electrically controlled machines.
- Hall effect keyboards are more expensive because of the more complex switch mechanism, but they are very dependable and have typically rated lifetime of 100 million or more keystrokes.



HALL EFFECT

Keyboard Circuit Connections and Interfacing

- In most keyboards, the keyswitches are connecting in a matrix of rows and columns, as shown in fig.
- We will use simple mechanical switches for our examples, but the principle is same for other type of switches.
- Getting meaningful data from a keyboard, it requires the following three major tasks:

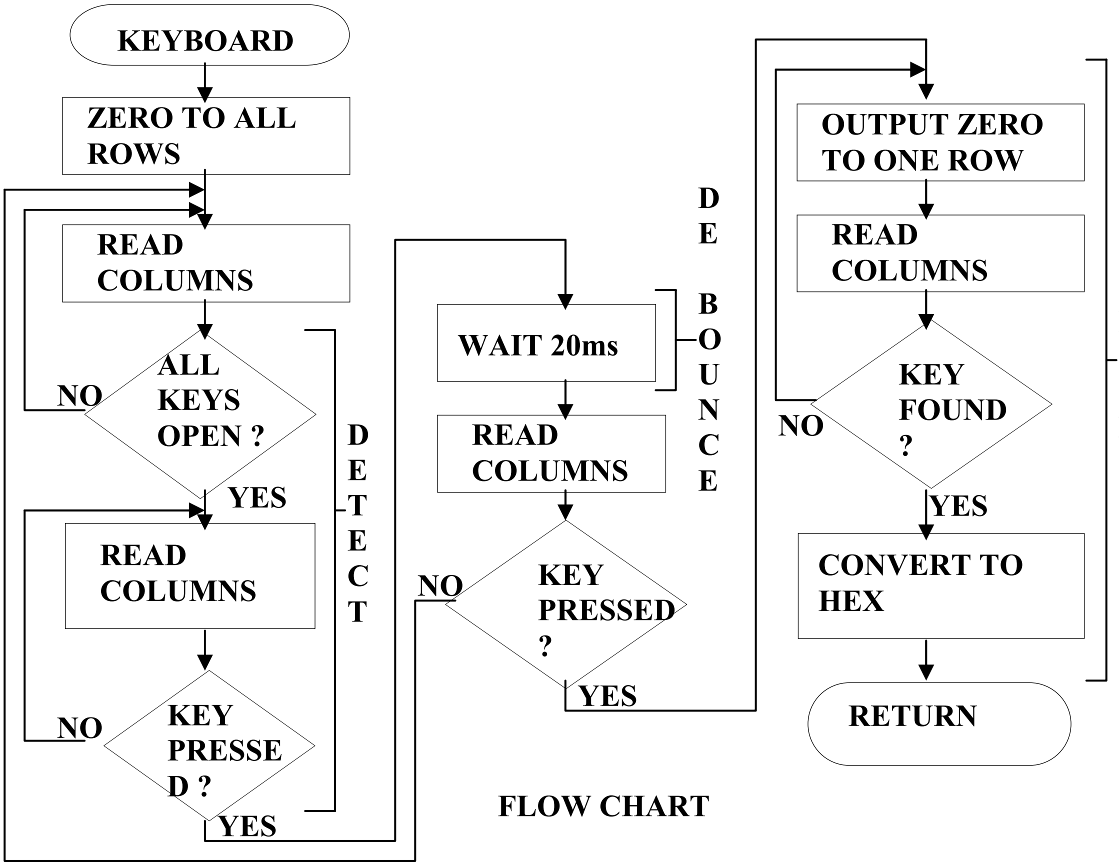
1. Detect a keypress.

2. Debounce the keypress.
3. Encode the keypress

- Three tasks can be done with hardware, software, or a combination of two, depending on the application.

1. **Software Keyboard Interfacing:**

- ***Circuit connection and algorithm :*** The following fig (a) shows how a hexadecimal keypad can be connected to a couple of microcomputer ports so the three interfacing tasks can be done as part of a program.
- The rows of the matrix are connected to four output port lines. The column lines of matrix are connected to four input-port lines. To make the program simpler, the row lines are also connected to four input lines.
- When no keys are pressed, the column lines are held high by the pull-up resistor connected to +5V. Pressing a key connects a row to a column. If a low is output on a row and a key in that row is pressed, then the low will appear on the column which contains that key and can be detected on the input port.
- If you know the row and column of the pressed key, you then know which key was pressed, and you can convert this information into any code you want to represent that key.
- The following flow chart for a procedure to detect, debounce and produce the hex code for a pressed key.
- An easy way to detect if any key in the matrix is pressed is to output 0’s to all rows and then check the column to see if a pressed key has connected a low to a column.
- In the algorithm we first output lows to all the rows and check the columns over and over until the column are all high. This is done before the previous key has been released before looking for the next one. In the standard keyboard terminology, this is called two-key lockout.



FLOW CHART

- Once the columns are found to be all high, the program enters another loop, which waits until a low appears on one of the columns, indicating that a key has been pressed. This second loop does the detect task for us. A simple 20-ms delay procedure then does the debounce task.
- After the debounce time, another check is made to see if the key is still pressed. If the columns are now all high, then no key is pressed and the initial detection was caused by a noise pulse or a light brushing past a key. If any of the columns are still low, then the assumption is made that it was a valid keypress.
- The final task is to determine the row and column of the pressed key and convert this row and column information to the hex code for the pressed key. To get the row and column information, a low is output to one row and the column are read. If none of the columns is low, the pressed key is not in that row. So the low is rotated to the next row and the column are checked again. The process is repeated until a low on a row produces a low on one of the column.
- The pressed key then is in the row which is low at that time.
- The connection fig shows the byte read in from the input port will contain a 4-bit code which represents the row of the pressed key and a 4-bit code which represent the column of the pressed key.
- ***Error trapping:*** The concept of detecting some error condition such as “ no match found” is called error trapping. Error trapping is a very important part of real programs. Even in simple programs, think what might happen with no error trap if

two keys in the same row were pressed at exactly at the same time and a column code with two lows in it was produced.

- This code would not match any of the row-column codes in the table, so after all the values in the table were checked, assigned register in program would be decremented from 0000H to FFFFH. The compare decrement cycle would continue through 65,536 memory locations until, by change the value in a memory location matched the row-column code. The contents of the lower byte register at hat point would be passed back to the calling routine. The changes are 1 in 256 that would be the correct value for one of the pressed keys. You should keep an error trap in a program whenever there is a chance for it.

2. **Keyboard Interfacing with Hardware:** For the system where the CPU is too busy to be bothered doing these tasks in software, an external device is used to do them.

- One of a MOS device which can be do this is the General Instruments AY5-2376 which can be connected to the rows and columns of a keyboard switch matrix.
- The AY5-2376 independently detects a keypress by cycling a low down through the rows and checking the columns. When it finds a key pressed, it waits a debounce time.
- If the key is still pressed after the debounce time, the AY5-2376 produces the 8-bit code for the pressed key and send it out to microcomputer port on 8 parallel lines. The microcomputer knows that a valid ASCII code is on the data lines, the AY5-2376 outputs a strobe pulse.
- The microcomputer can detect this strobe pulse and read in ASCII code on a polled basis or it can detect the strobe pulse on an interrupt basis.
- With the interrupt method the microcomputer doesn’t have to pay any attention to the keyboard until it receives an interrupt signal.
- So this method uses very little of the microcomputer time. The AY5-2376 has a feature called *two-key rollover*. This means that if two keys are pressed at nearly the same time, each key will be detected, debounced and converted to ASCII.
- The ASCII code for the first key and a strobe signal for it will be sent out then the ASCII code for the second key and a strobe signal for it will be sent out and compare this with two-key lockout.

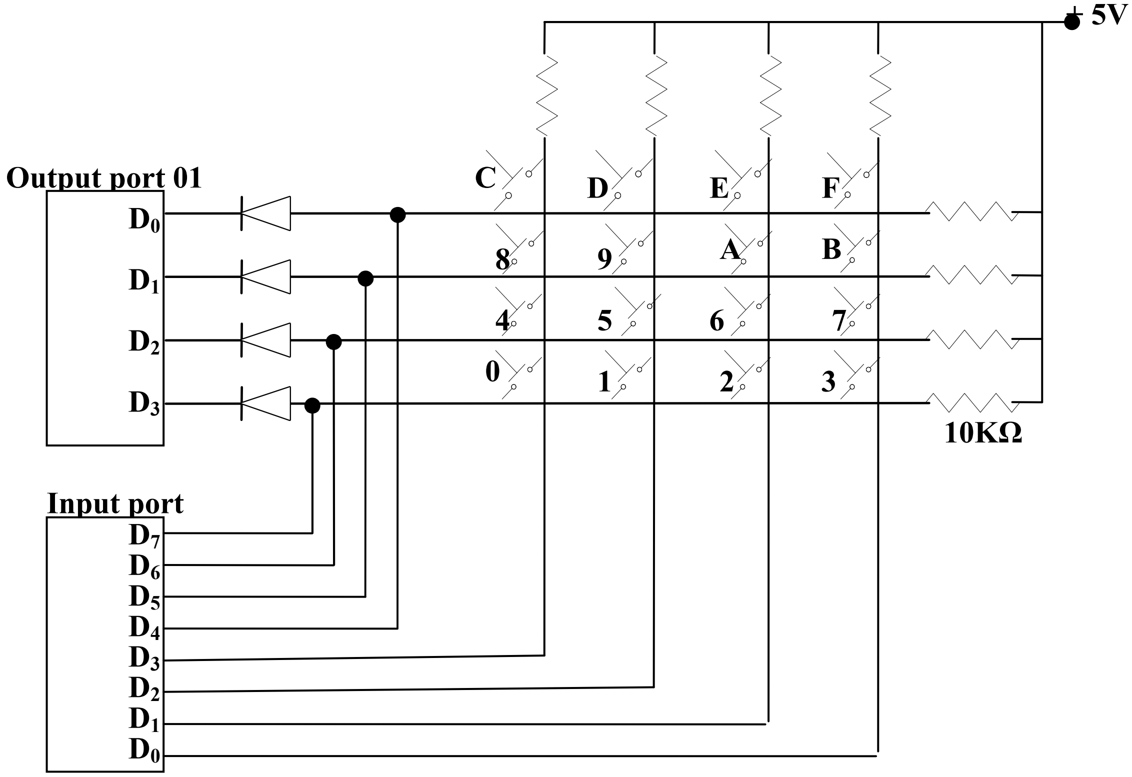
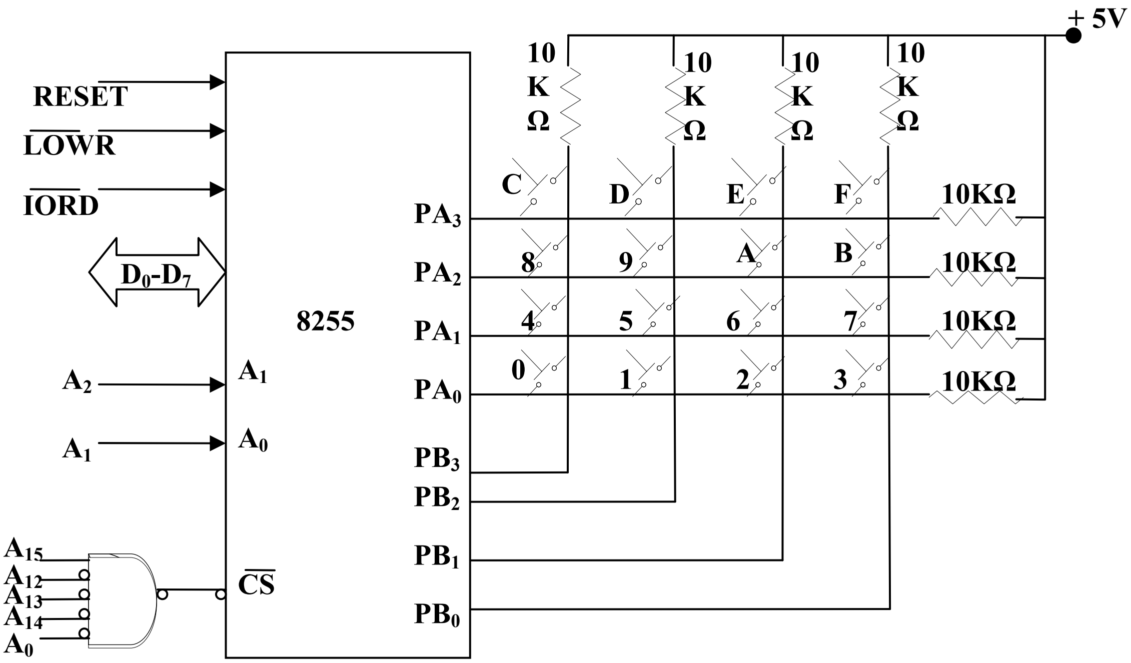


Fig: (a) Port connections

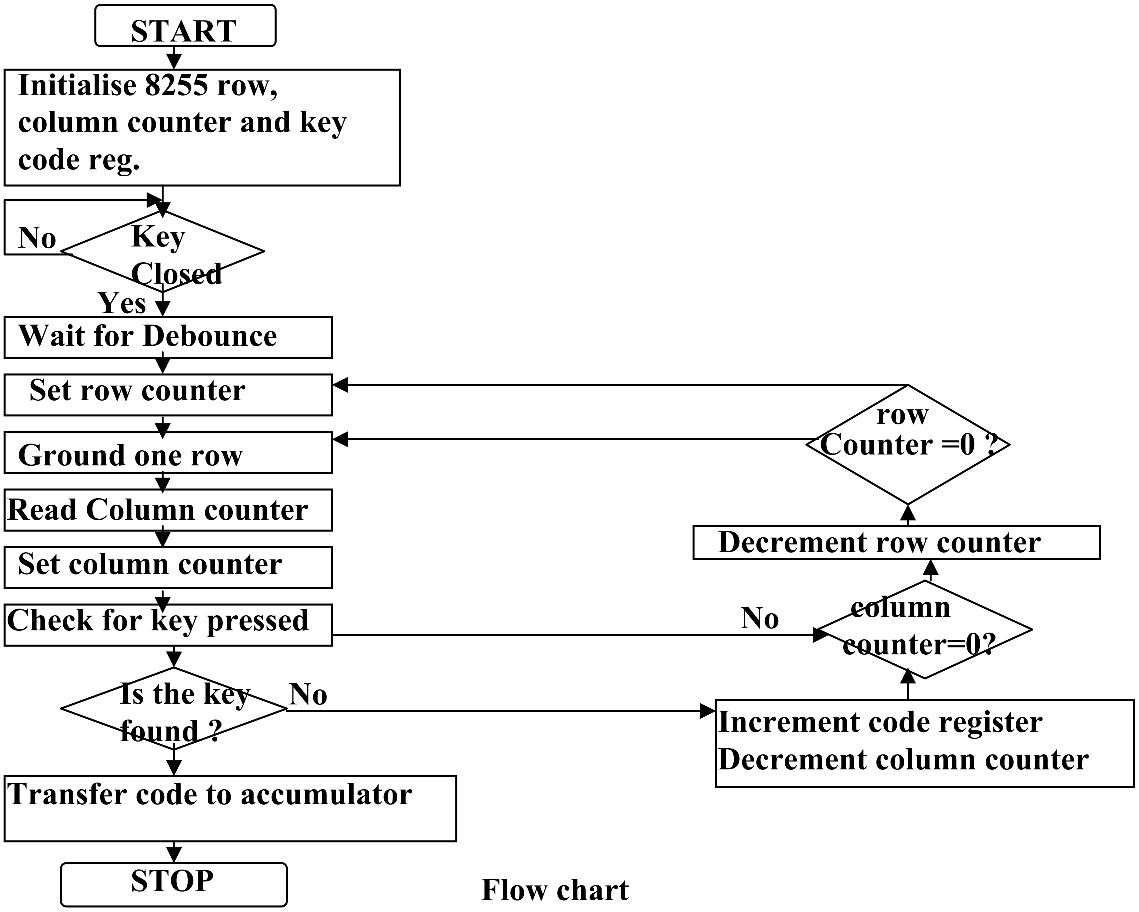
Example

- Interface a 4 * 4 keyboard with 8086 using 8255 an write an ALP for detecting a key closure and return the key code in AL. The debounce period for a key is 10ms. Use software debouncing technique. DEBOUNCE is an available 10ms delay routine.
- Solution: Port A is used as output port for selecting a row of keys while Port B is used as an input port for sensing a closed key. Thus the keyboard lines are selected one by one through port A and the port B lines are polled continuously till a key closure is sensed. The routine DEBOUNCE is called for key debouncing. The key code is depending upon the selected row and a low sensed column.



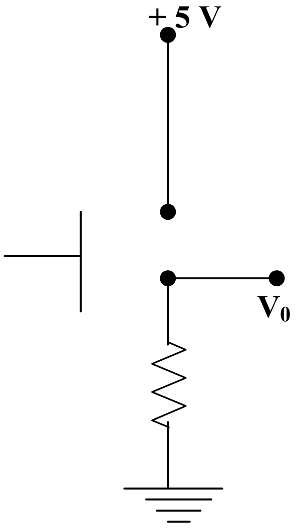
Interfacing 4 * 4 Keyboard

- The higher order lines of port A and port B are left unused. The address of port A and port B will respectively 8000H and 8002H while address of CWR will be 8006H. The flow chart of the complete program is as given. The control word for this problem will be 82H. Code segment CS is used for storing the program code.
- **Key Debounce** : Whenever a mechanical push-button is pressed or released once, the mechanical components of the key do not change the position smoothly, rather it generates a transient response .

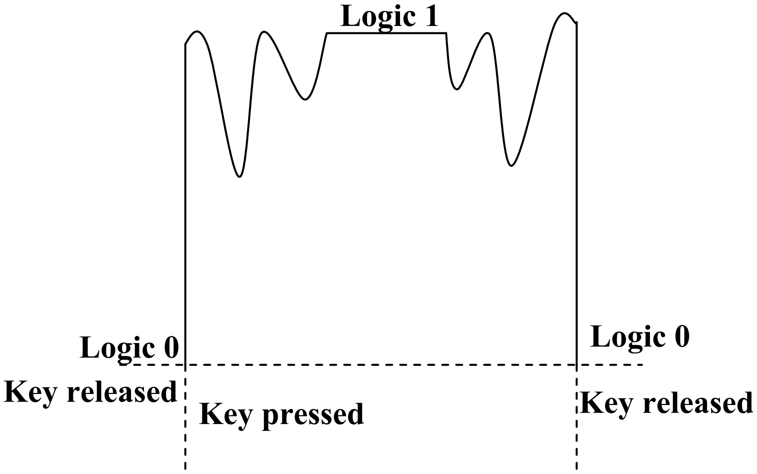


Flow chart

- These transient variations may be interpreted as the multiple key pressure and responded accordingly by the microprocessor system.
- To avoid this problem, two schemes are suggested: the first one utilizes a bistable multivibrator at the output of the key to debounce .
- The other scheme suggests that the microprocessor should be made to wait for the transient period (usually 10ms), so that the transient response settles down and reaches a steady state.
- A logic ‘0’ will be read by the microprocessor when the key is pressed.
- In a number of high precision applications, a designer may have two options- the first is to have more than one 8-bit port, read (write) the port one by one and then from the multibyte data, the second option allows forming 16-bit ports using two 8-bit ports and use 16-bit read or write operations.



A Mechanical Key



Response

Interfacing To Alphanumeric Displays

- To give directions or data values to users, many microprocessor-controlled instruments and machines need to display letters of the alphabet and numbers. In systems where a large amount of data needs to be displayed a CRT is used to display the data. In system where only a small amount of data needs to be displayed, simple digit-type displays are often used.
- There are several technologies used to make these digit-oriented displays but we are discussing only the two major types.
- These are *light emitting diodes* (LED) and *liquid-crystal displays* (LCD).
- LCD displays use very low power, so they are often used in portable, battery-powered instruments. They do not emit their own light, they simply change the reflection of available light. Therefore, for an instrument that is to be used in low-light conditions, you have to include a light source for LCDs or use LEDs which emit their own light.
- Alphanumeric LED displays are available in three common formats. For displaying only number and hexadecimal letters, simple 7-segment displays such as that as shown in fig are used.
- To display numbers and the entire alphabet, 18 segment displays such as shown in fig or 5 by 7 dot-matrix displays such as that shown in fig can be used. The 7-segment type is the least expensive, most commonly used and easiest to interface with, so we will concentrate first on how to interface with this type.

1. ***Directly Driving LED Displays:*** Figure shows a circuit that you might connect to a parallel port on a microcomputer to drive a single 7-segment , common-anode display. For a common-anode display, a segment is tuned on by applying a logic low to it.

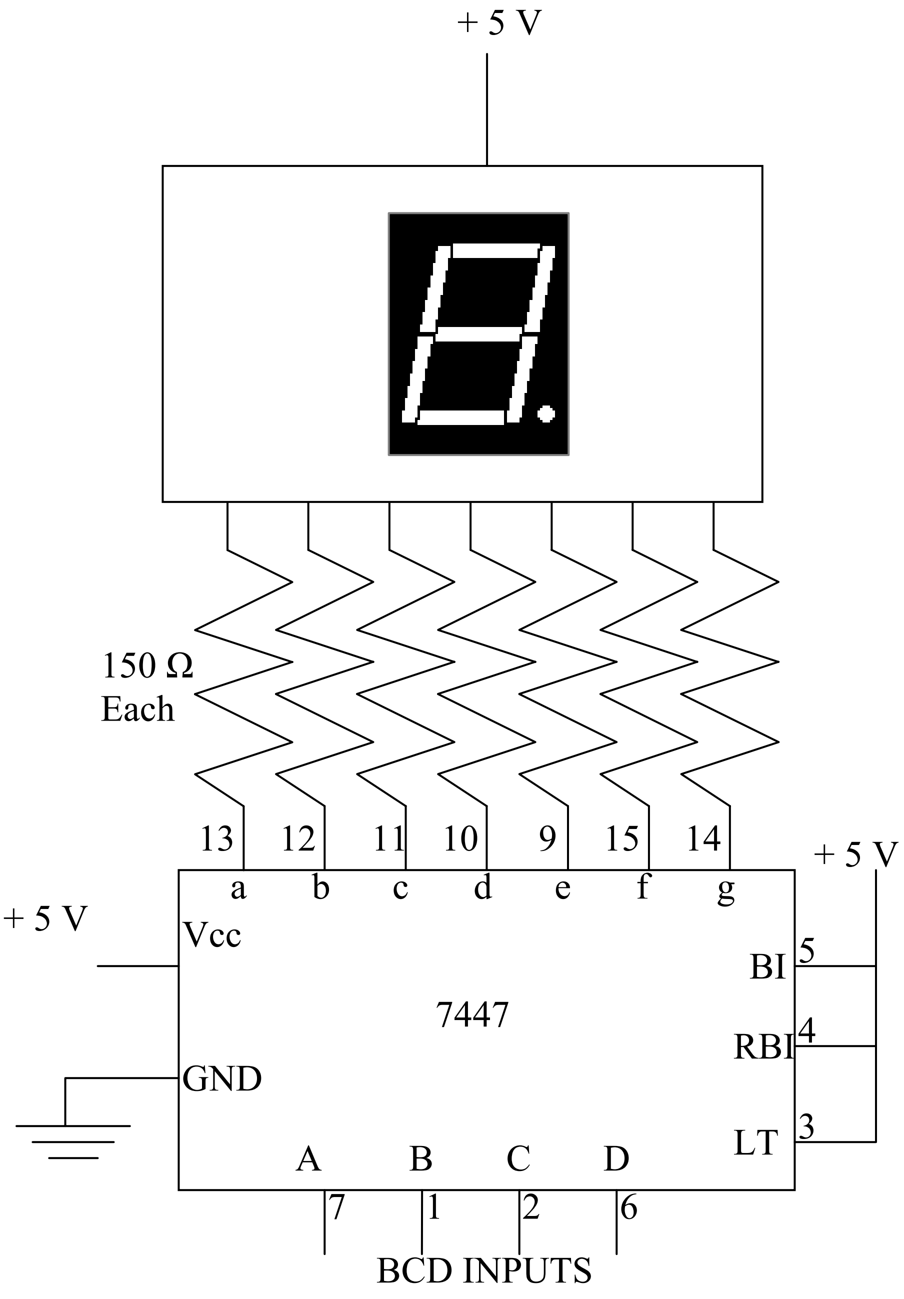
- The 7447 converts a BCD code applied to its inputs to the pattern of lows required to display the number represented by the BCD code. This circuit connection is referred to as a *static display* because current is being passed through the display at all times.
- Each segment requires a current of between 5 and 30mA to light. Let's assume you want a current of 20mA. The voltage drop across the LED when it is lit is about 1.5V.
- The output low voltage for the 7447 is a maximum of 0.4V at 40mA. So assume that it is about 0.2V at 20mA. Subtracting these two voltage drop from the supply voltage of 5V leaves 3.3V across the current limiting resistor. Dividing 3.3V by 20mA gives a value of 168Ω for the current-limiting resistor. The voltage drops across the LED and the output of 7447 are not exactly predictable and exact current through the LED is not critical as long as we don't exceed its maximum rating.

2. ***Software-Multiplexed LED Display:***

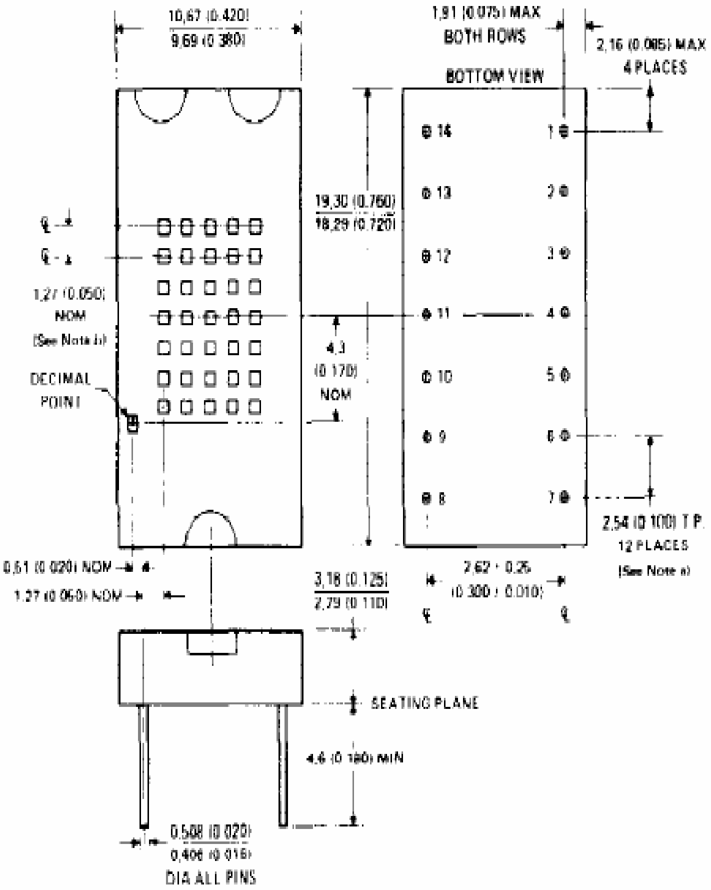
- The circuit in fig works for driving just one or two LED digits with a parallel output port. However, this scheme has several problem if you want to drive, eight digits.

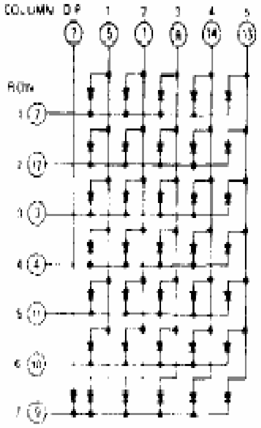
- The first problem is power consumption. For worst-case calculations, assume that all 8 digits are displaying the digit 8, so all 7 segments are all lit. Seven segment time 20mA per segment gives a current of 140mA per digit. Multiplying this by 8 digits gives a total current of 1120mA or 1.12A for 8 digits.
- A second problem of the static approach is that each display digit requires a separate 7447 decoder, each of which uses of another 13mA. The current required by the decoders and the LED displays might be several times the current required by the reset of the circuitry in the instrument.
- To solve the problem of the static display approach, we use a *multiplex method*, example for an explanation of the multiplexing.
- The fig shows a circuit you can add to a couple of microcomputer ports to drive some common anode LED displays in a multiplexed manner. The circuit has only one 7447 and that the segment outputs of the 7447 are bused in parallel to the segment inputs of all the digits.
- The question that may occur to you on first seeing this is: Aren't all the digits going to display the same number? The answer is that they would if all the digits were turned on at the same time. The tricky of multiplexing displays is that only one display digit is turned on at a time.
- The PNP transistor is series with the common anode of each digit acts as on/off switch for that digit. Here's how the multiplexing process works.
- The BCD code for digit 1 is first output from port B to the 7447. the 7447 outputs the corresponding 7-segment code on the segment bus lines. The transistor connected to digit 1 is then turned on by outputting a low to the appropriate bit of port A. All the rest of the bits of port A are made high to make sure no other digits are turned on. After 1 or 2 ms, digit 1 is turned off by outputting all highs to port A.
- The BCD code for digit 2 is then output to the 7447 on port B, and a word to turn on digit 2 is output on port A.
- After 1 or 2 ms, digit 2 is turned off and the process is repeated for digit 3. the process is continued until all the digits have had a turn. Then digit 1 and the following digits are lit again in turn.
- A procedure which is called on an interrupt basis every 2ms to keep these displays refreshed wit some values stored in a table. With 8 digits and 2ms per digit, you get back to digit 1 every 16ms or about 60 times a second.
- This refresh rate is fast enough so that the digits will each appear to be lit all time. Refresh rates of 40 to 200 times a second are acceptable.
- The immediately obvious advantages of multiplexing the displays are that only one 7447 is required, and only one digit is lit at a time. We usually increase the current per segment to between 40 and 60 mA for multiplexed displays so that they will appear as bright as they would if they were not multiplexed. Even with this increased segment current, multiplexing gives a large saving in power and parts.
- The software-multiplexed approach we have just described can also be used to drive 18-segment LED devices and dot-matrix LED device. For these devices, however you replace the 7447 in fig with ROM which generates the required

segment codes when the ASCII code for a character is applied to the address inputs of the ROM.

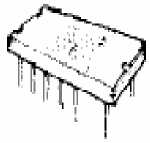


Circuit for driving single 7-segment LED display with 7447





TOP VIEW ORIENTATION

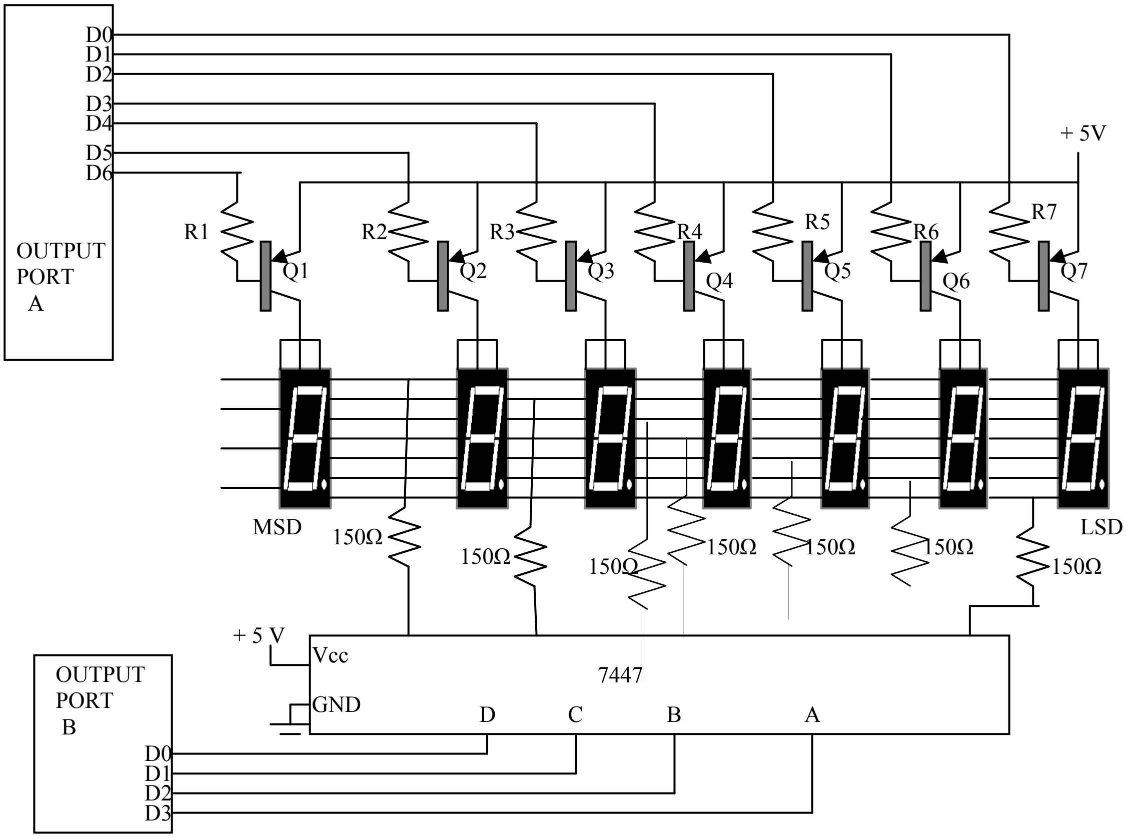


NOTES: a. The true position spacing is 2,54 mm (0.100 inch) between lead centerlines. Each pin centerline is located within 0,25 mm (0.010 inch) of its true longitudinal position.

b. Vertical and horizontal spacing between centerlines of rows and columns nominally 1,27 mm (0.050 inch).

ALL DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

D 37



Liquid Crystal Display

- Liquid Crystal displays are created by sandwiching a thin 10-12 µm layer of a liquid-crystal fluid between two glass plates. A transparent, electrically conductive film or backplane is put on the rear glass sheet. Transparent sections of conductive film in the shape of the desired characters are coated on the front glass plate.
- When a voltage is applied between a segment and the backplane, an electric field is created in the region under the segment. This electric field changes the transmission of light through the region under the segment film.
- There are two commonly available types of LCD : dynamic scattering and field-effect.
- The Dynamic scattering types of LCD: It scrambles the molecules where the field is present. This produces an etched-glass-looking light character on a dark background.

- Field-effect types use polarization to absorb light where the electric field is present. This produces dark characters on a silver- gray background.
- Most LCD's require a voltage of 2 or 3 V between the backplane and a segment to turn on the segment.
- We cannot just connect the backplane to ground and drive the segment with the outputs of a TTL decoder. The reason for this is a steady dc voltage of more than about 50mV is applied between a segment and the backplane.
- To prevent a dc buildup on the segments, the segment-drive signals for LCD must be square waves with a frequency of 30 to 150 Hz.
- Even if you pulse the TTL decoder, it still will not work because the output low voltage of TTL devices is greater than 50mV.
- CMOS gates are often used to drive LCDs.
- The Following fig shows how two CMOS gate outputs can be connected to drive an LCD segment and backplane.
- The off segment receives the same drive signal as the backplane. There is never any voltage between them, so no electric field is produced. The waveform for the on segment is 180 out of phase with the backplane signal, so the voltage between this segment and the backplane will always be +V.
- The logic for this signal, a square wave and its complement. To the driving gates, the segment-backplane sandwich appears as a somewhat leaky capacitor.
- The CMOS gates can be easily supply the current required to charge and discharge this small capacitance.
- Older inexpensive LCD displays turn on and off too slowly to be multiplexed the way we do LED display.
- At 0c some LCD may require as mush as 0.5s to turn on or off. To interface to those types we use a nonmultiplexed driver device.
- More expensive LCD can turn on and off faster, so they are often multiplexed using a variety of techniques.
- In the following section we show you how to interface a nonmultiplexed LCD to a microprocessor such as SDK-86.
- Intersil ICM7211M can be connected to drive a 4-digit, nonmultiplexed, 7-segment LCD display.
- The 7211M input can be connected to port pins or directly to microcomputer bus. We have connected the CS inputs to the Y2 output of the 74LS138 port decoder.
- According to the truth table the device will then be addressable as ports with a base address of FF10H. SDK-86 system address lines A2 is connected to the digit-select input (DS2) and system address lines A1 is connected to the DS1 input. This gives digit 4 a system address of FF10H.

A8-A15	A5-A7	A4	A3	A2	A1	A0	M/IŌ	Y Output Selected	System Base Address	Device
1	0	0	0	X	X	0	0	00	F F 0 0	8259A #1
1	0	0	1	X	X	0	0	1	F F 0 8	8259A #2
1	0	1	0	X	X	0	0	2	F F 1 0	
1	0	1	1	X	X	0	0	3	F F 1 8	
1	0	0	0	X	X	1	0	4	F F 0 1	8254
1	0	0	1	X	X	1	0	5	F F 0 9	
1	0	1	0	X	X	1	0	6	F F 1 1	
1	0	1	1	X	X	1	0	7	F F 1 9	
ALL OTHER STATES								NONE		

Fig : Truth table for 74LS138 address decoder

- Digit 3 will be addressed at FF12H, digit 2 at FF14H and digit 1 at FF16H.
- The data inputs are connected to the lower four lines of the SDK-86 data bus. The oscillator input is left open. To display a character on one of the digits, you simply keep the 4-bit hex code for that digit in the lower 4 bits of the AL register and output it to the system address for that digit.
- The ICM7211M converts the 4-bit hex code to the required 7-segment code.
- The rising edge of the CS input signal causes the 7-segment code to be latched in the output latches for the address digit.
- An internal oscillator automatically generates the segment and backplane drive waveforms as in fig . For interfacing with the LCD displays which can be multiplexed the Intersil ICM7233 can be use.

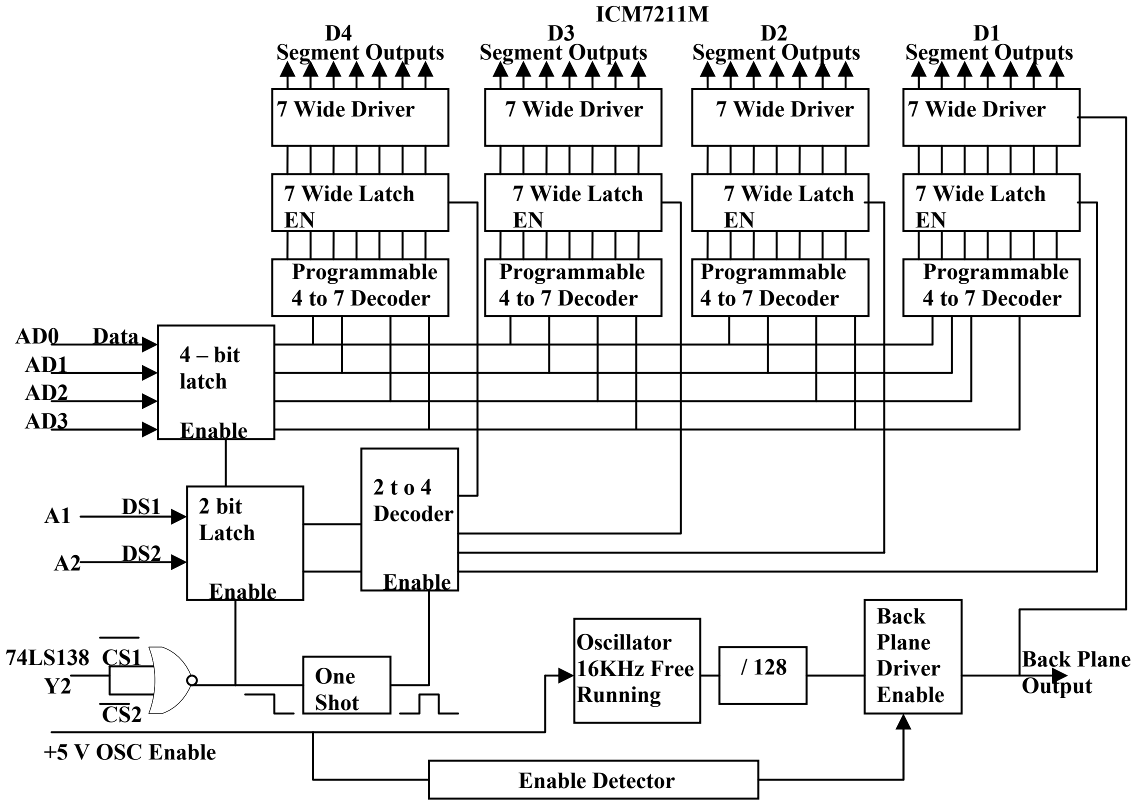


Fig : Circuit for interfacing four LCD digits to an SDK-86 bus using ICM7211M

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.
- 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.

- 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.
- 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.
5. 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.
6. 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.

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 -

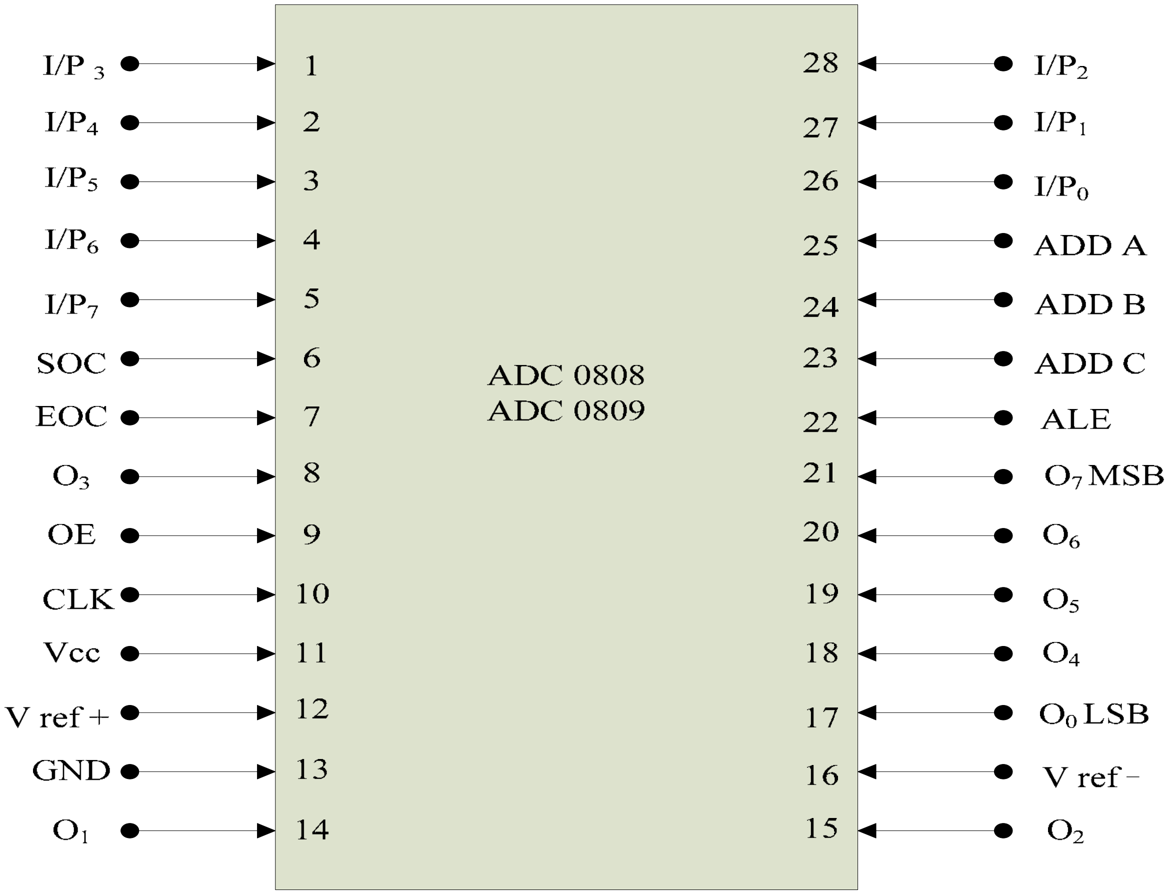
ADD A, ADD B, ADD C. Using these address inputs, multichannel data acquisition system can be designed using a single ADC. The CPU may drive these lines using output port lines in case of multichannel applications. In case of single input applications, these may be hardwired to select the proper input.

- 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.

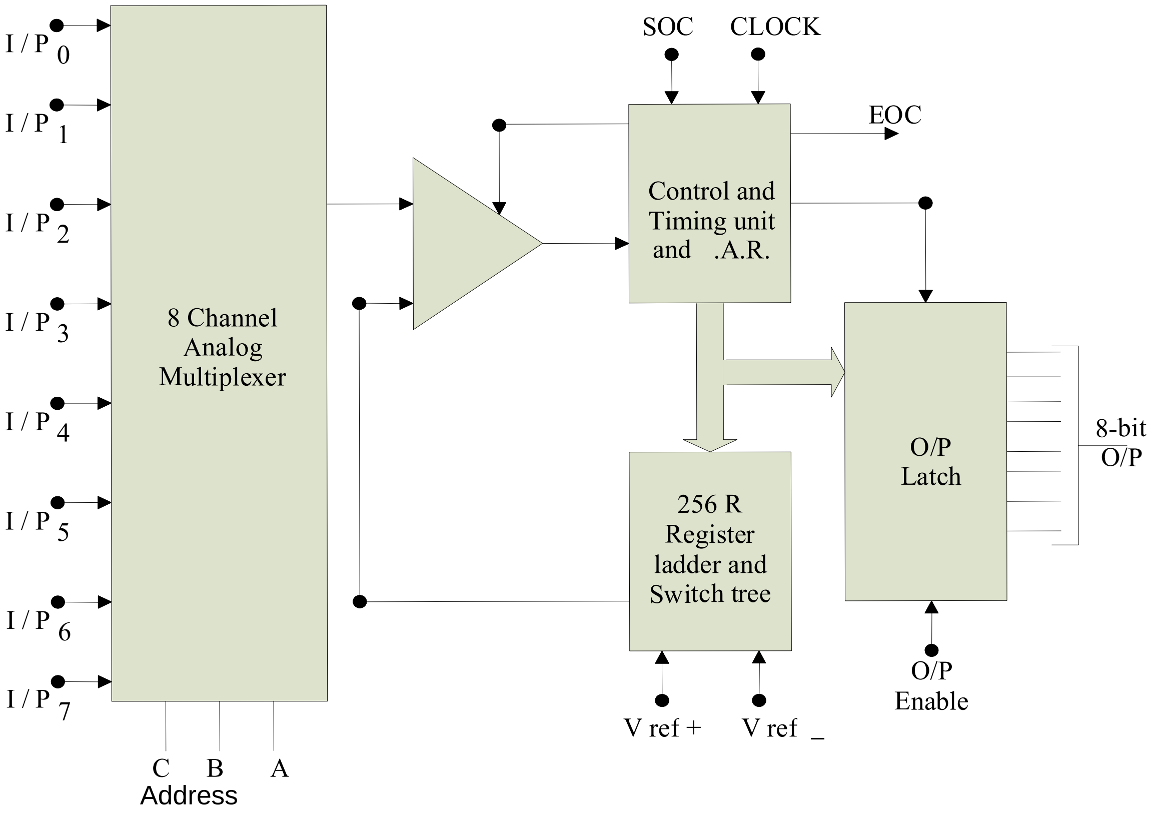
Analog /P selecte	Address lines		
	C	B	A
I / P $_0$	0	0	0
I / P $_1$	0	0	1
I / P $_2$	0	1	0
I / P $_3$	0	1	1
I / P $_4$	1	0	0
I / P $_5$	1	0	1
I / P $_6$	1	1	0
I / P $_7$	1	1	1

- 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.

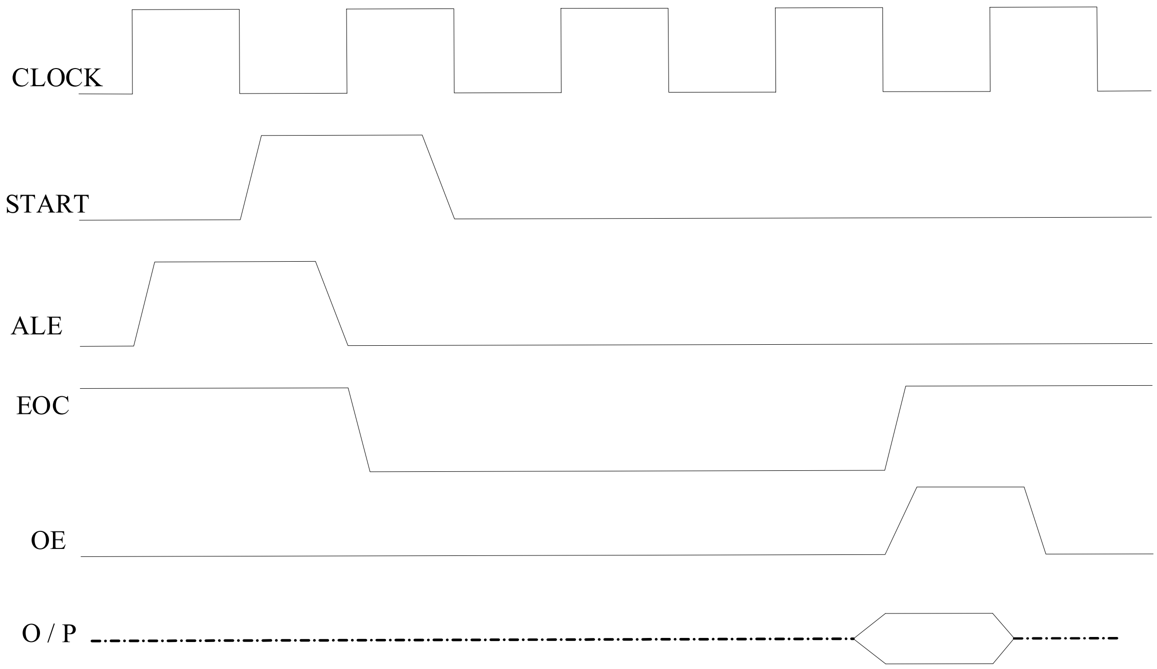
- Vcc Supply pins +5V
- GND GND
- Vref + Reference voltage positive +5 Volts maximum.
- Vref _ Reference voltage negative 0Volts minimum.
- I/P0 –I/P7 Analog inputs
- ADD A,B,C Address lines for selecting analog inputs.
- O7 – O0 Digital 8-bit output with O7 MSB and O0 LSB
- SOC Start of conversion signal pin
- EOC End of conversion signal pin
- OE Output latch enable pin, if high enables output
- CLK Clock input for ADC



PIN DIAGRAM OF ADC 0808 / 0809



Block Diagram of ADC 0808 / 0809



Timing Diagram of ADC 0808

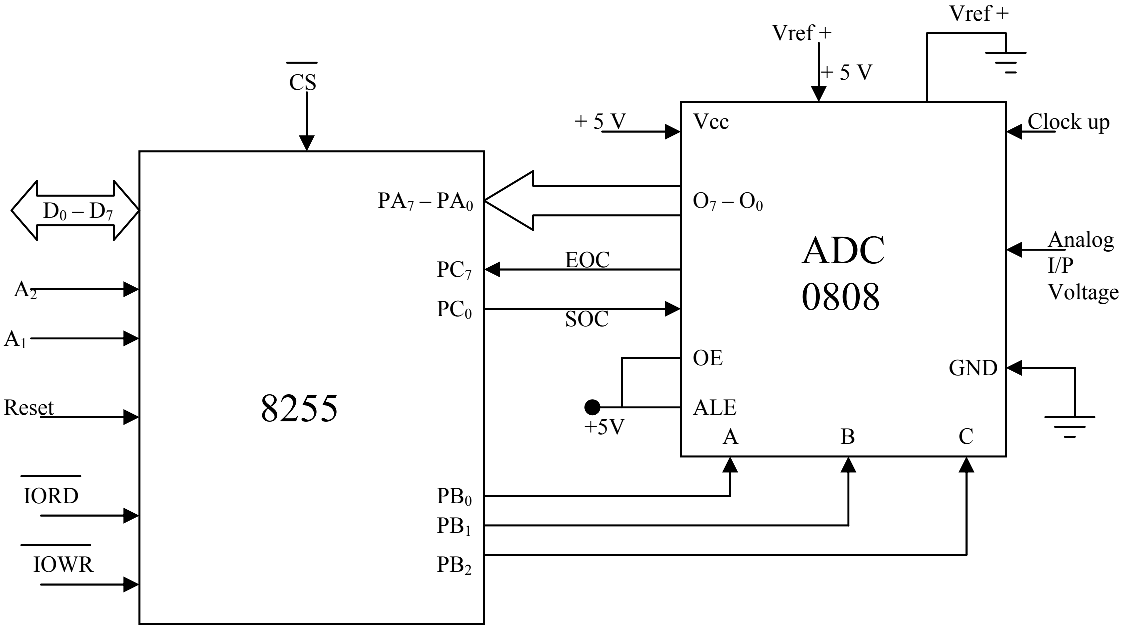
- ***Example:*** Interfacing ADC 0808 with 8086 using 8255 ports. Use port A of 8255 for transferring digital data output of ADC to the CPU and port C for control signals. Assume that an analog input is present at I/P2 of the ADC and a clock input of suitable frequency is available for ADC.
- **Solution**: The analog input I/P2 is used and therefore address pins A,B,C should be 0,1,0 respectively to select I/P2. 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.
- 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:

D7	D6	D5	D4	D3	D2	D1	D0
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.
        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.
```

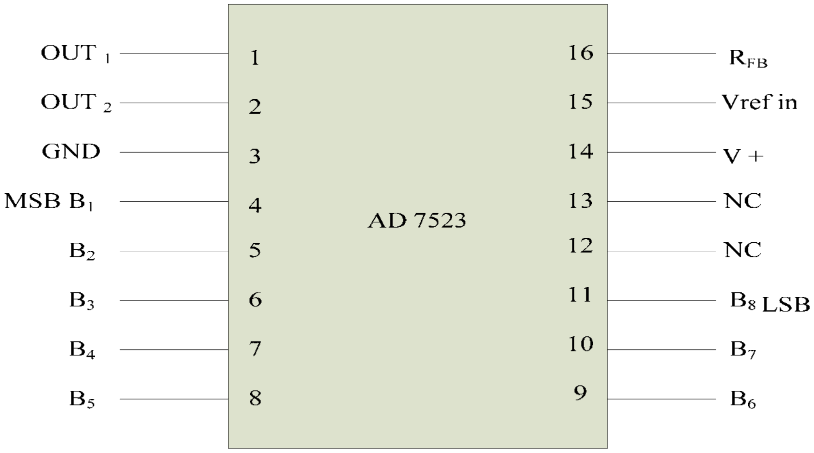


Interfacing 0808 with 8086

Interfacing Digital To Analog Converters

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

- 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.

- 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 8MHZ 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.

```
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
```

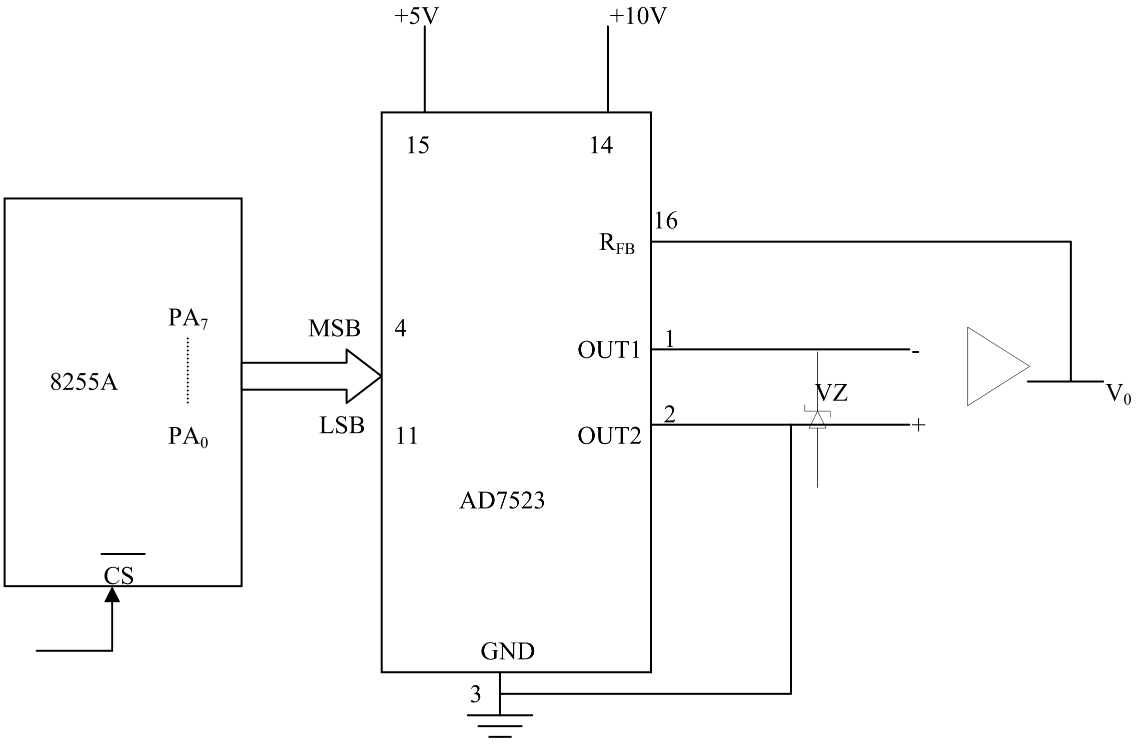


Fig: Interfacing of AD7523

- 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.

MODULE 4

COPROCESSOR 8087

Contents

- ❖ Architecture of 8087
- ❖ Data types
- ❖ Interfacing
- ❖ Instructions and programming

Overview

- Each processor in the 80x86 family has a corresponding coprocessor with which it is compatible.

- Math Coprocessor is known as NPX,NDP,FUP.

  Numeric processor extension (NPX),

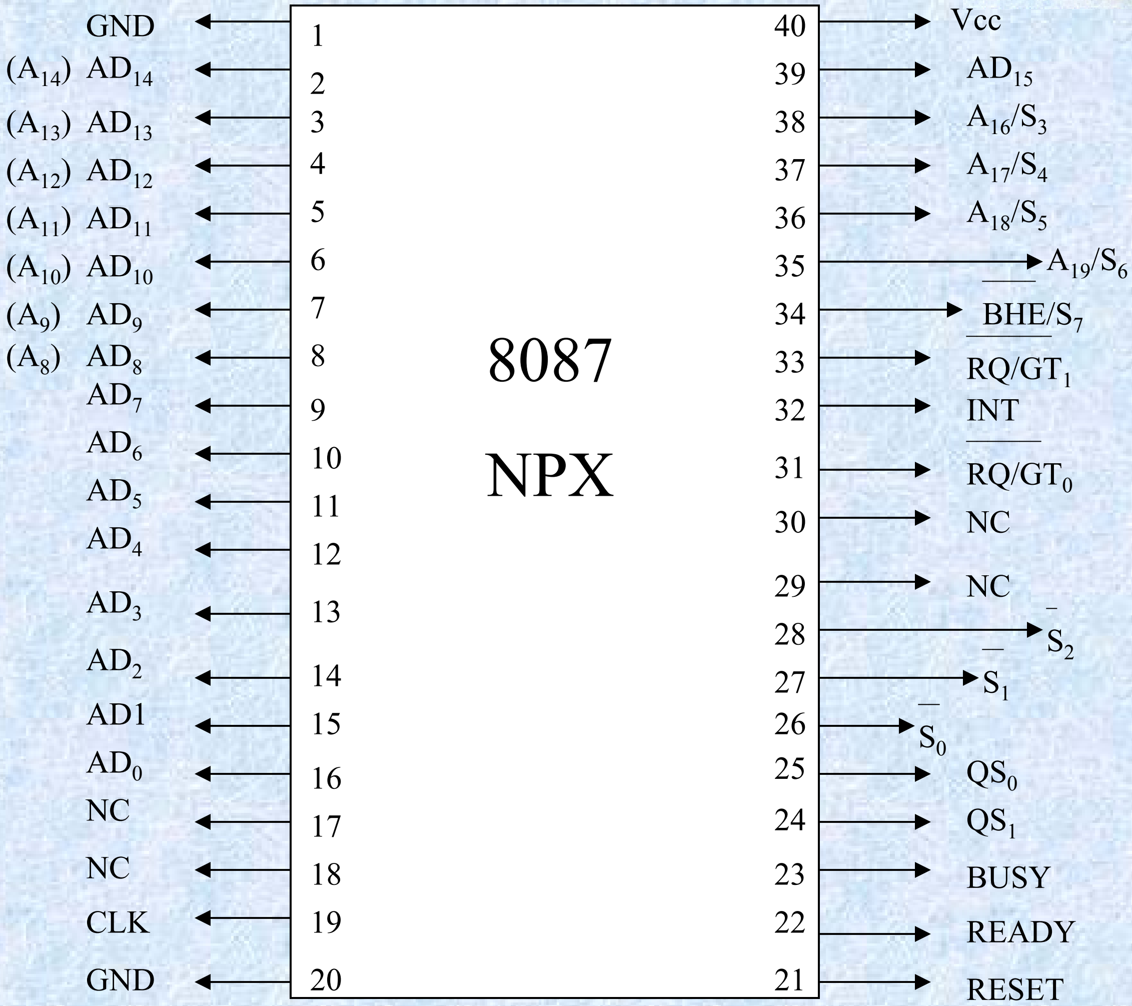
  Numeric data processor (NDP),

  Floating point unit (FUP).

Compatible Processor and Coprocessor

Processors	Coprocessors
1. 8086 & 8088	1. 8087
2. 80286	2. 80287,80287XL
3. 80386DX	3. 80287,80387DX
4. 80386SX	4. 80387SX
5. 80486DX	5. It is Inbuilt
6. 80486SX	6. 80487SX

Pin Diagram of 8087

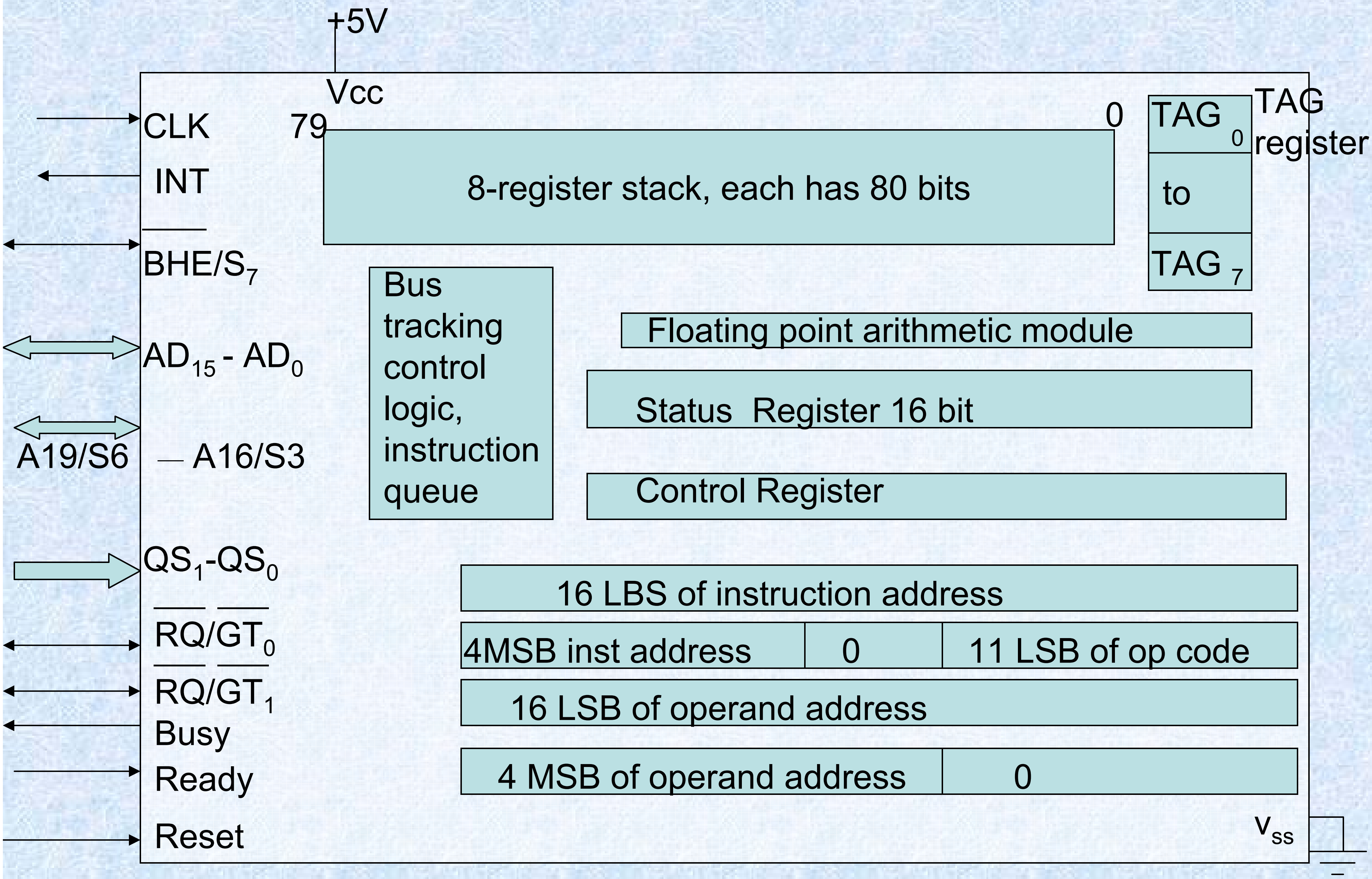


Architecture of 8087

- ❖ Control Unit
- ❖ Execution Unit

BLOCK DIAGRAM OF 8087

BLOCK DIAGRAM OF 8087



+5V

Vcc

CLK

INT

$\overline{BHE}/S_7$

AD_{15} - AD_0

A19/S6 — A16/S3

QS_1-QS_0

$\overline{RQ}/\overline{GT}_0$

$\overline{RQ}/\overline{GT}_1$

Busy

Ready

Reset

79

0

8-register stack, each has 80 bits

TAG_0

to

TAG_7

TAG register

Bus tracking control logic, instruction queue

Floating point arithmetic module

Status Register 16 bit

Control Register

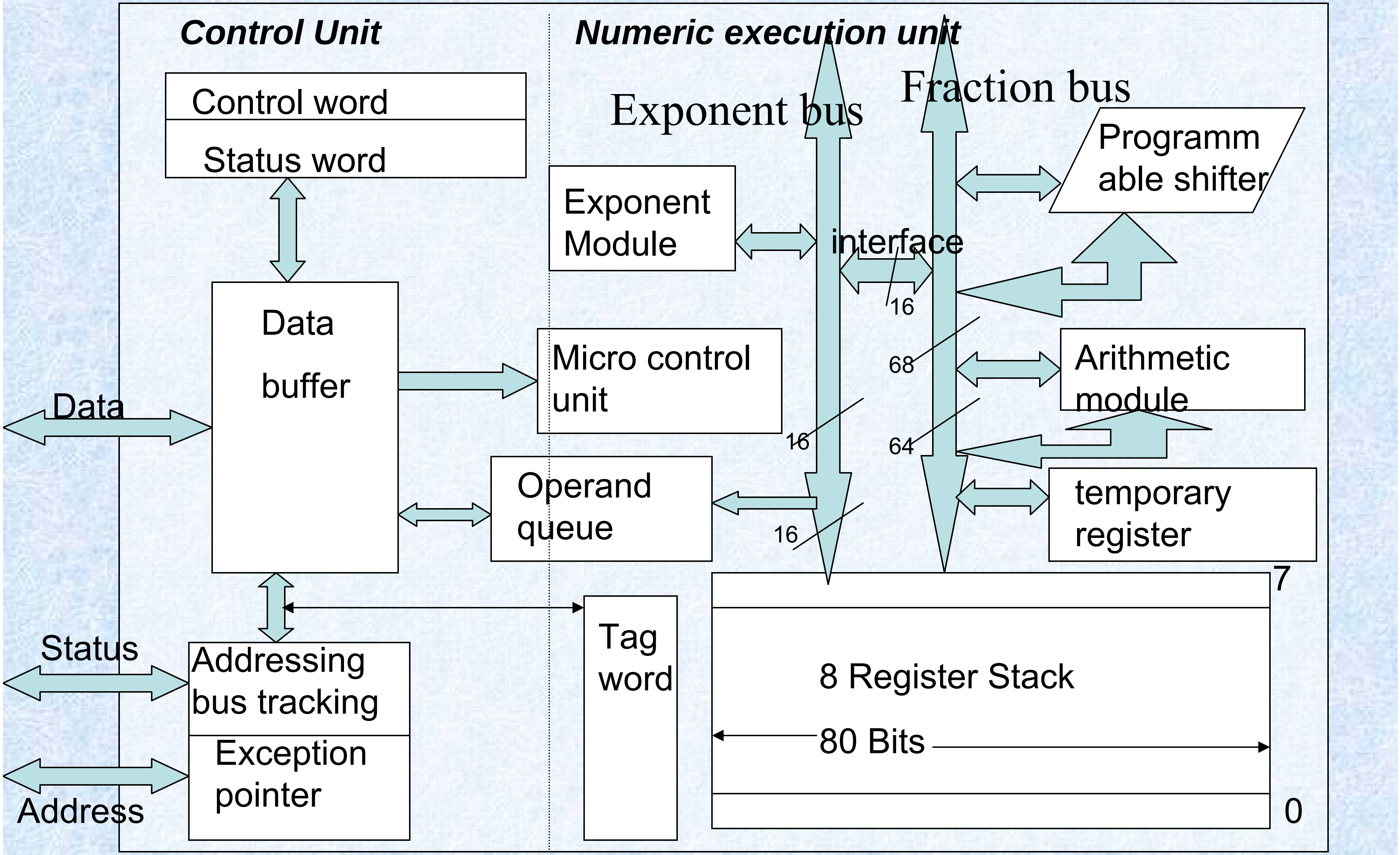
16 LBS of instruction address

4MSB inst address	0	11 LSB of op code

16 LSB of operand address

4 MSB of operand address	0

V_{ss}


Control Unit
Numeric execution unit
Control word
Status word
Exponent bus
Fraction bus
Programm able shifter
Exponent Module
interface
16
Data buffer
68
Micro control unit
Arithmetic module
Data
16
64
Operand queue
temporary register
16
7
Tag word
8 Register Stack
Status
Addressing bus tracking
80 Bits
Exception pointer
Address
0

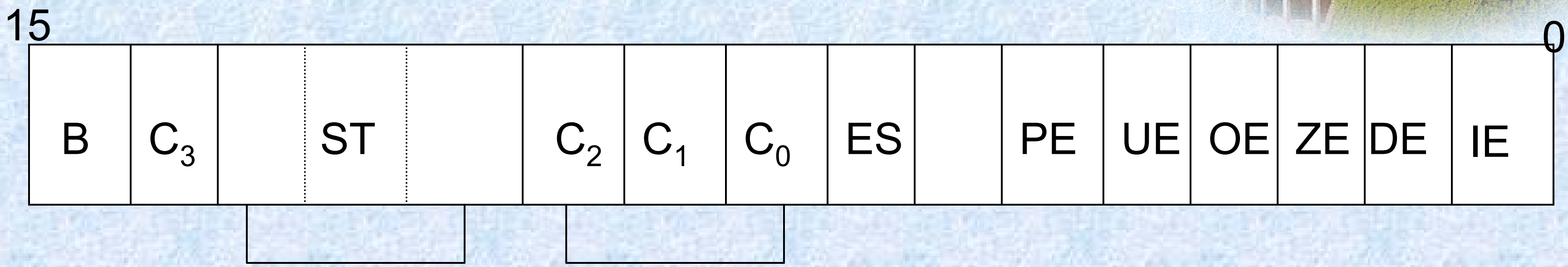
Control Unit

- Control unit: To synchronize the operation of the coprocessor and the processor.

- This unit has a Control word and Status word and Data Buffer

- If instruction is an *ESC*ape (coprocessor) instruction, the coprocessor executes it, if not the microprocessor executes.

Status Register



15													0
B	C_3	ST	C_2	C_1	C_0	ES		PE	UE	OE	ZE	DE	IE

• C_3-C_0	Condition code bits
• TOP	Top-of-stack (ST)
• ES	Error summary
• PE	Precision error
• UE	Under flow error
• OE	Overflow error
• ZE	Zero error
• DE	Denormalized error
• IE	Invalid error
• B	Busy bit

- Status register reflects the over all operation of the coprocessor.

- B-Busy bit indicates that coprocessor is busy executing a task. Busy can be tested by examining the status or by using the FWAIT instruction. Newer coprocessor automatically synchronize with the microprocessor, so busy flag need not be tested before performing additional coprocessor tasks.

- C_3-C_0 Condition code bits indicates conditions about the coprocessor.

- TOP- Top of the stack (ST) bit indicates the current register address as the top of the stack.

- ES-Error summary bit is set if any unmasked error bit (PE, UE, OE, ZE, DE, or IE) is set. In the 8087 the error summary is also caused a coprocessor interrupt.

- PE- Precision error indicates that the result or operand executes selected precision.

- UE-Under flow error indicates the result is too large to be represent with the current precision selected by the control word.

- OE-Over flow error indicates a result that is too large to be represented. If this error is masked, the coprocessor generates infinity for an overflow error.
- ZE-A Zero error indicates the divisor was zero while the dividend is a non-infinity or non-zero number.
- DE-Denormalized error indicates at least one of the operand is denormalized.
- IE-Invalid error indicates a stack overflow or underflow, indeterminate from (0/0,0,-0, etc) or the use of a NAN as an operand. This flag indicates error such as those produced by taking the square root of a negative number.

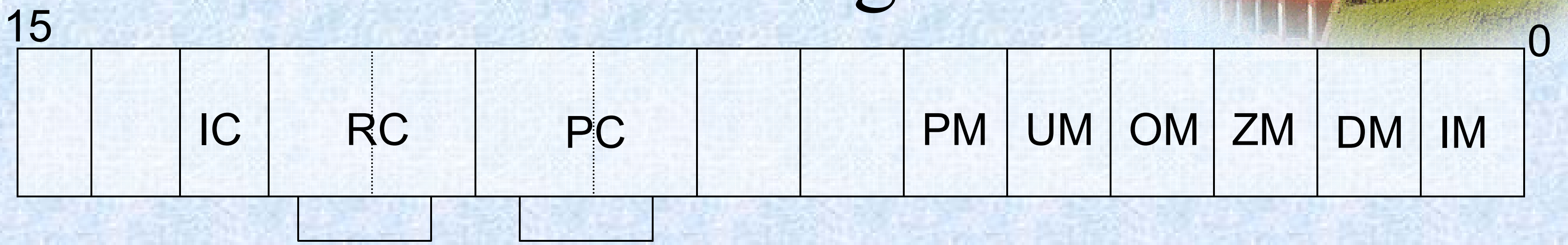
CONTROL REGISTER

- Control register selects precision, rounding control, infinity control.

- It also masks an unmasks the exception bits that correspond to the rightmost Six bits of status register.

- Instruction FLDCW is used to load the value into the control register.

Control Register



• IC	Infinity control
• RC	Rounding control
• PC	Precision control
• PM	Precision control
• UM	Underflow mask
• OM	Overflow mask
• ZM	Division by zero mask
• DM	Denormalized operand mask
• IM	Invalid operand mask

- IC –Infinity control selects either affine or projective infinity. Affine allows positive and negative infinity, while projective assumes infinity

  is unsigned.

INFINITY CONTROL

0 = Projective

1 = Affine

- RC –Rounding control determines the type of rounding.

ROUNDING CONTROL

00=Round to nearest or even

01=Round down towards minus infinity

10=Round up towards plus infinity

11=Chop or truncate towards zero

- PC- Precision control sets the precision of he result as define in table

- Exception Masks – It Determines whether the error indicated by the exception affects the error bit in the status register. If a logic1 is placed in one of the exception control bits, corresponding status register bit is masked off.

PRECISION CONTROL

00=Single precision (short)

01=Reserved

10=Double precision (long)

11=Extended precision (temporary)

Numeric Execution Unit

- This performs all operations that access and manipulate the numeric data in the coprocessor’s registers.

- Numeric registers in NUE are 80 bits wide.

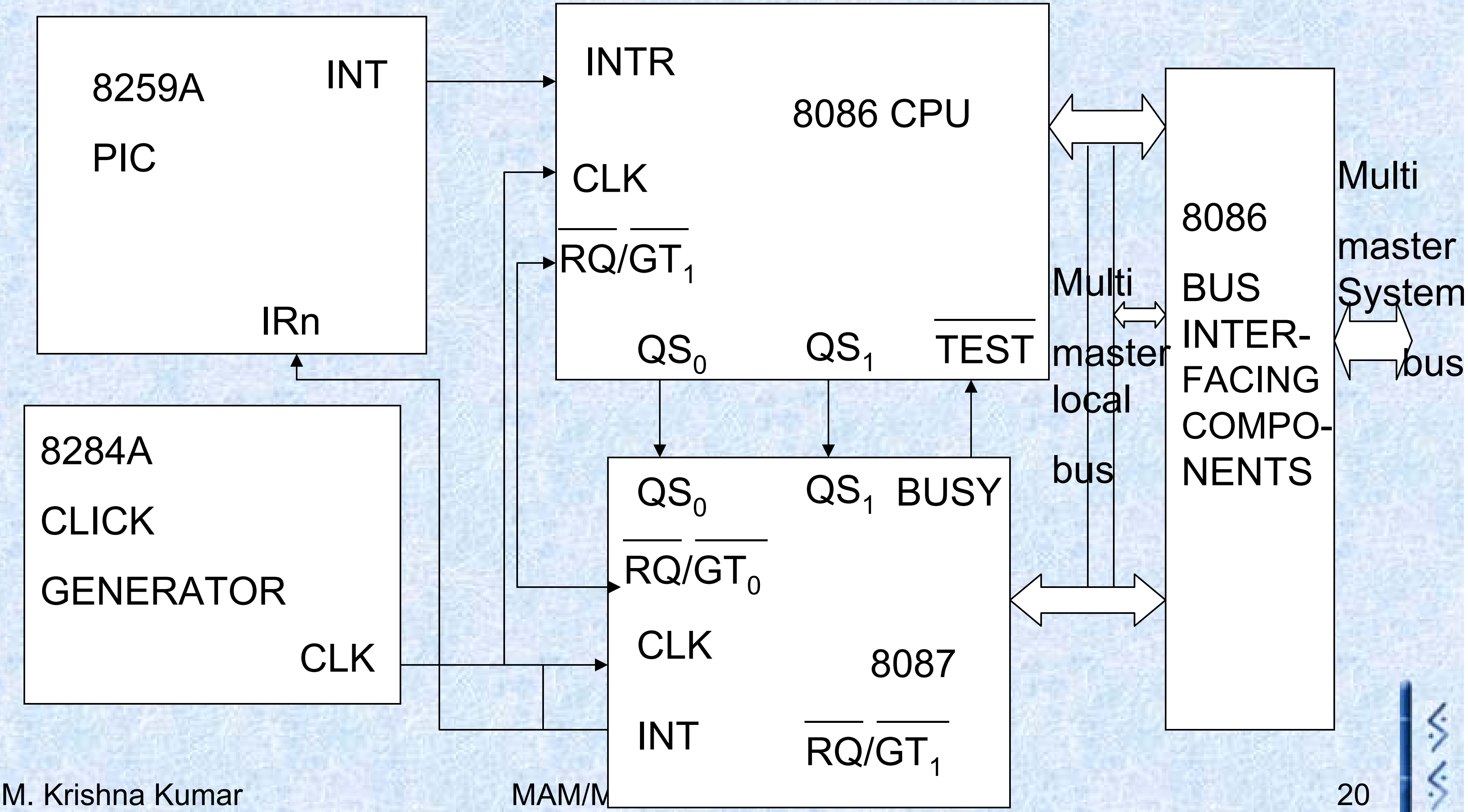
- NUE is able to perform arithmetic, logical and transcendental operations as well as supply a small number of mathematical constants from its on-chip ROM.

- Numeric data is routed into two parts ways

  a 64 bit mantissa bus and

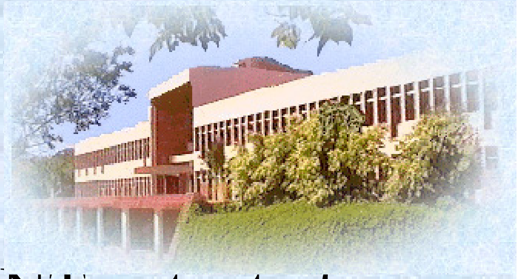
  a 16 bit sign/exponent bus.

Circuit Connection for 8086 - 8087



- Multiplexed address-data bus lines are connected directly from the 8086 to 8087.
- The status lines and the queue status lines connected directly from 8086 to 8087.
- The Request/Grant signal $\overline{RQ}/\overline{GT}_0$ of 8087 is connected to

  $\overline{RQ}/\overline{GT}_1$ of 8086.
- BUSY signal 8087 is connected to $\overline{\text{TEST}}$ pin of 8086.
- Interrupt output INT of the 8087 to NMI input of 8086. This intimates an error condition.



- The main purpose of the circuitry between the INT output of 8087 and the NMI input is to make sure that an NMI signal is not present upon reset, to make it possible to mask NMI input and to make it possible for other devices to cause an NMI interrupt.

- BHE pin is connected to the system BHE line to enable the upper bank of memory.

- The RQ/GT$_1$ input is available so that another coprocessor such as 8089 I/O processor can be connected and function in parallel with the 8087.

- One type of Cooperation between the two processors that you need to know about it is how the 8087 transfers data between memory and its internal registers.

- When 8086 reads an 8087 instruction that needs data from memory or wants to send data to memory, the 8086 sends out the memory address code in the instruction and sends out the appropriate memory read or memory write signal to transfer a word of data.

- In the case of memory read, the addressed word will be kept on the data bus by the memory. The 8087 then simply reads the word of data bus. The 8086 ignores this word .If the 8087 only needs this one word of data, it can then go on and executes its instruction.

- Some 8087 instructions need to read in or write out up to 80-bit word. For these cases 8086 outputs the address of the first data word on the address bus and outputs the appropriate control signal.

- The 8087 reads the data word on the data bus by memory or writes a data word to memory on the data bus. The 8087 grabs the 20-bit physical address that was output by the 8086.To transfer additional words it needs to/from memory, the 8087 then takes over the buses from 8086.
- To take over the bus, the 8087 sends out a low-going pulse on $\overline{RQ}/\overline{GT}_0$ pin. The 8086 responds to this by sending another low going pulse back to the $\overline{RQ}/\overline{GT}_0$ pin of 8087 and by floating its buses.

- The 8087 then increments the address it grabbed during the first transfer and outputs the incremented address on the address bus. When the 8087 output a memory read or memory write signal, another data word will be transferred to or from the 8087.

- The 8087 continues the process until it has transferred all the data words required by the instruction to/from memory.

- When the 8087 is using the buses for its data transfer, it

  sends another low-going pulse out on its $\overline{RQ}/\overline{GT}_0$ pin to 8086 to know it can have the buses back again.

The next type of the synchronization between the host processor and the coprocessor is that required to make sure the 8086 hast does not attempt to execute the next instruction before the 8087 has completed an instruction.

- Taking one situation, in the case where the 8086 needs the data produced by the execution of an 8087 instruction to carry out its next instruction.
- In the instruction sequence for example the 8087 must complete the ***FSTSW STATUS*** instruction before the 8086 will have the data it needs to execute the

  MOV AX , STATUS instruction.
- Without some mechanism to make the 8086 wait until the 8087 completes the FSTSW instruction, the 8086 will go on and execute the ***MOV AX , STATUS*** with erroneous data .
- We solve this problem by connecting the 8087 BUSY output to the TEST pin of the 8086 and putting on the WAIT instruction in the program.

- While 8087 is executing an instruction it asserts its BUSY pin high. When it is finished with an instruction, the 8087 will drop its BUSY pin low. Since the BUSY pin from 8087 is connected to the TEST pin 8086 the processor can check its pin of 8087 whether it finished it instruction or not.

- You place the 8086 WAIT instruction in your program after the 8087 FSTSW instruction .When 8086 executes the WAIT instruction it enters an internal loop where it repeatedly checks the logic level on the TEST input. The 8086 will stay in this loop until it finds the TEST input asserted low, indicating the 8087 has completed its instruction. The 8086 will then exit the internal loop, fetch and execute the next instruction.

Example

```
FSTSW          STATUS      ;copy  8087 status word to memory
MOV            AX, STATUS ;copy status word to AX to check
                           ; bits
```

(a)

- In this set of instructions we are not using WAIT instruction. Due to this the flow of execution of command will takes place continuously even though the previous instruction had not finished it’s completion of its work .so we may lost data .

```
FSTSW     STATUS       ;copy 8087 status word to memory
FWAIT                  ;wait for 8087 to finish before-
                       ; doing next 8086 instruction
MOV       AX,STATUS    ;copy status word to AX to check
                       ; bits
```

(b)

- In this code we are adding up of FWAIT instruction so that it will stop the execution of the command until the above instruction is finishes it’s work .so that you are not loosing data and after that you will allow to continue the execution of instructions.

- Another case where you need synchronization of the processor and the coprocessor is the case where a program has several 8087 instructions in sequence.
- The 8087 are executed only one instruction at a time so you have to make sure that 8087 has completed one instruction before you allow the 8086 to fetch the next 8087 instruction from memory.
- Here again you use the BUSY-$\overline{\text{TEST}}$ connection and the FWAIT instruction to solve the problem. If you are hand coding, you can just put the 8086 WAIT(FWAIT) instruction after each instruction to make sure that instruction is completed before going on to next.

- If you are using the assembler which accepts 8087 mnemonics, the assembler will automatically insert the 8-bit code for the WAIT instruction ,10011011 binary (9BH), as the first byte of the code for 8087 instruction.

INTERFACING

- Multiplexed address-data bus lines are connected directly from the 8086 to 8087.
- The status lines and the queue status lines connected directly from 8086 to 8087.
- The Request/Grant signal $\overline{RQ}/\overline{GT0}$ of 8087 is connected to

  $\overline{RQ}/\overline{GT}1$ of 8086.
- BUSY signal 8087 is connected to $\overline{TEST}$ pin of 8086.
- Interrupt output INT of the 8087 to NMI input of 8086. This intimates an error condition.
- A WAIT instruction is passed to keep looking at its $\overline{TEST}$ pin, until it finds pin Low to indicates that the 8087 has completed the computation.

- SYNCHRONIZATION must be established between the processor and coprocessor in two situations.

a) The execution of an ESC instruction that require the participation of the NUE must not be initiated if the NUE has not completed the execution of the previous instruction.

b) When a processor instruction accesses a memory location that is an operand of a previous coprocessor instruction .In this case CPU must synchronize with NPX to ensure that it has completed its instruction.

Processor WAIT instruction is provided.

Exception Handling

- The 8087 detects six different types of exception conditions that occur during instruction execution. These will cause an interrupt if unmasked and interrupts are enabled.

1) INVALID OPERATION
2) OVERFLOW
3) ZERO DIVISOR
4) UNDERFLOW
5) DENORMALIZED OPERAND
6) INEXACT RESULT

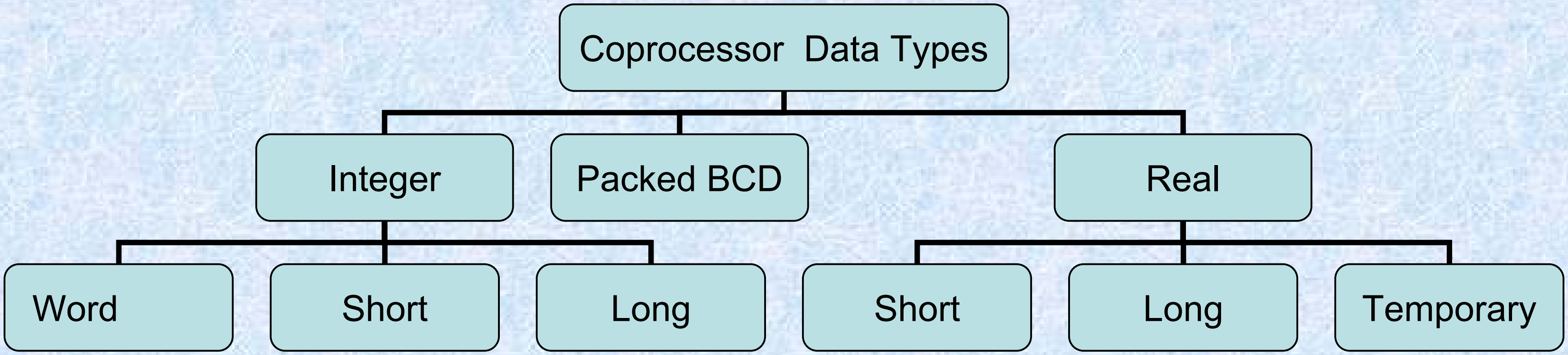
Data Types

- Internally, all data operands are converted to the 80-bit temporary real format.

We have 3 types.

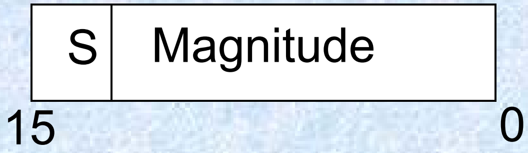
- Integer data type
- Packed BCD data type
- Real data type

Coprocessor data types



Integer Data Type

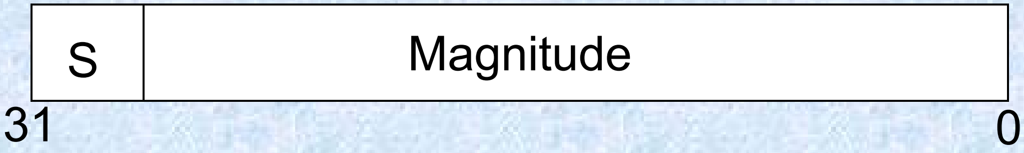
- Word integer 2 bytes



S Magnitude

15 0

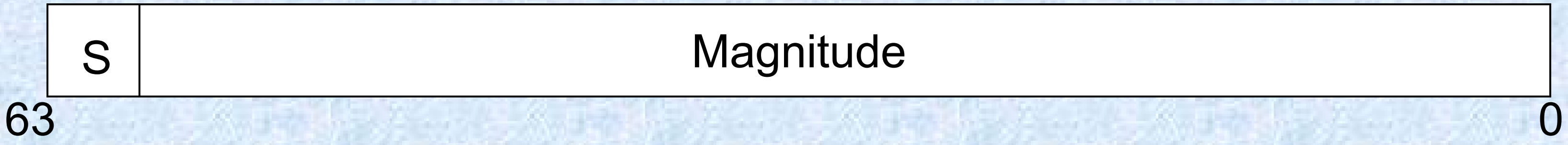
- Short integer 4 bytes



S Magnitude

31 0

- Long integer 8 bytes



S Magnitude

63 0

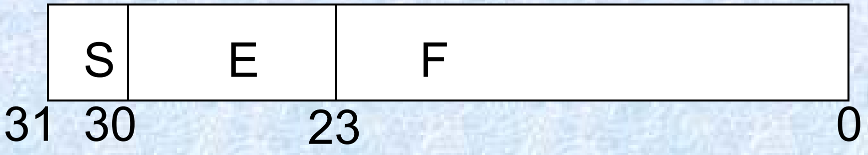
Packed BCD

- Packed BCD 10 bytes

S	0	d_{17}		d_1	d_0
79	78	72			0

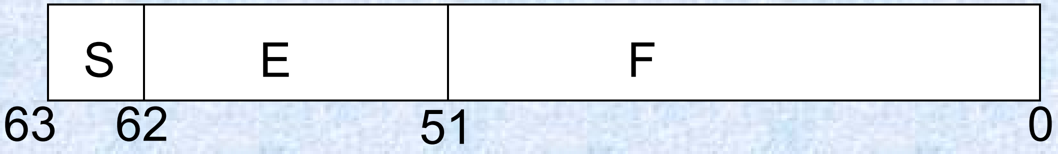
Real data type

- Short real 4 bytes 178.625 decimal



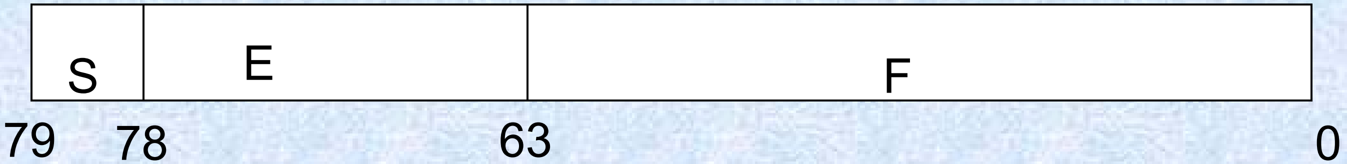
= 4332A000h

- Long real 8 bytes



= 4066540000000000h

- Temporary real 10 bytes



= 4006B2A0000000000000h

Example

- ➢ Converting a decimal number into a Floating-point number.

1) Converting the decimal number into binary form.
2) Normalize the binary number
3) Calculate the biased exponent.
4) Store the number in the floating-point format.

Example

Step	Result
1	100.25
2	$1100100.01 = 1.10010001 * 2^6$
3	110+01111111=10000101

4	Sign = 0
	Exponent =10000101
	Significand = 10010001000000000000000

- In step 3 the biased exponent is the exponent a 2^6 or 110,plus a bias of 01111111(7FH) ,single precision no use 7F and double precision no use 3FFFH.
- IN step 4 the information found in prior step is combined to form the floating point no.

INSTRUCTION SET

- The 8087 instruction mnemonics begins with the letter F which stands for Floating point and distinguishes from 8086.

- These are grouped into Four functional groups.

- The 8087 detects an error condition usually called an exception when it executing an instruction it will set the bit in its Status register.

Types

I. DATA TRANSFER INSTRUCTIONS.

II. ARITHMETIC INSTRUCTIONS.

III. COMPARE INSTRUCTIONS.

IV. TRANSCENDENTAL INSTRUCTIONS.
(Trigonometric and Exponential)

I. Data Transfers Instructions

- **REAL TRANSFER**

FLD	Load real
FST	Store real
FSTP	Store real and pop
FXCH	Exchange registers

- **INTEGER TRANSFER**

FILD	Load integer
FIST	Store integer
FISTP	Store integer and pop

- **PACKED DECIMAL TRANSFER(BCD)**

FBLD	Load BCD
FBSTP	Store BCD and pop

Example

- **FLD Source-** Decrements the stack pointer by one and copies a real number from a stack element or memory location to the new ST.

- FLD ST(3) ;Copies ST(3) to ST.
- FLD LONG_REAL[BX] ;Number from memory ;copied to ST.

- **FLD Destination-** Copies ST to a specified stack position or to a specified memory location .

- FST ST(2) ;Copies ST to ST(2),and ;increment stack pointer.
- FST SHORT_REAL[BX] ;Copy ST to a memory at a ;SHORT_REAL[BX]

- **FXCH Destination** – Exchange the contents of ST with the contents of a specified stack element.
- FXCH ST(5) ;Swap ST and ST(5)
- **FILD Source** – Integer load. Convert integer number from memory to temporary-real format and push on 8087 stack.
- FILD DWORD PTR[BX] ;Short integer from memory at ; [BX].
- **FIST Destination-** Integer store. Convert number from ST to integer and copy to memory.
- FIST LONG_INT ;ST to memory locations named ;LONG_INT.

- **FISTP Destination-**Integer store and pop. Identical to FIST except that stack pointer is incremented after copy.
- **FBLD Source-** Convert BCD number from memory to temporary- real format and push on top of 8087 stack.

II. Arithmetic Instructions.

❖ Four basic arithmetic functions:
Addition, Subtraction, Multiplication, and Division.

➢ **Addition**

FADD	Add real
FADDP	Add real and pop
FIADD	Add integer

- **Subtraction**

FSUB	Subtract real
FSUBP	Subtract real and pop
FISUB	Subtract integer
FSUBR	Subtract real reversed
FSUBRP	Subtract real and pop
FISUBR	Subtract integer reversed

- **Multiplication**

FMUL	Multiply real
FMULP	Multiply real and pop
FIMUL	Multiply integer

- **Division**

FDIV	Division real
FDIVP	Division real and pop
FIDIV	Division integer
FDIVR	Division real reversed
FDIVRP	Division real reversed and pop
FIDIVR	Division integer reversed

- **Advanced**

FABS	Absolute value
FCHS	Change sign
FPREM	Partial remainder
FPRNDINT	Round to integer
FSCALE	Scale
FSQRT	Square root
FXTRACT	Extract exponent and mantissa.

Example

- **FADD** – Add real from specified source to specified destination Source can be a stack or memory location. Destination must be a stack element. If no source or destination is specified, then ST is added to ST(1) and stack pointer is incremented so that the result of addition is at ST.

```
•  FADD      ST(3), ST      ;Add ST to ST(3), result in ST(3)
•  FADD      ST,ST(4)       ;Add ST(4) to ST, result in ST.
•  FADD                     ;ST + ST(1), pop stack result at ST
•  FADDP     ST(1)          ;Add ST(1) to ST. Increment stack
                            ;pointer so ST(1) become ST.
•  FIADD      Car_Sold      ;Integer number from memory + ST
```

- **FSUB** - Subtract the real number at the specified source from the real number at the specified destination and put the result in the specified destination.
- FSUB   ST(2), ST   ;ST(2)=ST(2) – ST.
- FSUB   Rate   ;ST=ST – real no from memory.
- FSUB   ;ST=(ST(1) – ST)
- **FSUBP** - Subtract ST from specified stack element and put result in specified stack element .Then increment the pointer by one.
- FSUBP   ST(1)   ;ST(1)-ST. ST(1) becomes new ST
- **FISUB** – Integer from memory subtracted from ST, result in ST.
- FISUB   Cars_Sold   ;ST becomes ST – integer from ;memory

III. Compare Instructions.

- **Comparison**

FCOM	Compare real
FCOMP	Compare real and pop
FCOMPP	Compare real and pop twice
FICOM	Compare integer
FICOMP	Compare integer and pop
FTST	Test ST against +0.0
FXAM	Examine ST

IV. Transcendental Instruction.

- **Transcendental**

FPTAN	Partial tangent
FPATAN	Partial arctangent
F2XM1	$2^x - 1$
FYL2X	$Y \log_2 X$
FYL2XP1	$Y \log_2(X+1)$

Example

- **FPTAN –** Compute the values for a ratio of Y/X for an angle in ST. The angle must be in radians, and the angle must be in the range of $0 < \text{angle} < \pi/4$.
- **F2XM1 –** Compute $Y=2^x-1$ for an X value in ST. The result Y replaces X in ST. X must be in the range $0 \leq X \leq 0.5$.
- **FYL2X** - Calculate $Y(LOG_2X)$.X must be in the range of $0 < X < \infty$ any Y must be in the range $-\infty<Y<+\infty$.
- **FYL2XP1 –** Compute the function $Y(LOG_2(X+1))$.This instruction is almost identical to FYL2X except that it gives more accurate results when compute log of a number very close to one.

Constant Instructions.

- **Load Constant Instruction**

FLDZ	Load +0.0
FLDI	Load+1.0
FLDPI	Load π
FLDL2T	Load $\log_2 10$
FLDL2E	Load $\log_2 e$
FLDLG2	Load $\log_{10} 2$
FLDLN2	Load $\log_e 2$

ALGORITHM

To calculate x to the power of y

- Load base, power.
- Compute (y)*($\log_2$ x)
- Separate integer(i) ,fraction(f) of a real number
- Divide fraction (f) by 2
- Compute $({}_2{}^{f/2}) * ({}_2{}^{f/2})$
- $x^y = ({}_2{}^x) * ({}_2{}^y)$

Program

Program to calculate x to the power of y

```
.MODEL SMALL
.DATA
x          Dq     4.567   ;Base
y          Dq     2.759   ;Power
temp       DD
temp1      DD
temp2      DD             ;final  real result
tempint    DD
tempint1   DD             ;final integer result
two        DW
diff       DD
trunc_cw   DW     0fffh
```

```
                .STACK 100h
                .CODE
start:          mov ax,@DATA          ;init data segment
                mov ds,ax

load:           fld y                 ;load the power
                fld x                 ;load the base

comput:         fyl2x                 ;compute (y * log2(x))
                fst temp              ;save the temp result
```

```
trunc:      fldcw  trunc_cw         ;set truncation command
            frndint
            fld  temp               ;load real number of fyl2x
            fist  tempint           ;save integer after
                                    ;truncation
            fld  temp          ;load the real number

getfrac:    fisub  tempint   ;subtract the integer
            fst   diff       ;store the fraction
```

```
fracby2:        fidiv two      ;divide the fraction by 2

twopwrx:        f2xm1           ;calculate the 2 to the
                                ;power fraction
                fst temp1      ;minus 1 and save the result
                fld1           ;load1
                fadd           ;add 1 to the previous result
                fst temp1      ;save the result
```

```
sqfrac:    fmul st(0),st(0)   ;square the result as fraction
           fst temp1          ;was halved and save the
                              ;result
           fild tempint       ;save the integer portion
           fxch               ;interchange the integer
                              ;and power of fraction.
```

```
scale:      fscale              ;scale the result in real  and
                                ;integer
            fst temp2           ;in st(1) and store
            fist tempint1       ;save the final result in real and
                                ;integer

 over:      mov ax,4c00h        ;exit to dos
            int 21h
            end start
```

Contents

❖Architecture of 8087

❖Data types

❖Interfacing

❖Instructions and programming

Overview

➢Each processor in the 80x86 family has a corresponding coprocessor with which it is compatible.

➢Math Coprocessor is known as NPX,NDP,FUP.
Numeric processor extension (NPX),
Numeric data processor (NDP),
Floating point unit (FUP).

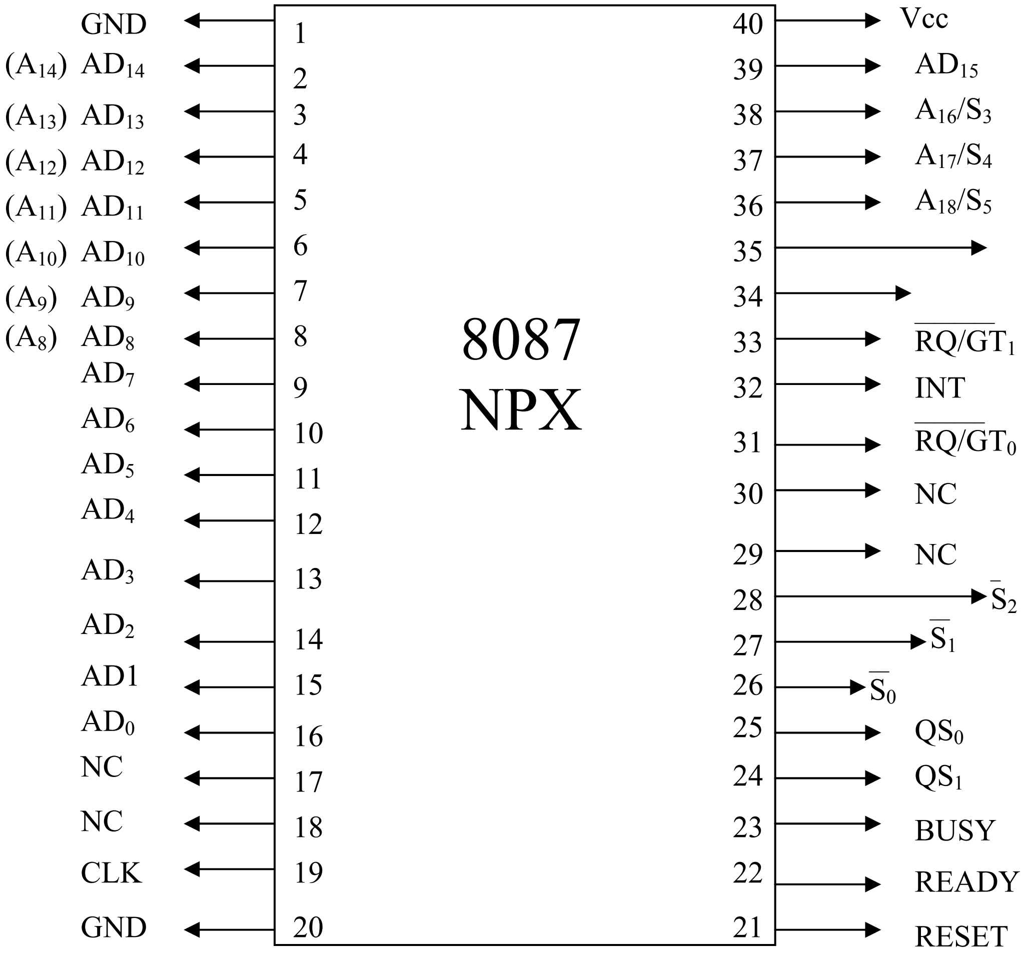
Compatible Processor and Coprocessor

Processors

1. 8086 & 8088
2. 80286
3. 80386DX
4. 80386SX
5. 80486DX
6. 80486SX

Coprocessors

1. 8087
2. 80287,80287XL
3. 80287,80387DX
4. 80387SX
5. It is Inbuilt
6. 80487SX

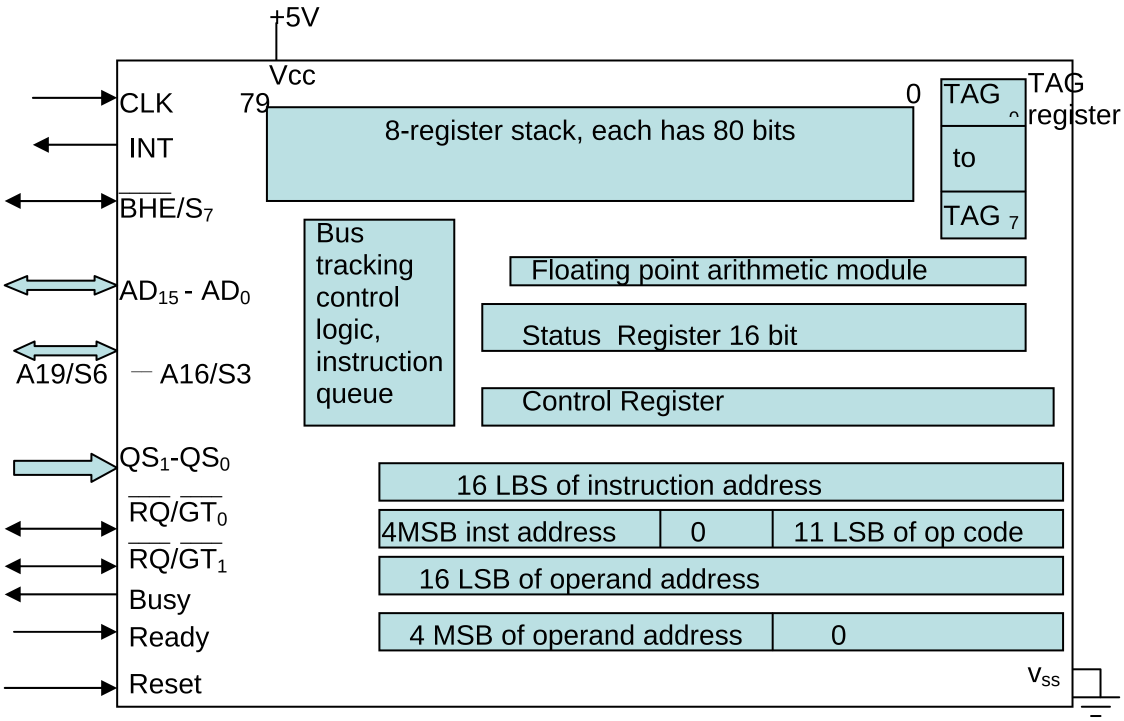


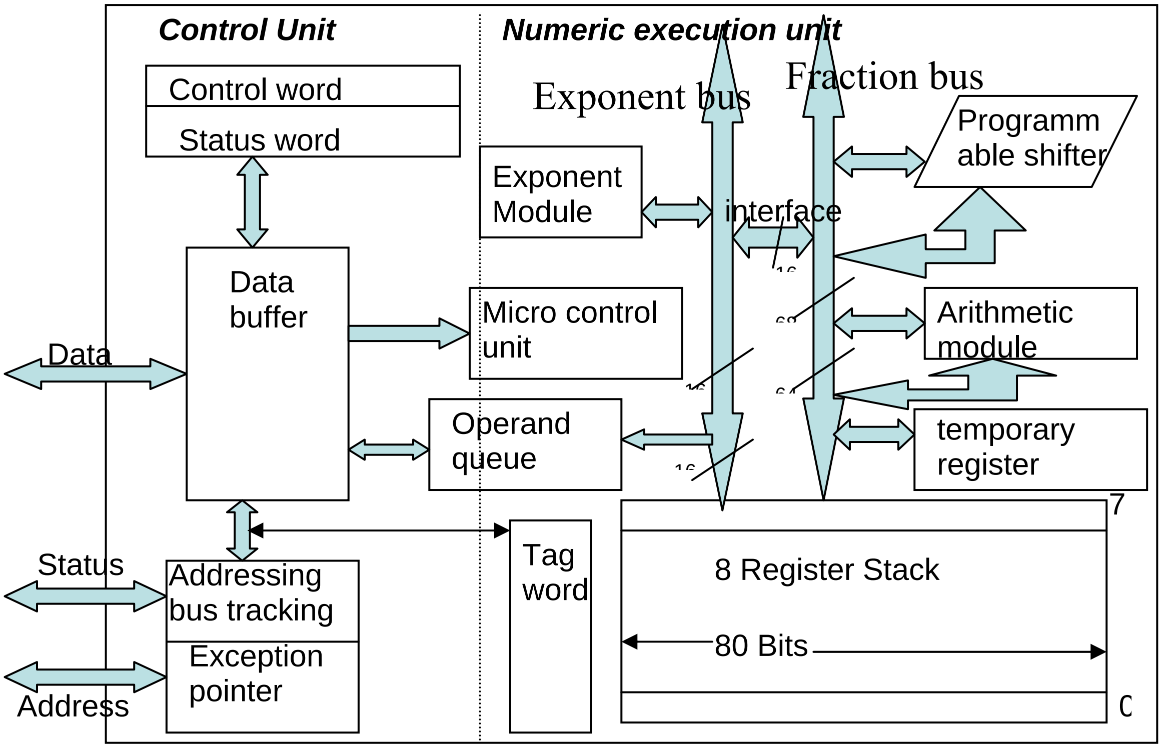
Pin Diagram of 8087

Architecture of 8087

❖Control Unit

❖Execution Unit


+5V
Vcc
CLK
INT
BHE/S7
AD15 - AD0
A19/S6
A16/S3
QS1-QS0
RQ/GT0
RQ/GT1
Busy
Ready
Reset
79
0
8-register stack, each has 80 bits
TAG 0
to
TAG 7
TAG register
Bus tracking control logic, instruction queue
Floating point arithmetic module
Status Register 16 bit
Control Register
16 LBS of instruction address
4MSB inst address
0
11 LSB of op code
16 LSB of operand address
4 MSB of operand address
0
VSS


Control Unit
Numeric execution unit
Control word
Status word
Exponent bus
Fraction bus
Programmable shifter
Exponent Module
interface
Data buffer
Micro control unit
Arithmetic module
Data
Operand queue
temporary register
16
68
16
64
16
7
Tag word
8 Register Stack
80 Bits
0
Status
Addressing bus tracking
Exception pointer
Address

Control Unit

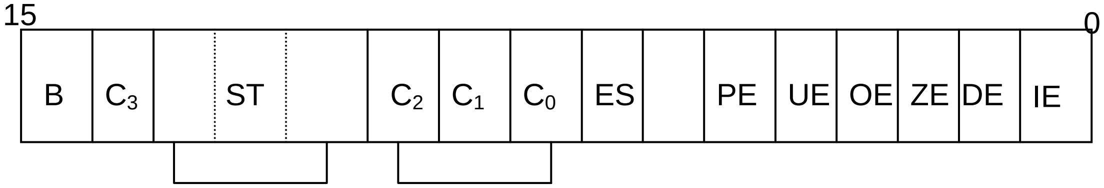
➢Control unit: To synchronize the operation of the coprocessor and the processor.

➢This unit has a Control word and Status word and Data Buffer

➢If instruction is an *ESC*ape (coprocessor) instruction, the coprocessor executes it, if not the microprocessor executes.

➢Status register reflects the over all operation of the coprocessor.

Status Register



• C3-C0	Condition code bits
• TOP	Top-of-stack (ST)
• ES	Error summary
• PE	Precision error
• UE	Under flow error
• OE	Overflow error
• ZE	Zero error
• DE	Denormalized error
• IE	Invalid error
• B	Busy bit

➢B-Busy bit indicates that coprocessor is busy executing a task. Busy can be tested by examining the status or by using the FWAIT instruction. Newer coprocessor automatically synchronize with the microprocessor, so busy flag need not be tested before performing additional coprocessor tasks.

➢C_3-C_0 Condition code bits indicates conditions about the coprocessor.

➢TOP- Top of the stack (ST) bit indicates the current register address as the top of the stack.

➢ES-Error summary bit is set if any unmasked error bit (PE, UE, OE, ZE, DE, or IE) is set. In the 8087 the error summary is also caused a coprocessor interrupt.

➢PE- Precision error indicates that the result or operand executes selected precision.

➢UE-Under flow error indicates the result is too large to be represent with the current precision selected by the control word.

➢OE-Over flow error indicates a result that is too large to be represented. If this error is masked, the coprocessor generates infinity for an overflow error.

➢ZE-A Zero error indicates the divisor was zero while the dividend is a non-infinity or non-zero number.

➢DE-Denormalized error indicates at least one of the operand is denormalized.

➢IE-Invalid error indicates a stack overflow or underflow, indeterminate from (0/0,0,-0, etc) or the use of a NAN as an operand. This flag indicates error such as those produced by taking the square root of a negative number.

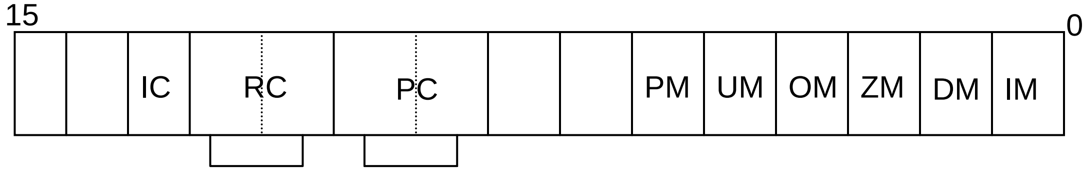
CONTROL REGISTER

➢Control register selects precision, rounding control, infinity control.

➢It also masks an unmasks the exception bits that correspond to the rightmost Six bits of status register.

➢Instruction FLDCW is used to load the value into the control register.

Control Register



•IC	Infinity control
•RC	Rounding control
•PC	Precision control
•PM	Precision control
•UM	Underflow mask
•OM	Overflow mask
•ZM	Division by zero mask
•DM	Denormalized operand mask
•IM	Invalid operand mask

➢IC –Infinity control selects either affine or projective infinity. Affine allows positive and negative infinity, while projective assumes infinity is unsigned.

INFINITY CONTROL
0 = Projective
1 = Affine

➢RC –Rounding control determines the type of rounding.

ROUNDING CONTROL
00=Round to nearest or even
01=Round down towards minus infinity
10=Round up towards plus infinity
11=Chop or truncate towards zero

➢PC- Precision control sets the precision of he result as define in table

PRECISION CONTROL

00=Single precision (short)
01=Reserved
10=Double precision (long)
11=Extended precision (temporary)

➢Exception Masks – It Determines whether the error indicated by the exception affects the error bit in the status register. If a logic1 is placed in one of the exception control bits, corresponding status register bit is masked off.

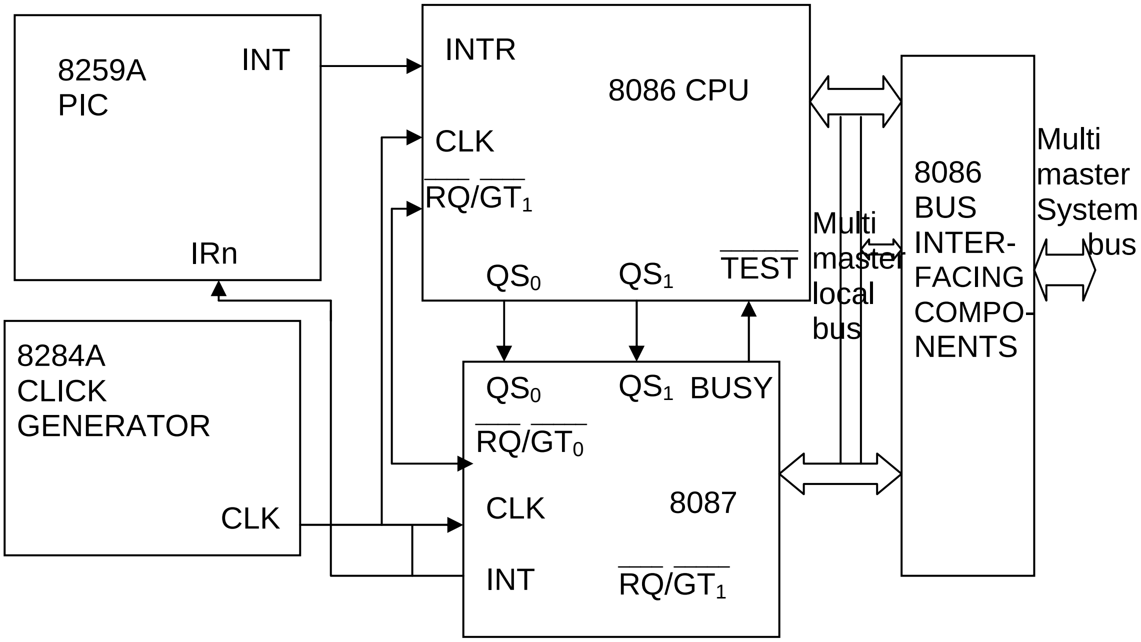
Numeric Execution Unit

➢This performs all operations that access and manipulate the numeric data in the coprocessor’s registers.

➢Numeric registers in NUE are 80 bits wide.

➢NUE is able to perform arithmetic, logical and transcendental operations as well as supply a small number of mathematical constants from its on-chip ROM.

➢Numeric data is routed into two parts ways a 64 bit mantissa bus and
a 16 bit sign/exponent bus.



- Multiplexed address-data bus lines are connected directly from the 8086 to 8087.
- The status lines and the queue status lines connected directly from 8086 to 8087.
- The Request/Grant signal $\overline{RQ}/\overline{GT0}$ of 8087 is connected to

$\overline{RQ}/\overline{GT1}$ of 8086.

- BUSY signal 8087 is connected to $\overline{TEST}$ pin of 8086.
- Interrupt output INT of the 8087 to NMI input of 8086. This intimates an error condition.

- The main purpose of the circuitry between the INT output of 8087 and the NMI input is to make sure that an NMI signal is not present upon reset, to make it possible to mask NMI input and to make it possible for other devices to cause an NMI interrupt.

- BHE pin is connected to the system BHE line to enable the upper bank of memory.

- The RQ/GT$_1$ input is available so that another coprocessor such as 8089 I/O processor can be connected and function in parallel with the 8087.

- One type of Cooperation between the two processors that you need to know about it is how the 8087 transfers data between memory and its internal registers.

➢When 8086 reads an 8087 instruction that needs data from memory or wants to send data to memory, the 8086 sends out the memory address code in the instruction and sends out the appropriate memory read or memory write signal to transfer a word of data.

➢In the case of memory read, the addressed word will be kept on the data bus by the memory. The 8087 then simply reads the word of data bus. The 8086 ignores this word .If the 8087 only needs this one word of data, it can then go on and executes its instruction.

➢Some 8087 instructions need to read in or write out up to 80-bit word. For these cases 8086 outputs the address of the first data word on the address bus and outputs the appropriate control signal.

➢The 8087 reads the data word on the data bus by memory or writes a data word to memory on the data bus. The 8087 grabs the 20-bit physical address that was output by the 8086.To transfer additional words it needs to/from memory, the 8087 then takes over the buses from 8086.

➢To take over the bus, the 8087 sends out a low-going pulse on

$\overline{RQ}/\overline{GT}_0$ pin. The 8086 responds to this by sending another low

going pulse back to the $\overline{RQ}/\overline{GT}_0$ pin of 8087 and by floating its buses.

➢The 8087 then increments the address it grabbed during the first transfer and outputs the incremented address on the address bus. When the 8087 output a memory read or memory write signal, another data word will be transferred to or from the 8087.

➢The 8087 continues the process until it has transferred all the data words required by the instruction to/from memory.

➢When the 8087 is using the buses for its data transfer, it

sends another low-going pulse out on its $\overline{RQ}/\ \overline{GT}_0$ pin to 8086 to know it can have the buses back again.

The next type of the synchronization between the host processor and the coprocessor is that required to make sure the 8086 hast does not attempt to execute the next instruction before the 8087 has completed an instruction.

➢Taking one situation, in the case where the 8086 needs the data produced by the execution of an 8087 instruction to carry out its next instruction.

➢In the instruction sequence for example the 8087 must complete the ***FSTSW STATUS*** instruction before the 8086 will have the data it needs to execute the ***MOV AX , STATUS*** instruction.

➢Without some mechanism to make the 8086 wait until the 8087 completes the FSTSW instruction, the 8086 will go on and execute the ***MOV AX , STATUS*** with erroneous data .

➢We solve this problem by connecting the 8087 BUSY output to the TEST pin of the 8086 and putting on the WAIT instruction in the program.

➢While 8087 is executing an instruction it asserts its BUSY pin high. When it is finished with an instruction, the 8087 will drop its BUSY pin low. Since the BUSY pin from 8087 is connected to the TEST pin 8086 the processor can check its pin of 8087 whether it finished it instruction or not.

➢You place the 8086 WAIT instruction in your program after the 8087 FSTSW instruction .When 8086 executes the WAIT instruction it enters an internal loop where it repeatedly checks the logic level on the TEST input. The 8086 will stay in this loop until it finds the TEST input asserted low, indicating the 8087 has completed its instruction. The 8086 will then exit the internal loop, fetch and execute the next instruction.

Example

```
FSTSW      STATUS    ;copy  8087 status word to memory
MOV        AX, STATUS ;copy status word to AX to check
                     ; bits
```

(a)

➢In this set of instructions we are not using WAIT instruction. Due to this the flow of execution of command will takes place continuously even though the previous instruction had not finished it's completion of its work .so we may lost data .

```
FSTSW      STATUS    ;copy 8087 status word to memory
FWAIT                ;wait for 8087 to finish before-
                     ; doing next 8086 instruction
MOV        AX,STATUS ;copy status word to AX to check
                     ; bits
```

(b)

➢In this code we are adding up of FWAIT instruction so that it will stop the execution of the command until the above instruction is finishes it's work .so that you are not loosing data and after that you will allow to continue the execution of instructions.

➢Another case where you need synchronization of the processor and the coprocessor is the case where a program has several 8087 instructions in sequence.
➢ The 8087 are executed only one instruction at a time so you have to make sure that 8087 has completed one instruction before you allow the 8086 to fetch the next 8087 instruction from memory.
➢Here again you use the BUSY-TEST connection and the $\overline{\text{FWAIT}}$ instruction to solve the problem. If you are hand coding, you can just put the 8086 WAIT(FWAIT) instruction after each instruction to make sure that instruction is completed before going on to next.

➢If you are using the assembler which accepts 8087 mnemonics, the assembler will automatically insert the 8-bit code for the WAIT instruction ,10011011 binary (9BH), as the first byte of the code for 8087 instruction.

INTERFACING

➢Multiplexed address-data bus lines are connected directly from the 8086 to 8087.
➢The status lines and the queue status lines connected directly from 8086 to 8087.

➢The Request/Grant signal $\overline{\text{RQ}}/\overline{\text{GT0}}$ of 8087 is connected to

$\overline{\text{RQ}}/\overline{\text{GT1}}$ of 8086.

➢BUSY signal 8087 is connected to $\overline{\text{TEST}}$ pin of 8086.

➢Interrupt output INT of the 8087 to NMI input of 8086. This intimates an error condition.
➢A WAIT instruction is passed to keep looking at its $\overline{\text{TEST}}$ pin, until it finds pin Low to indicates that the 8087 has completed the computation.

➢SYNCHRONIZATION must be established between the processor and coprocessor in two situations.

a) The execution of an ESC instruction that require the participation of the NUE must not be initiated if the NUE has not completed the execution of the previous instruction.

b) When a processor instruction accesses a memory location that is an operand of a previous coprocessor instruction .In this case CPU must synchronize with NPX to ensure that it has completed its instruction.
Processor WAIT instruction is provided.

Exception Handling

➢The 8087 detects six different types of exception conditions that occur during instruction execution. These will cause an interrupt if unmasked and interrupts are enabled.

1)INVALID OPERATION
2)OVERFLOW
3)ZERO DIVISOR
4)UNDERFLOW
5)DENORMALIZED OPERAND
6)INEXACT RESULT

Data Types

➢Internally, all data operands are converted to the 80-bit temporary real format.

We have 3 types.

•Integer data type
•Packed BCD data type
•Real data type

Coprocessor data types

Integer Data Type

Packed BCD

Real data type

Example

➢Converting a decimal number into a Floating-point number.

1) Converting the decimal number into binary form.
2) Normalize the binary number
3) Calculate the biased exponent.
4) Store the number in the floating-point format.

Example

Step Result

1 100.25
21100100.01 = 1.10010001 * 2^6
3110+01111111=10000101
4 Sign = 0
Exponent =10000101
Significand = 10010001000000000000000

•In step 3 the biased exponent is the exponent a 2^6 or 110,plus a bias of 01111111(7FH) ,single precision no use 7F and double precision no use 3FFFH.
•IN step 4 the information found in prior step is combined to form the floating point no.

INSTRUCTION SET

➢The 8087 instruction mnemonics begins with the letter F which stands for Floating point and distinguishes from 8086.

➢These are grouped into Four functional groups.

➢The 8087 detects an error condition usually called an exception when it executing an instruction it will set the bit in its Status register.

Types

I. DATA TRANSFER INSTRUCTIONS.

II. ARITHMETIC INSTRUCTIONS.

III. COMPARE INSTRUCTIONS.

IV. TRANSCENDENTAL INSTRUCTIONS.
(Trigonometric and Exponential)

Data Transfers Instructions

- **REAL TRANSFER**

FLD	Load real
FST	Store real
FSTP	Store real and pop
FXCH	Exchange registers

- **INTEGER TRANSFER**

FILD	Load integer
FIST	Store integer
FISTP	Store integer and pop

- **PACKED DECIMAL TRANSFER(BCD)**

FBLD	Load BCD
FBSTP	Store BCD and pop

Example

- **FLD Source**- Decrements the stack pointer by one and copies a real number from a stack element or memory location to the new ST.

```
•FLD   ST(3)              ;Copies ST(3) to ST.
•FLD   LONG_REAL[BX]      ;Number from memory
                          ;copied to ST.
```

- **FLD Destination-** Copies ST to a specified stack position or to a specified memory location .

```
•FST          ST(2)              ;Copies ST to ST(2),and
                                 ;increment stack pointer.
•FST          SHORT_REAL[BX]     ;Copy ST to a memory at a
                                 ;SHORT_REAL[BX]
```

➢**FXCH Destination** – Exchange the contents of ST with the contents of a specified stack element.

•FXCH ST(5) ;Swap ST and ST(5)

➢**FILD Source** – Integer load. Convert integer number from memory to temporary-real format and push on 8087 stack.

•FILD DWORD PTR[BX] ;Short integer from memory at [BX].

➢**FIST Destination-** Integer store. Convert number from ST to integer and copy to memory.

•FIST LONG_INT ;ST to memory locations named LONG_INT.

➢**FISTP Destination**-Integer store and pop. Identical to FIST except that stack pointer is incremented after copy.

➢**FBLD Source-** Convert BCD number from memory to temporary- real format and push on top of 8087 stack.

Arithmetic Instructions.

❖Four basic arithmetic functions:

Addition, Subtraction, Multiplication, and
Division.

➢**Addition**

FADD	Add real
FADDP	Add real and pop
FIADD	Add integer

➢**Subtraction**

FSUB	Subtract real
FSUBP	Subtract real and pop
FISUB	Subtract integer

FSUBR	Subtract real reversed
FSUBRP	Subtract real and pop
FISUBR	Subtract integer reversed

➢**Multiplication**

FMUL	Multiply real
FMULP	Multiply real and pop
FIMUL	Multiply integer

➢**Advanced**

FABS	Absolute value
FCHS	Change sign
FPREM	Partial remainder
FPRNDINT	Round to integer
FSCALE	Scale
FSQRT	Square root
FXTRACT	Extract exponent and mantissa.

Example

➢**FADD** – Add real from specified source to specified destination Source can be a stack or memory location. Destination must be a stack element. If no source or destination is specified, then ST is added to ST(1) and stack pointer is incremented so that the result of addition is at ST.

•FADD ST(3), ST ;Add ST to ST(3), result in ST(3)
•FADD ST,ST(4) ;Add ST(4) to ST, result in ST.
•FADD ;ST + ST(1), pop stack result at ST
•FADDP ST(1) ;Add ST(1) to ST. Increment stack
;pointer so ST(1) become ST.
•FIADD Car_Sold ;Integer number from memory + ST

➢**FSUB -** Subtract the real number at the specified source from the real number at the specified destination and put the result in the specified destination.

•FSUB ST(2), ST ;ST(2)=ST(2) – ST.
•FSUB Rate ;ST=ST – real no from memory.
•FSUB ;ST=(ST(1) – ST)

➢**FSUBP** - Subtract ST from specified stack element and put result in specified stack element .Then increment the pointer by one.

•FSUBP ST(1) ;ST(1)-ST. ST(1) becomes new ST

➢**FISUB** – Integer from memory subtracted from ST, result in ST.
•FISUB Cars_Sold ;ST becomes ST – integer from memory

Compare Instructions.

➢**Comparison**

FCOM	Compare real
FCOMP	Compare real and pop
FCOMPP	Compare real and pop twice
FICOM	Compare integer
FICOMP	Compare integer and pop
FTST	Test ST against +0.0
FXAM	Examine ST

Transcendental Instruction.

➢**Transcendental**

FPTAN	Partial tangent
FPATAN	Partial arctangent
F2XM1	2^x - 1
FYL2X	Y $\log_2$X
FYL2XP1	Y $\log_2$(X+1)

Example

➢**FPTAN –** Compute the values for a ratio of Y/X for an angle in ST. The angle must be in radians, and the angle must be in the range of $0 < \text{angle} < \pi/4$.➢**F2XM1 –** Compute $Y=2^x-1$ for an X value in ST. The result Y replaces X in ST. X must be in the range $0 \le X \le 0.5$.

➢**FYL2X -** Calculate Y(LOG_2X).X must be in the range of $0 < X < \infty$ any Y must be in the range $-\infty < Y < +\infty$.

➢**FYL2XP1 –** Compute the function Y(LOG$_2$(X+1)).This instruction is almost identical to FYL2X except that it gives more accurate results when compute log of a number very close to one.

Constant Instructions.

➢**Load Constant Instruction**

FLDZ	Load +0.0
FLDI	Load+1.0
FLDPI	Load π
FLDL2T	Load $\log_2 10$
FLDL2E	Load $\log_2 e$
FLDLG2	Load $\log_{10} 2$
FLDLN2	Load $\log_e 2$

ALGORITHM

To calculate x to the power of y

- Load base, power.
- Compute (y)*($\log_2$ x)
- Separate integer(i) ,fraction(f) of a real number
- Divide fraction (f) by 2
- Compute $({}_{2}{}^{f/2}) * ({}_{2}{}^{f/2})$
- $x^y = ({}_{2}{}^{x}) * ({}_{2}{}^{y})$

Program:

Program to calculate x to the power of y

```
.MODEL SMALL
.DATA
```

```
        x               Dq      4.567   ;Base
        y               Dq      2.759   ;Power
        temp    DD
        temp1   DD
        temp2   DD              ;final  real result
        tempint         DD
        tempint1        DD      ;final integer result
        two             DW
        diff            DD
        trunc_cw        DW      0fffh


                        .STACK 100h
                        .CODE
start:          mov ax, @DATA   ;init data segment
                        mov ds, ax
•
load:           fld y                   ;load the power
                fld x                   ;load the base
•
comput:         fyl2x                   ;compute (y * log2(x))
                fst temp                ;save the temp result



        trunc:  fldcw  trunc_cw         ;set truncation command
                        frndint
                        fld  temp       ;load real number of fyl2x
                        fist  tempint   ;save integer after truncation
                        fld  temp       ;load the real number
•
   getfrac:     fisub  tempint           ;subtract the integer
                fst   diff              ;store the fraction

 fracby2:       fidiv two               ;divide the fraction by 2

twopwrx:                f2xm1            ;calculate the 2 to the power fraction
                        fst temp1       ;minus 1 and save the result
                        fld1            ;load1
                        fadd            ;add 1 to the previous result
                        fst temp1       ;save the result


sqfrac:                 fmul st(0),st(0)   ;square the result as fraction
                        fst temp1          ;was halved and save the result
                        fild tempint       ;save the integer portion
```

```
                               fxch        ;interchange the integer
                                           ;and power of fraction.


scale:                 fscale              ;scale the result in real  and
                                                 ;integer
                       fst temp2           ;in st(1) and store
                       fist tempint1      ;save the final result in real and integer


      over:            mov ax,4c00h       ;exit to dos
                       int 21h
                       end start
```

Microcontroller

Prof. M.Krishna Kumar

Contents

- Introduction
- Inside 8051
- Instructions
- Interfacing

Introduction

- Definition of a Microcontroller
- Difference with a Microprocessor
- Microcontroller is used where ever

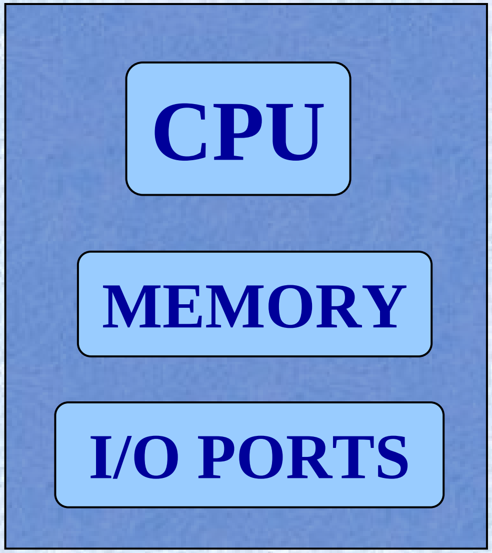
Definition

- It is a single chip
- Consists of Cpu, Memory
- I/O ports, timers and other peripherals

Difference

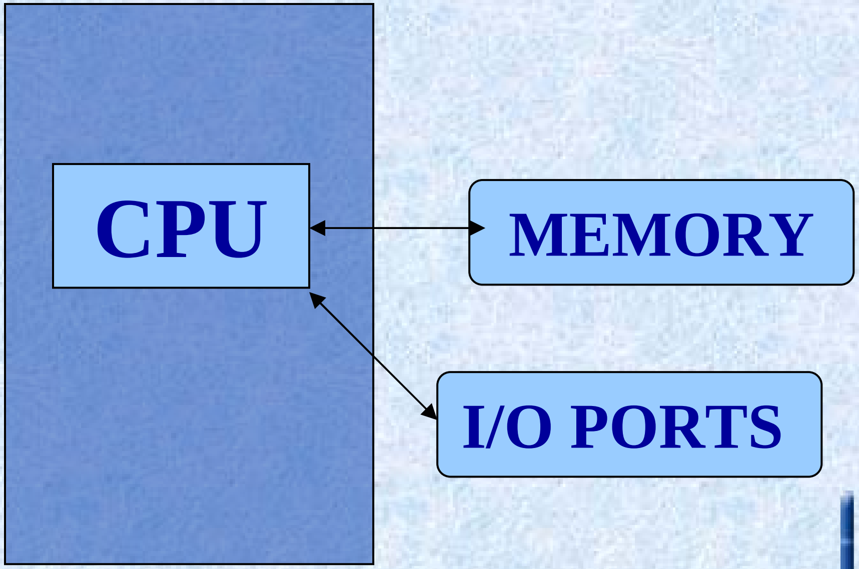
MICRO CONTROLLER

- It is a single chip
- Consists Memory,
- I/o ports



MICRO PROCESSER

- It is a cpu
- Memory, I/O Ports to be connected externally



Where ever

- Small size
- Low cost
- Low power

Architecture

- Harvard university

The Architecture given by Harvard University has the following advantages:

1: Data Space and Program Space are distinct

2: There is no Data corruption or loss of data

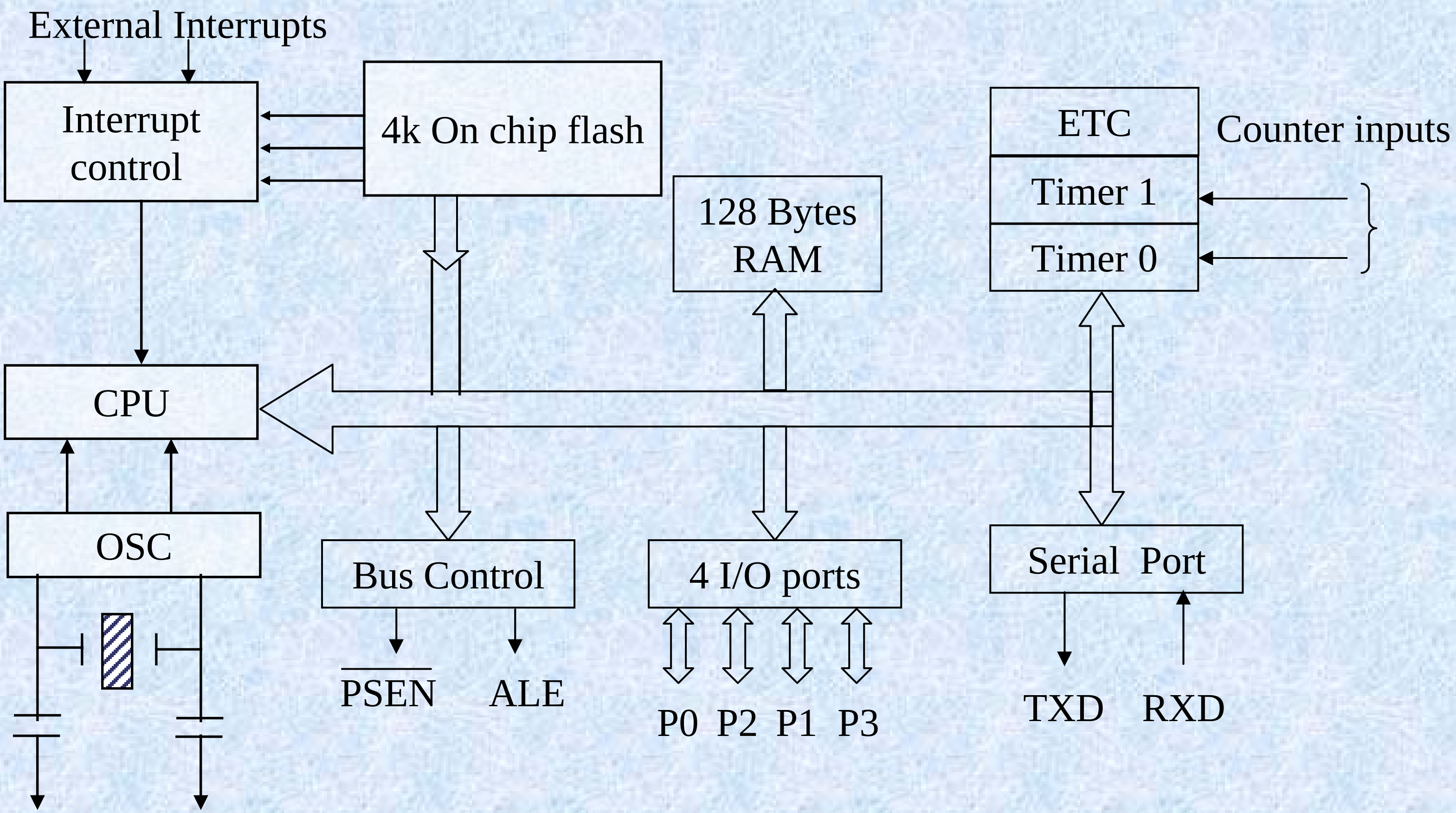
Disadvantage is:

1: The circuitry is very complex.

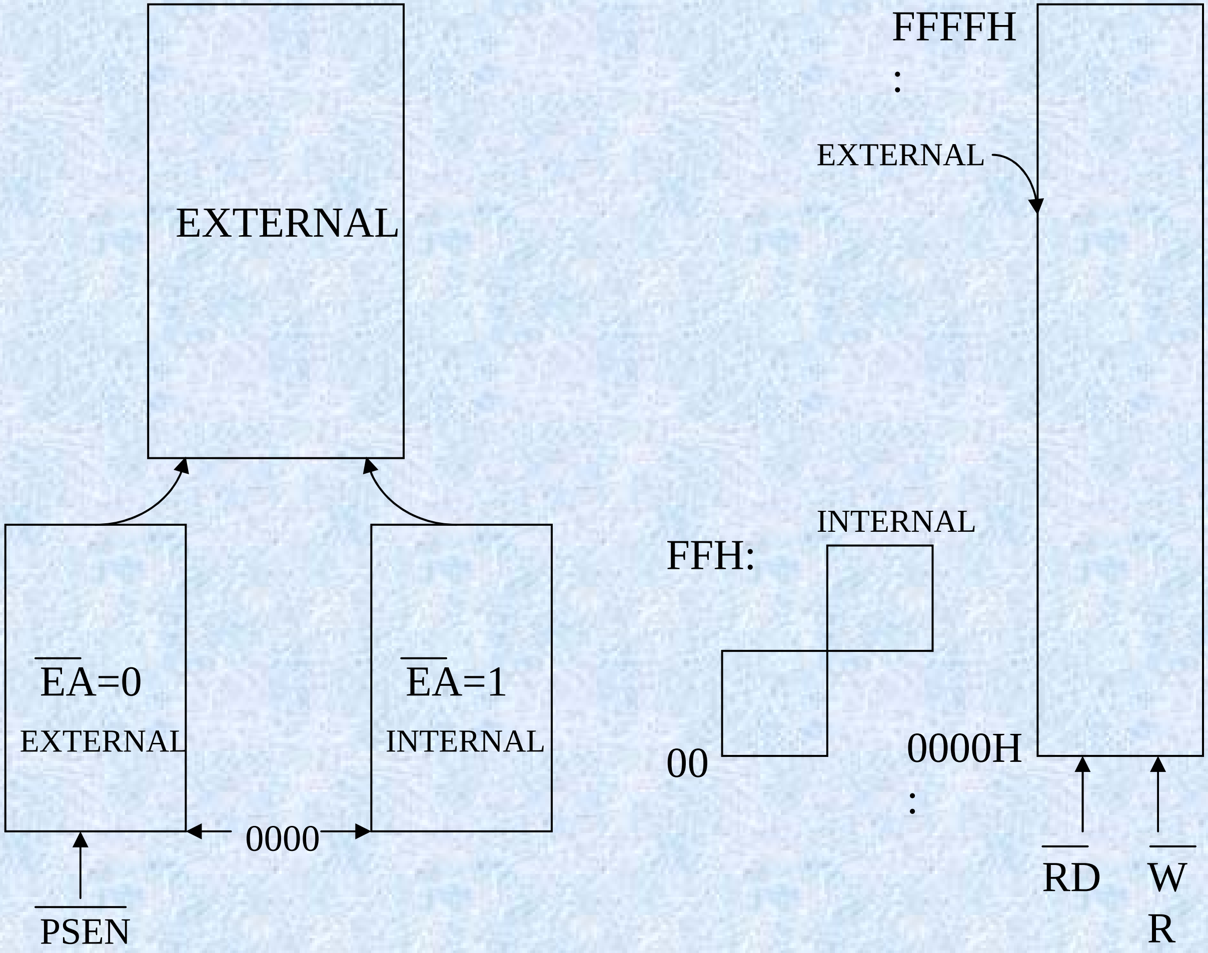
Features

- 8 bit cpu
- 64k Program memory (4k on chip)
- 64k Data memory
- 128 Bytes on chip
- 32   I/O
- Two 16 bit timers
- Full duplex UART
- 6 Source/5 Vector interrupts with two level priority levels
- On chip clock Oscillator.

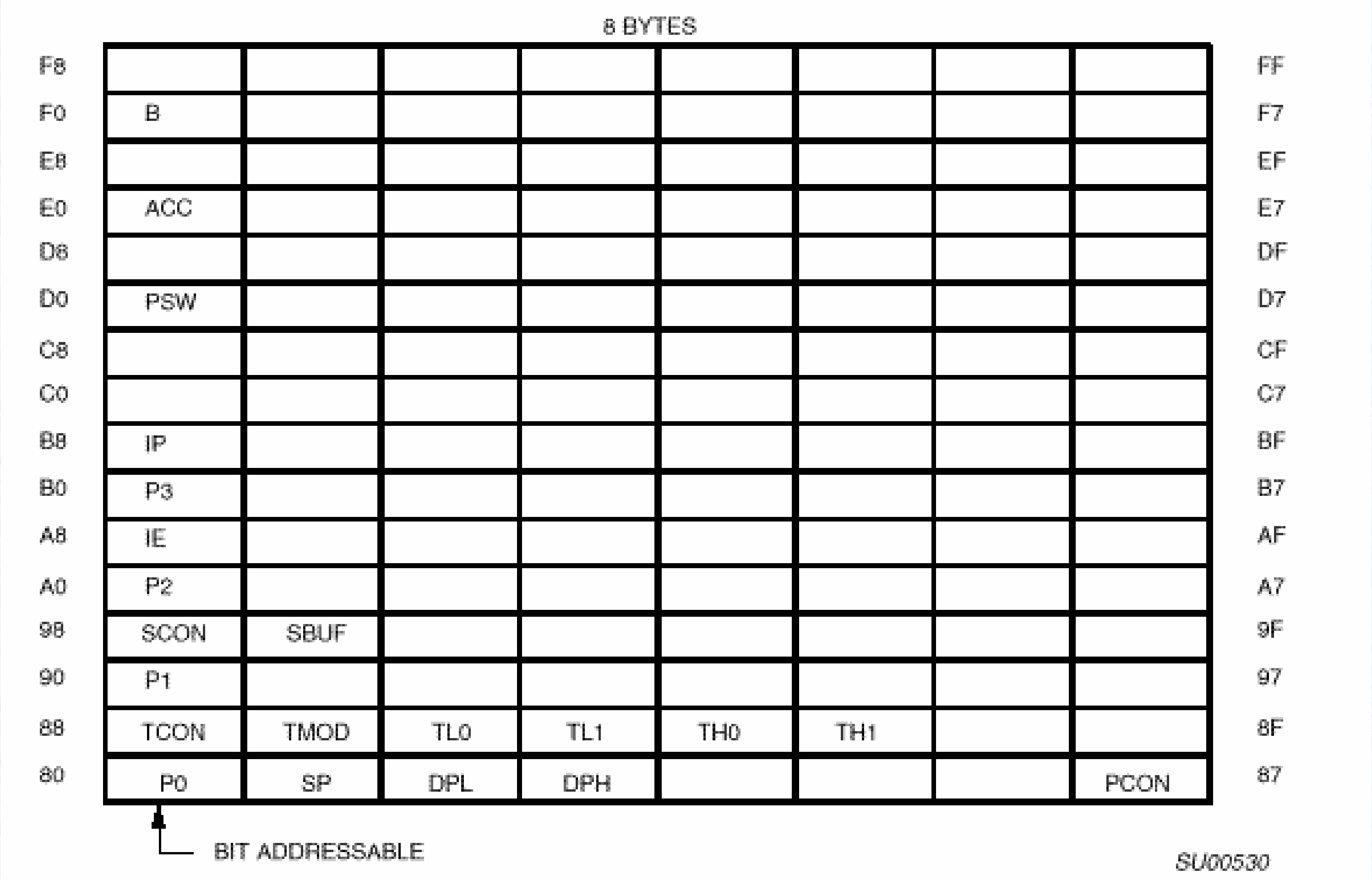
Block Diagram



Memory Architecture



SFR Map



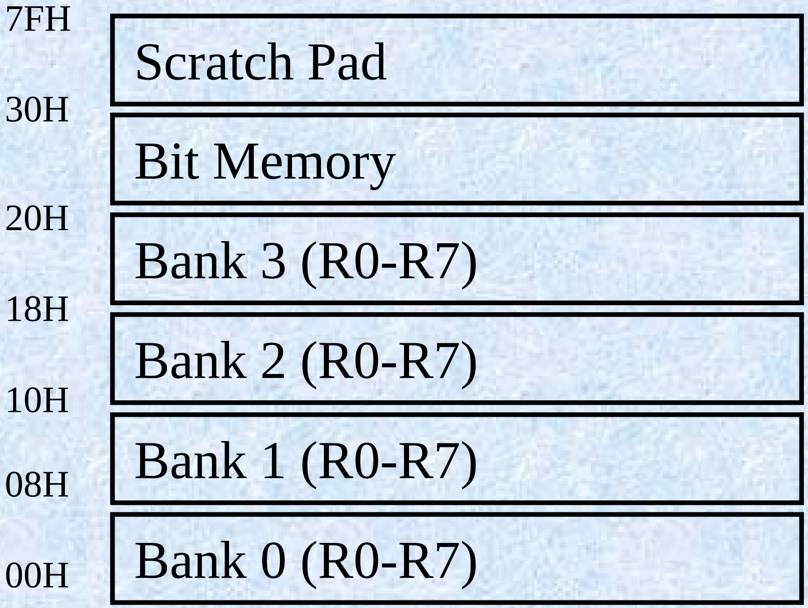
8 BYTES

F8									FF
F0	B								F7
E8									EF
E0	ACC								E7
D8									DF
D0	PSW								D7
C8									CF
C0									C7
B8	IP								BF
B0	P3								B7
A8	IE								AF
A0	P2								A7
98	SCON	SBUF							9F
90	P1								97
88	TCON	TMOD	TL0	TL1	TH0	TH1			8F
80	P0	SP	DPL	DPH				PCON	87

BIT ADDRESSABLE

SU00530

Internal Memory



7FH

Scratch Pad

30H

Bit Memory

20H

Bank 3 (R0-R7)

18H

Bank 2 (R0-R7)

10H

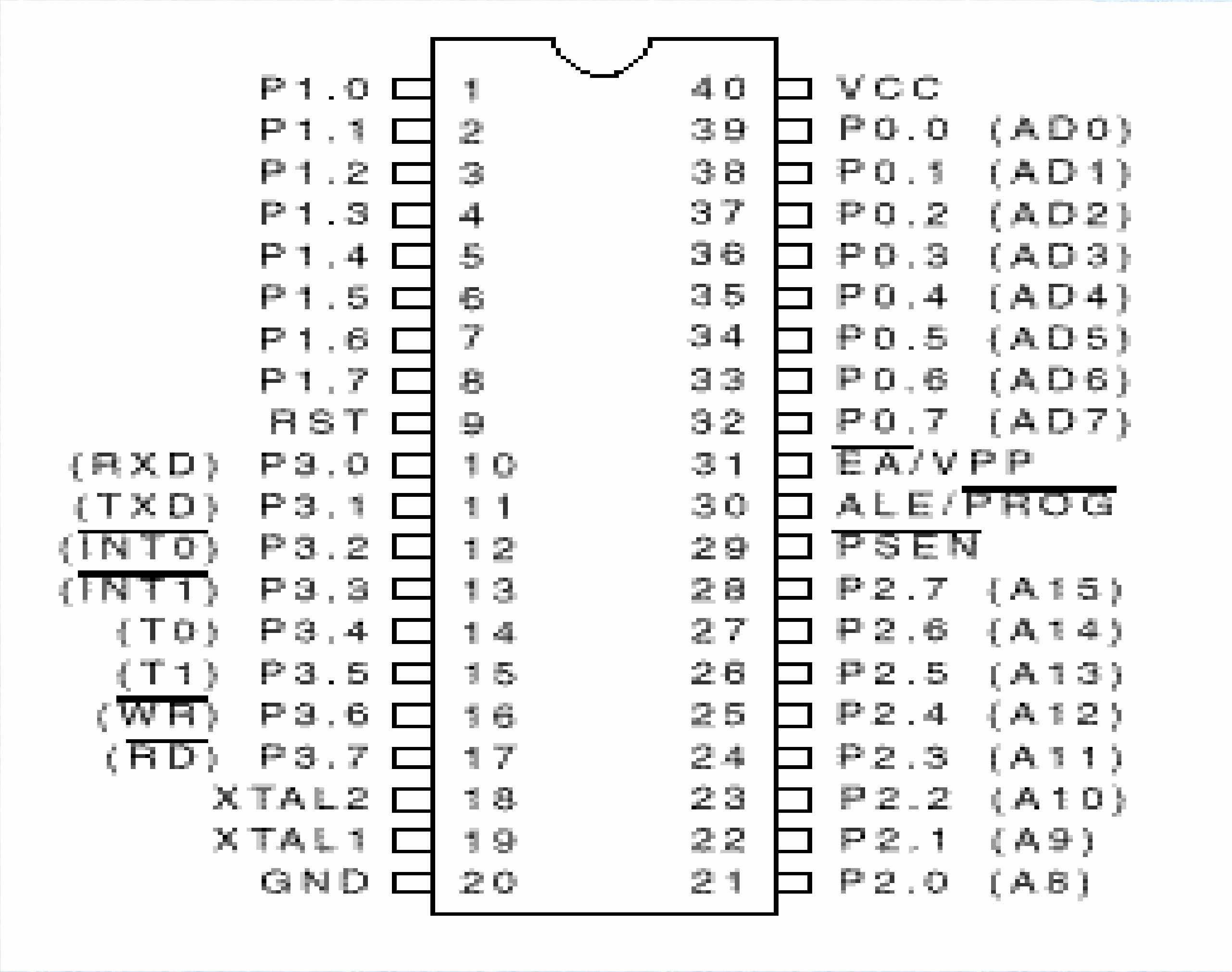
Bank 1 (R0-R7)

08H

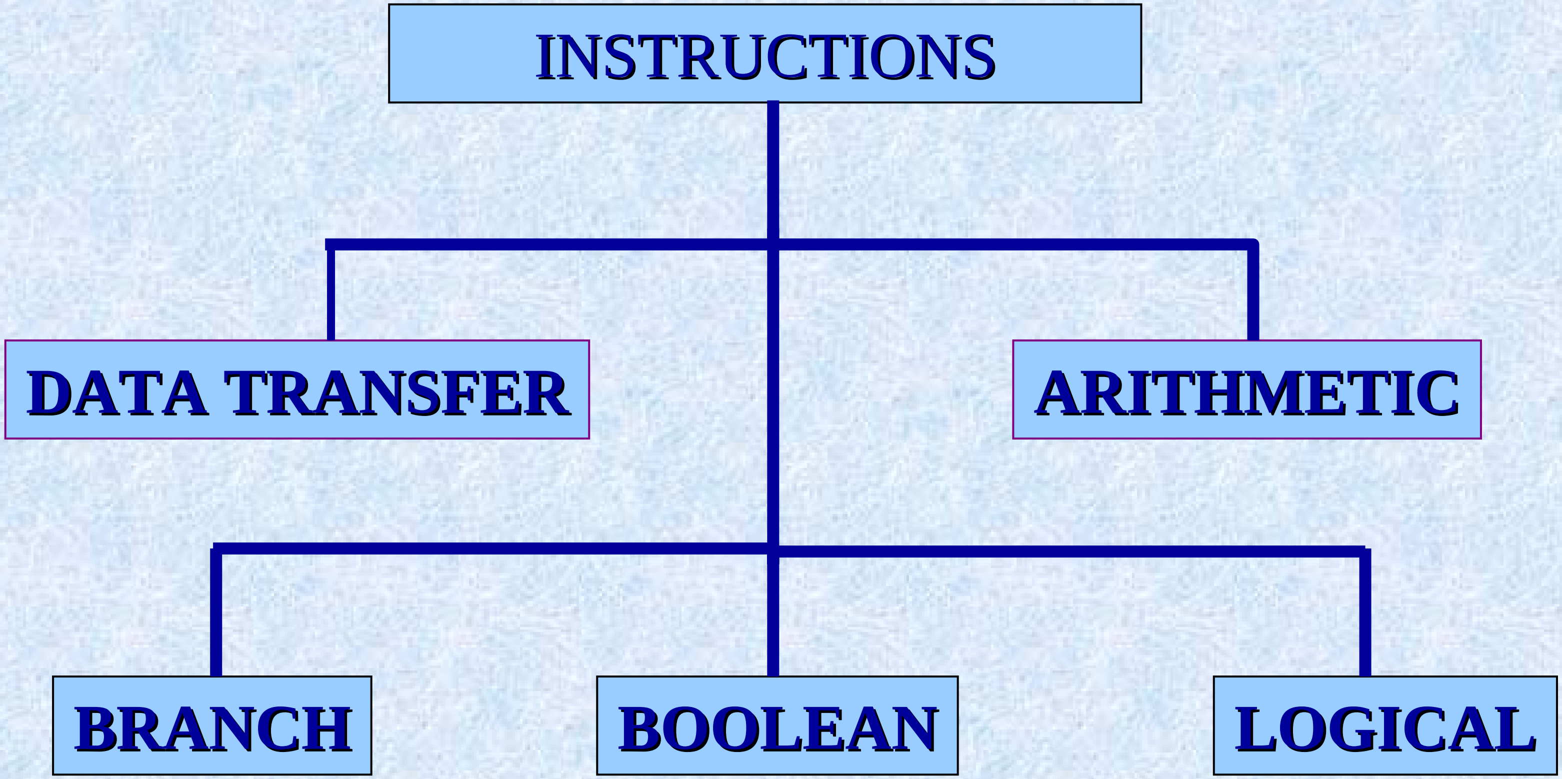
Bank 0 (R0-R7)

00H

Pin connections



Classification of Instructions



INSTRUCTIONS

DATA TRANSFER

ARITHMETIC

BRANCH

BOOLEAN

LOGICAL

Data transfer Instructions

- Mov A, Rn
- Mov A, Direct
- Mov A, @Ri
- Mov A, #Data$_8$
- Mov Dptr, #Data$_{16}$

Data transfer Instructions contd

- Mov Rn, A
- Mov Rn, Direct
- Mov Rn, #Data$_{\mathbf{8}}$
- Mov Direct, A
- Mov Direct, Rn
- Mov Direct, #Data$_8$
- Mov Direct, Direct

Data Transfer Instructions contd

- Mov Direct, @Ri
- Mov Direct, # Data$_8$
- Mov @Ri, A
- Mov @Ri, Direct
- Mov @Ri, #Data$_8$

Data Transfer Instructions contd

- Movx A, @Ri
- Movx A, @Dptr
- Movx @Ri, A
- Movx @dptr, A
- Movc A, @A+Dptr
- Movc A, @A+Pc

Data transfer Instructions contd

- Push Direct
- Pop Direct
- Xch A, Rn
- Xch A, Direct
- Xch A, @Ri
- Xchd A, @Ri

Boolean Instructions

- Clr C
- Clr Bit
- Setb C
- Setb Bit
- Cpl C
- Cpl Bit

Boolean Instructions contd

- Anl C, Bit
- Anl C, /Bit
- Orl C, Bit
- Orl C, /Bit
- Mov C, Bit
- Mov Bit, C

Branch Instructions contd

- Jc Reladdr
- Jnc Reladdr
- Jb Bit, Reladdr
- Jnb Bit, Reladdr
- Jbc Bit, Reladdr

Arithmetic Instructions

- Add A, Rn
- Add A, Direct
- Add A, @Ri
- Add A, #Data$_8$

Arithmetic Instructions contd

- Addc A, Rn
- Addc A, Direct
- Addc A, @Ri
- Addc A, #Data$_8$

Arithmetic Instructions contd

- Subb A, Rn
- Subb A, Direct
- Subb A, @Ri
- Subb A, #Data$_8$

Arithmetic Instructions contd

- Inc A
- Inc Rn
- Inc Direct
- Inc @Ri
- Inc Dptr

Arithmetic Instructions contd

- Dec A
- Dec Rn
- Dec Direct
- Dec @Ri

Arithmetic Instructions contd

- Mul AB
- Div AB
- DA A

Logical Instructions

- Anl A, Rn
- Anl A, Direct
- Anl A, @Ri
- Anl A, #Data$_8$
- Anl Direct, A
- Anl Direct, #Data$_8$

Logical Instructions contd

- Orl A, Rn
- Orl A, Direct
- Orl A, @Ri
- Orl A, #Data$_8$
- Orl Direct, A
- Orl Direct, #Data$_8$

Logical Instructions contd

- Xrl A, Rn
- Xrl A, Direct
- Xrl A, @Ri
- Xrl A, #Data$_8$
- Xrl Direct, A
- Xrl Direct, #Data$_8$

Logical Instructions contd

- Clr A
- Cpl A
- Rl A
- Rlc A
- Rr A
- Rrc A
- Swap A

Branch Instructions

- Acall Addr11
- Lcall Addr16
- Ret
- Reti
- Ajmp Addr11
- Ljmp Addr16
- Sjmp Reladdr

Branch Instructions

- Jmp @A+Dptr
- Jz   Reladdr
- Jnz Reladdr

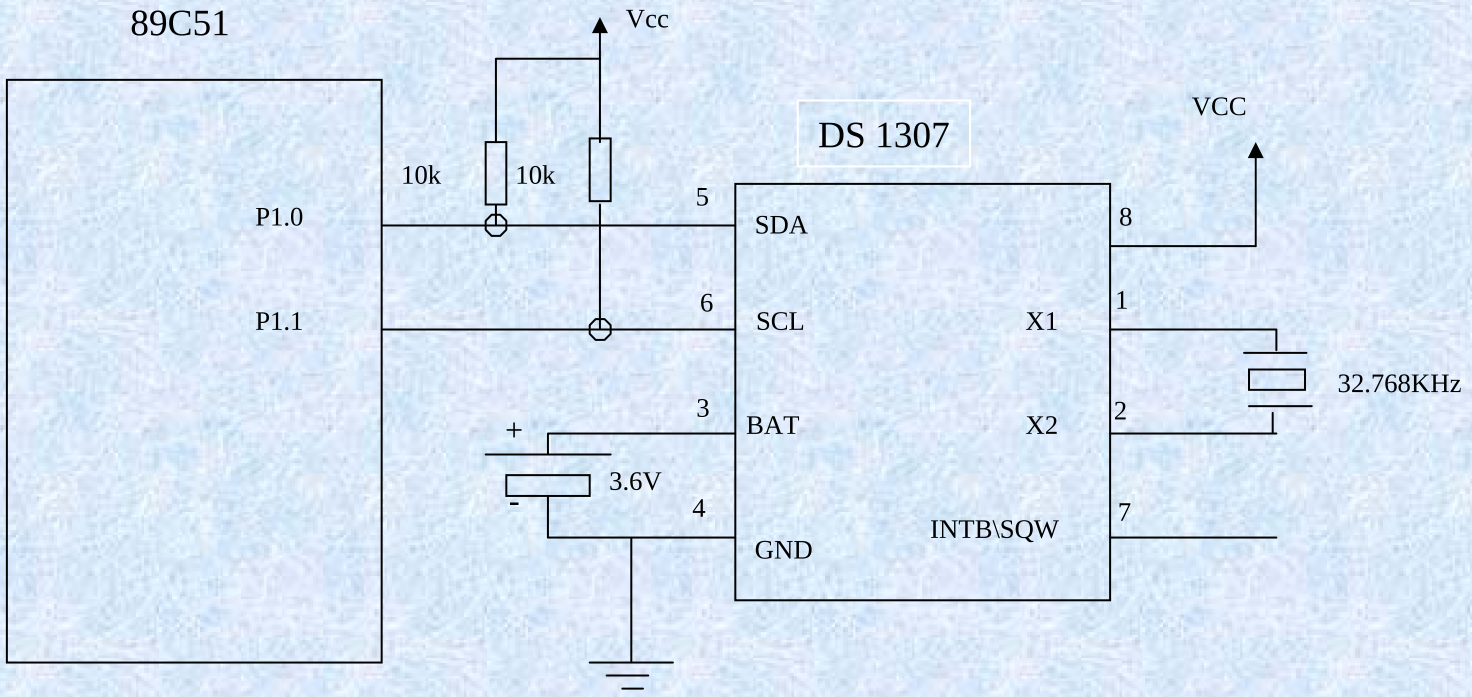
Branch Instructions contd

- Cjne Rn, #Data, Reladdr
- Cjne @Ri, #Data, Reladdr
- Cjne A, #Data, Reladdr
- Cjne A, Direct, Reladdr

Branch Instructions contd

- Djnz Rn, Reladdr
- Djnz Direct, Reladdr
- Nop

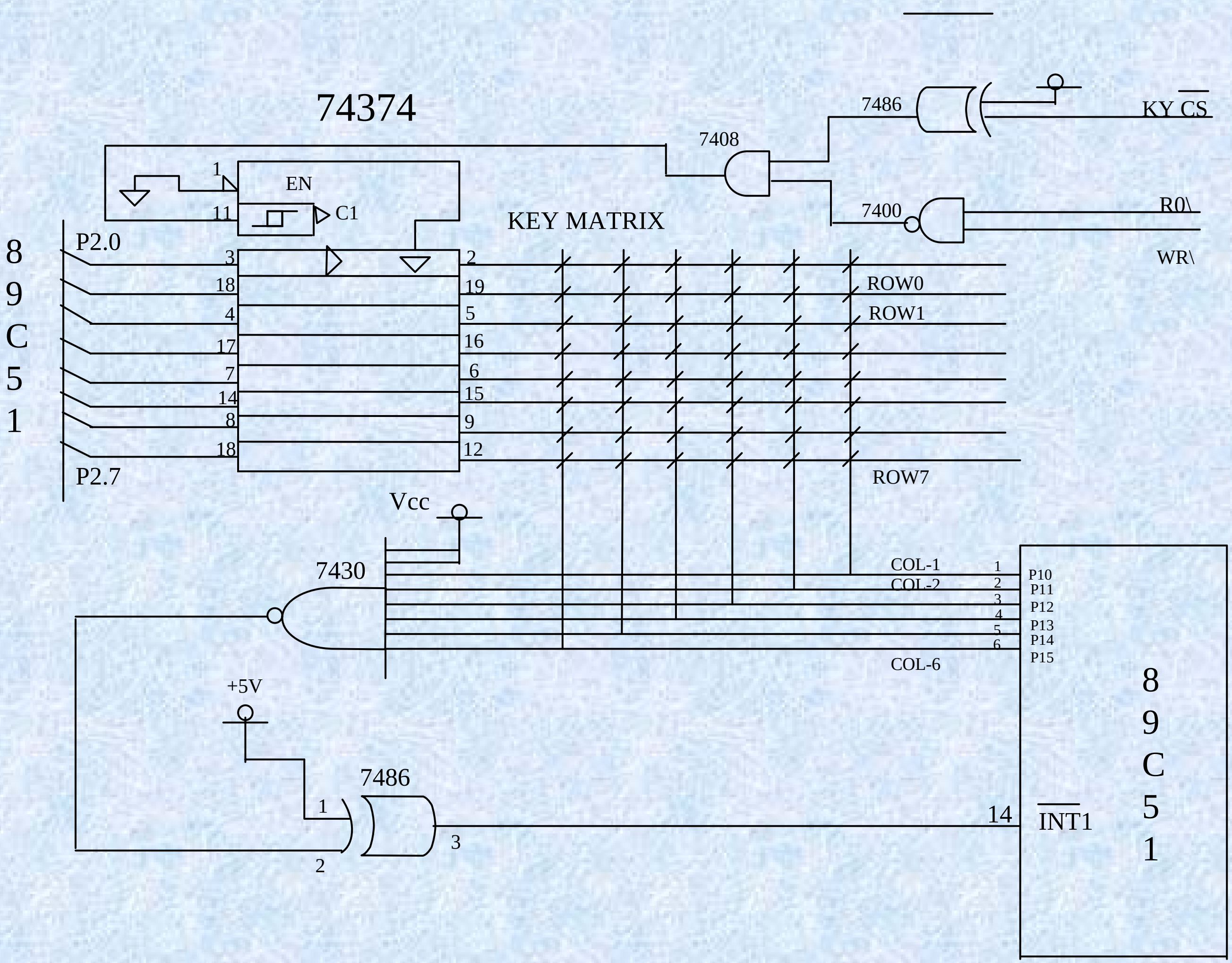
RTC Interface



RTC Interface contd….

- DS 1307 is a real time clock chip
- Maintains real time clock once powered up Year, Month, Day , Time in hours, Minutes and seconds can be written into or read out serially
- Has 56 bytes of data space to save or retrieve data of importance like settings
- Consumes very low power2or 3 uw @ 32.768KHz with a backup battery of 2.5 to 3.6V
- Has SDA, SCL pins to send data and clock respectively
- SDA, SCL are directly interfaced to I/O pins of 89C51

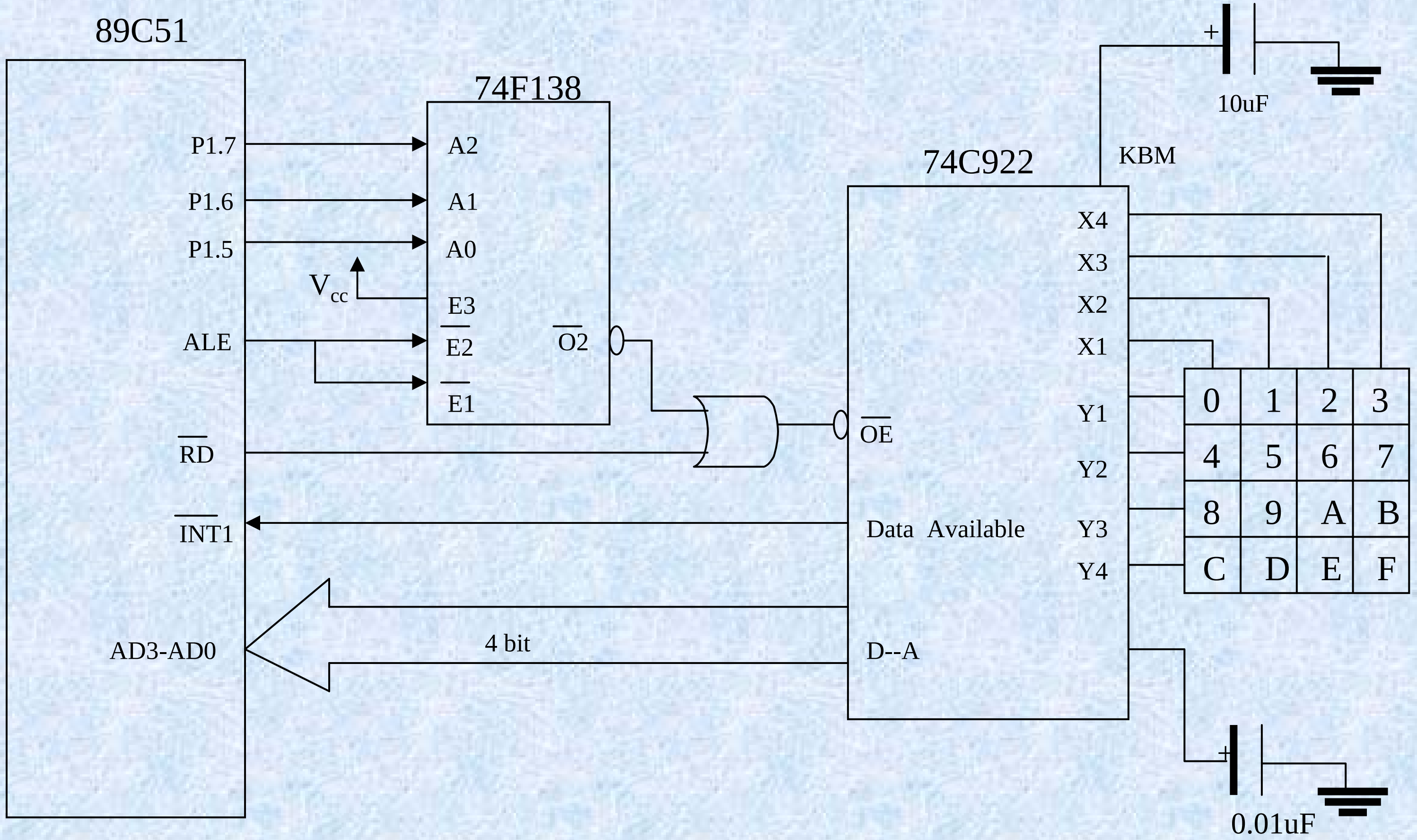
Keyboard Interface-1



Keyboard Interface-1 contd…

- Outputs 8 bit row code (0FEH, 0FDH etc.,) on port0
- Interrupts micro-controller when a key is pressed
- Interrupt software to find which column and key is pressed

Keyboard Interface -2

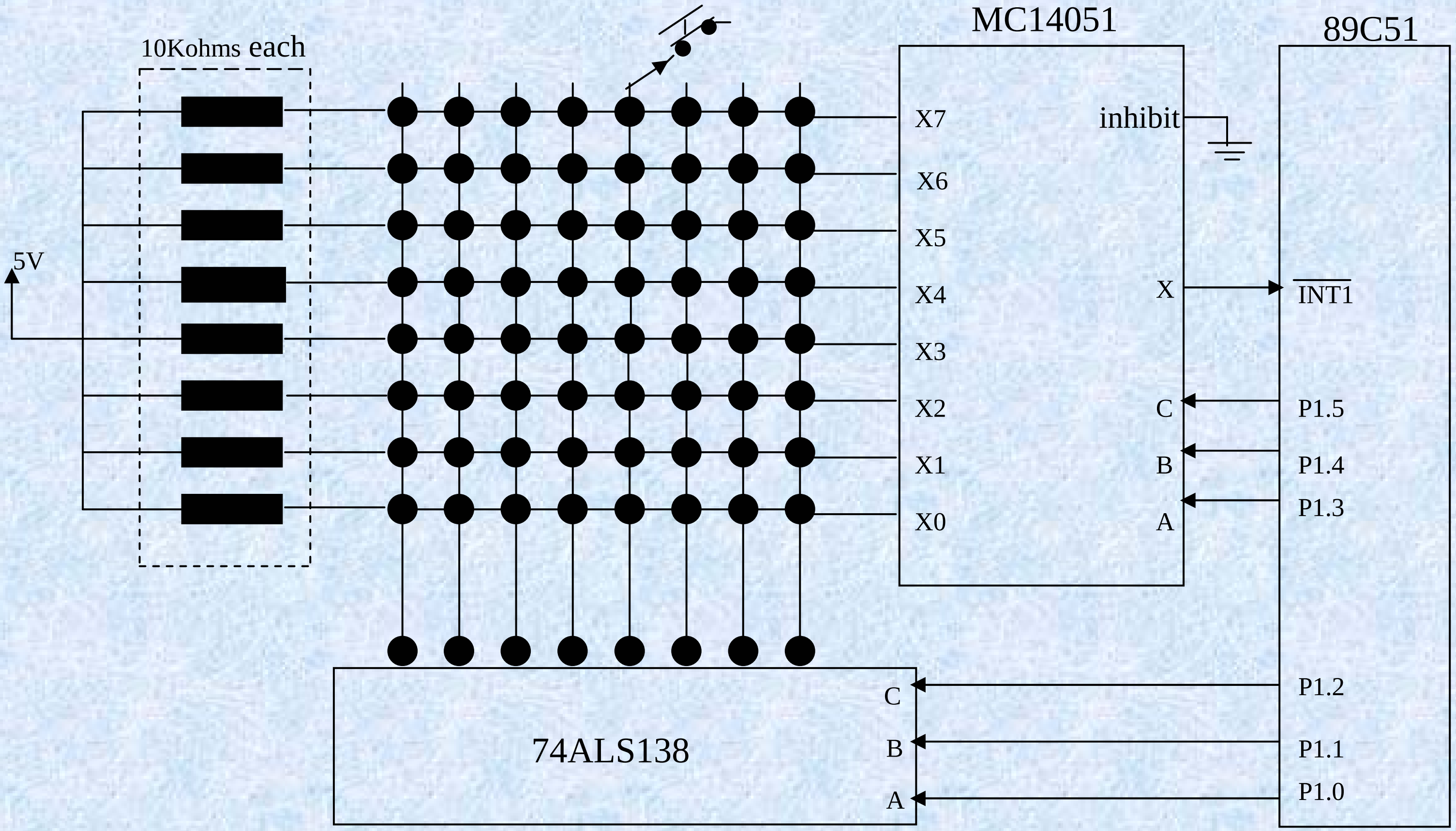


0	1	2	3
4	5	6	7
8	9	A	B
C	D	E	F

Keyboard Interface-2 contd…

- 74C922 is a 16 key encoder that performs keypad scanning and de-bouncing
- When key is pressed it outputs a 4 bit code
- When interfaced to micro-controller, it reads the code through its port pins
- Has key De-bouncing and key mask features
- It has a data available output that interrupts the micro-controller
- Interrupt software to find the key pressed

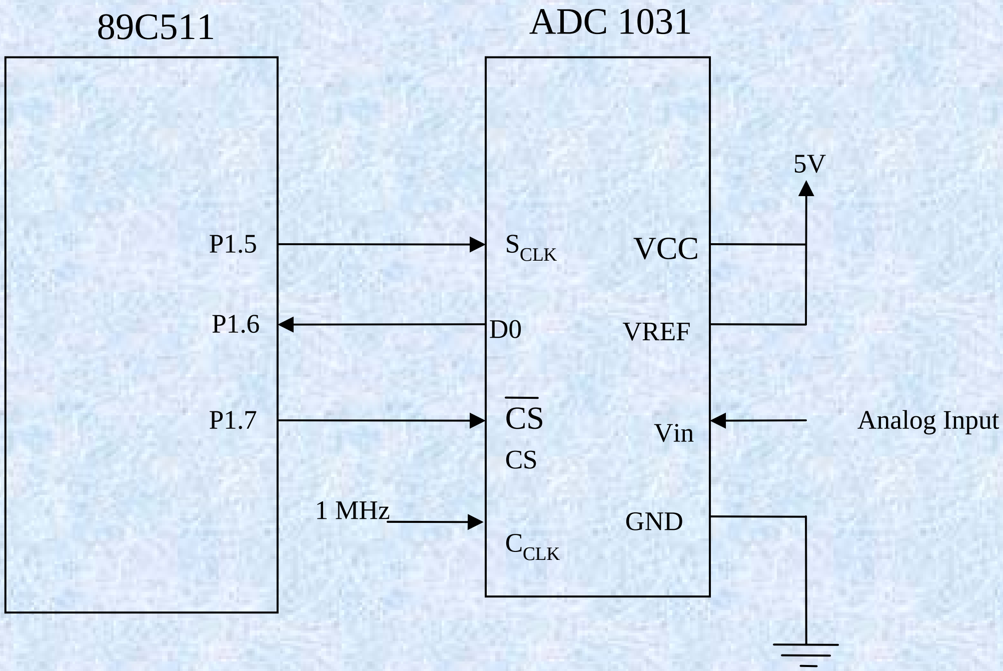
Keyboard Interface-3



Keyboard Interface-3 contd…

- The circuit interfaces 64 keys
- It consists of 14051 a 8:1 multiplexer and 74138 a 3:8 decoder
- When a key is pressed 89C51 is interrupted
- The 3 bit input of multiplexer and 3 bit input of the decoder gives the key code
- which is read in the interrupt routine

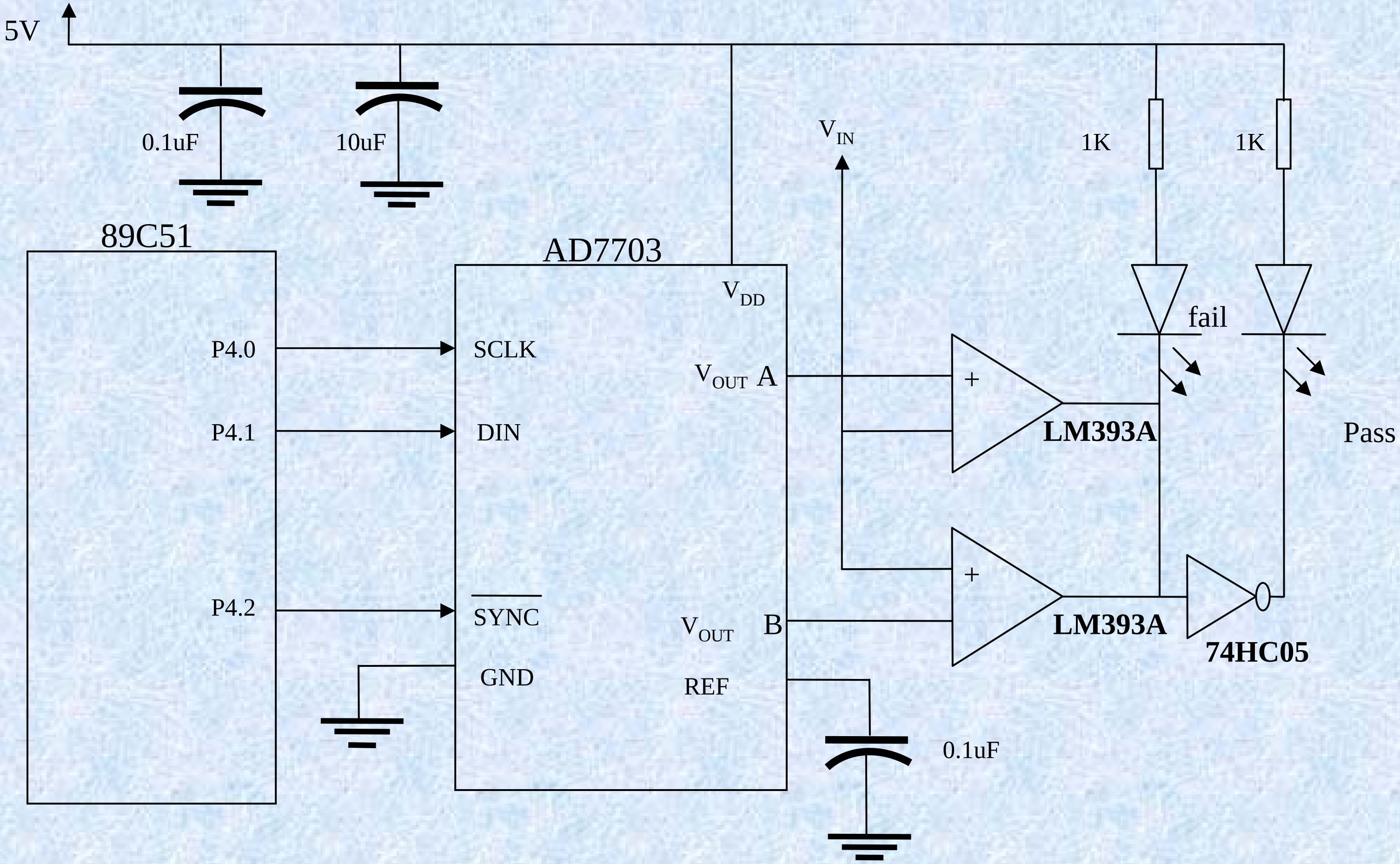
Serial ADC Interface



Serial ADC Interface contd…

- ADC1031 from National semiconductor is a 10 bit ADC
- with Serial interface
- Conversion time is 13.7 us @ 3MHz.
- Conversion starts as soon as CS is enabled
- External clock 1MHz is connected to C_{CLK}

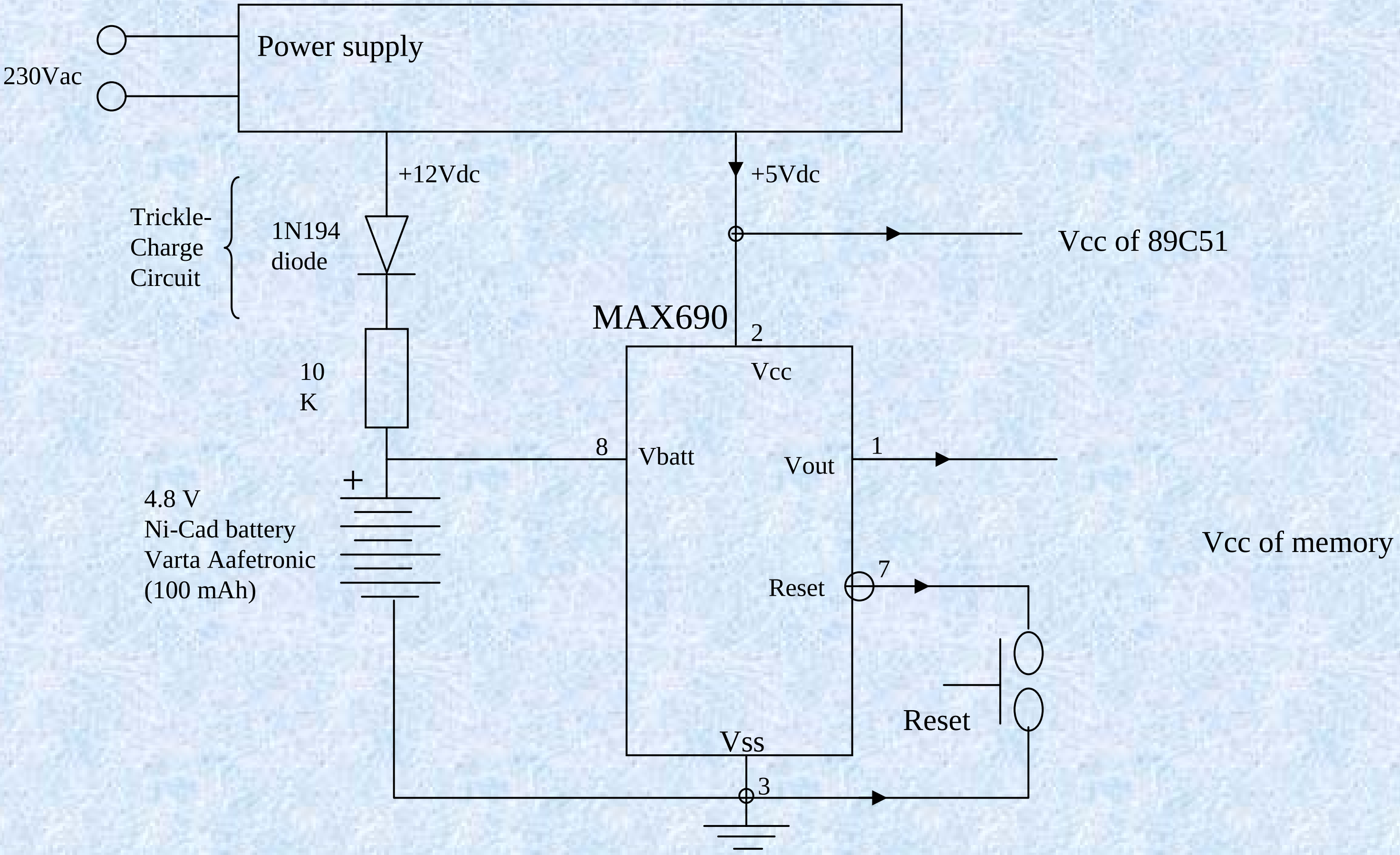
Serial DAC Interface



Serial DAC Interface contd…

- AD7303 is dual channel 8 bit DAC
- Has 16 bit input registers, 8 bit for data and 8 bit for control
- Out put voltage = $\dfrac{2 \text{ x } V_{ref} \text{ x } N}{255}$
- Interface is shown to realize window detector
- If the data is between upper limit and lower limit pass LED glows else fail LED glows

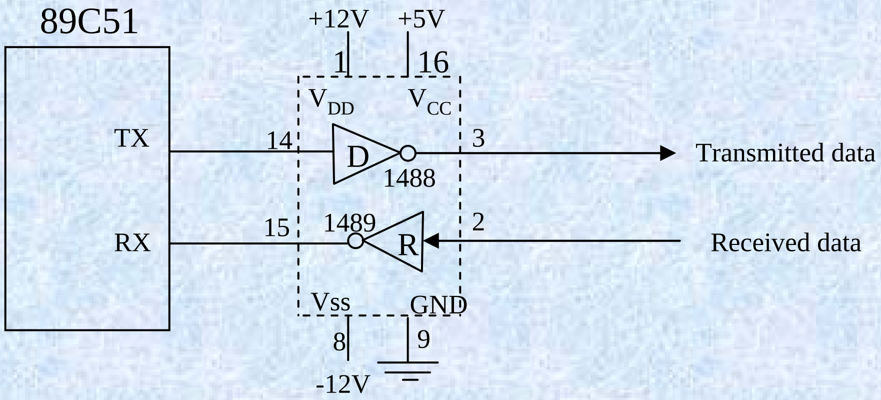
Battery Backup

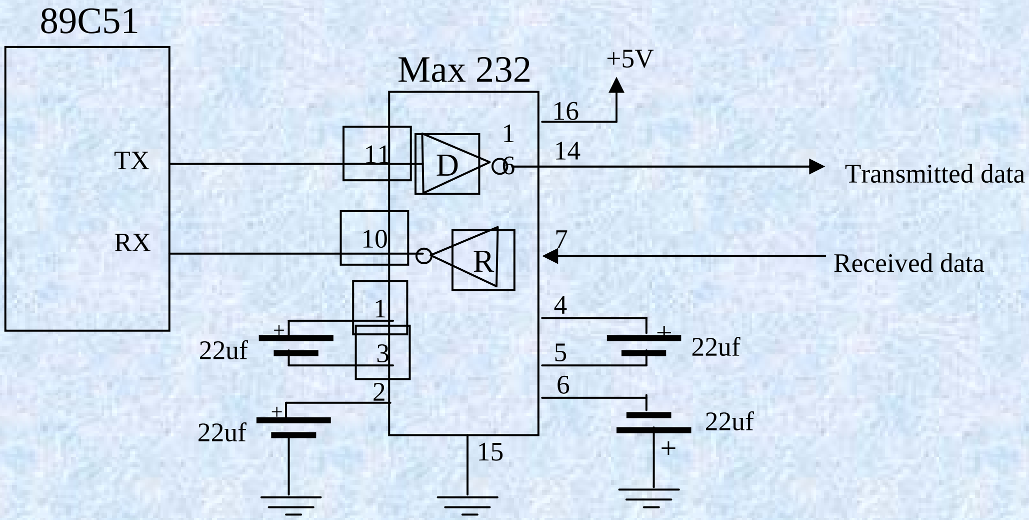


Battery Backup

- Max 690 is a battery switchover/reset generator chip
- It provides a voltage thresh hold mechanism for bringing the chip out of reset at startup and for returning it to reset at power down
- The reset out is connected to the reset pin of 89C51 through an inverter
- V_{OUT} is connected to the Vcc of any memory chip which requires battery back up

Serial Interface

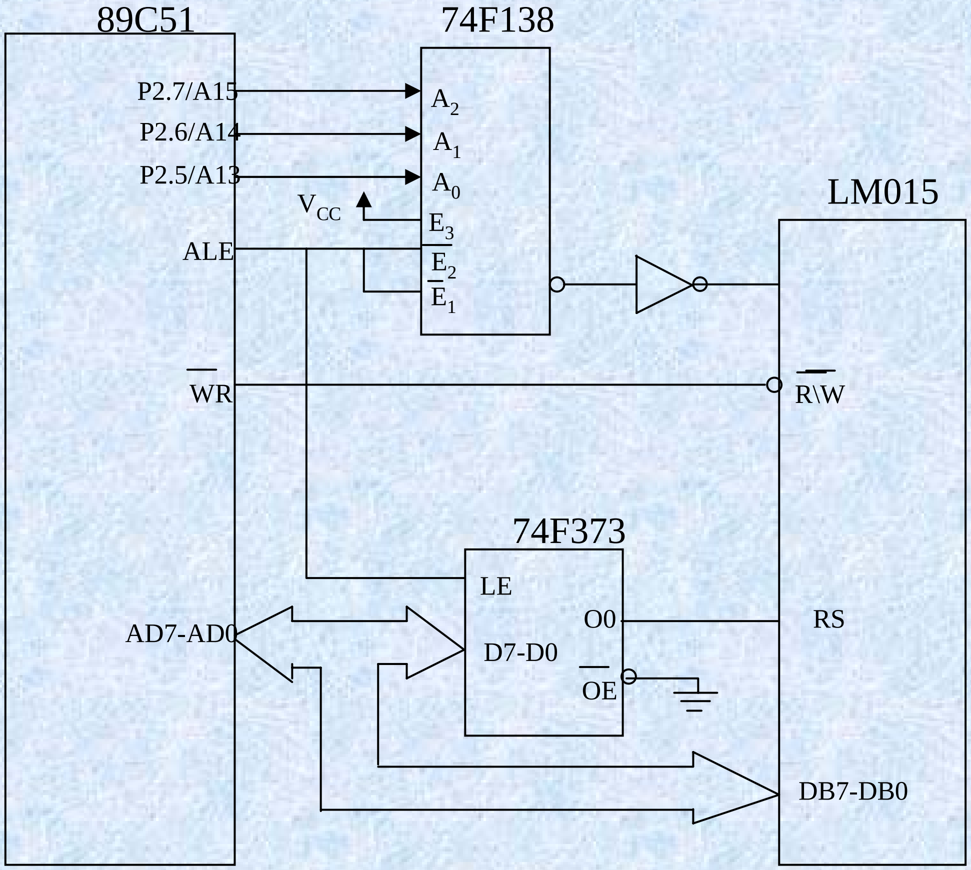




Serial Interface

- RS 232 interface can be realised with 1488 (transmitter) and 1489 (receiver) level translator Ics
- These ICs require +\- 12V supplies
- Max 232 IC require only 5V and four external capacitors

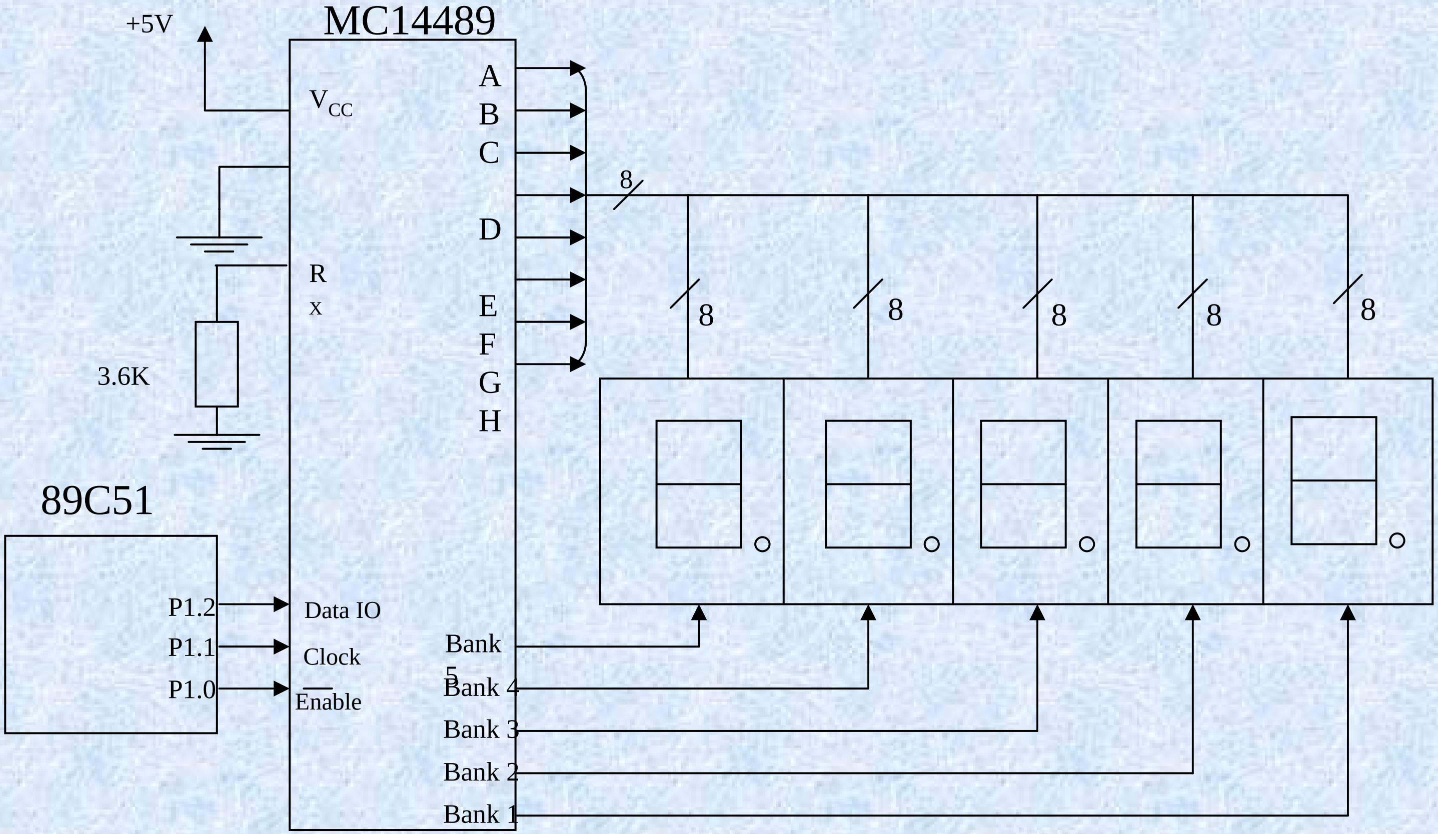
LCD Interface



LCD Interface

- LCD module LM015 displays one line of 16 characters.
- LM015 is initialized with some command words through its control register
- The data to be displayed is written into its data register in ASCII format
- RS pin distinguishes the control and data registers when E is logic high

LED Interface-1



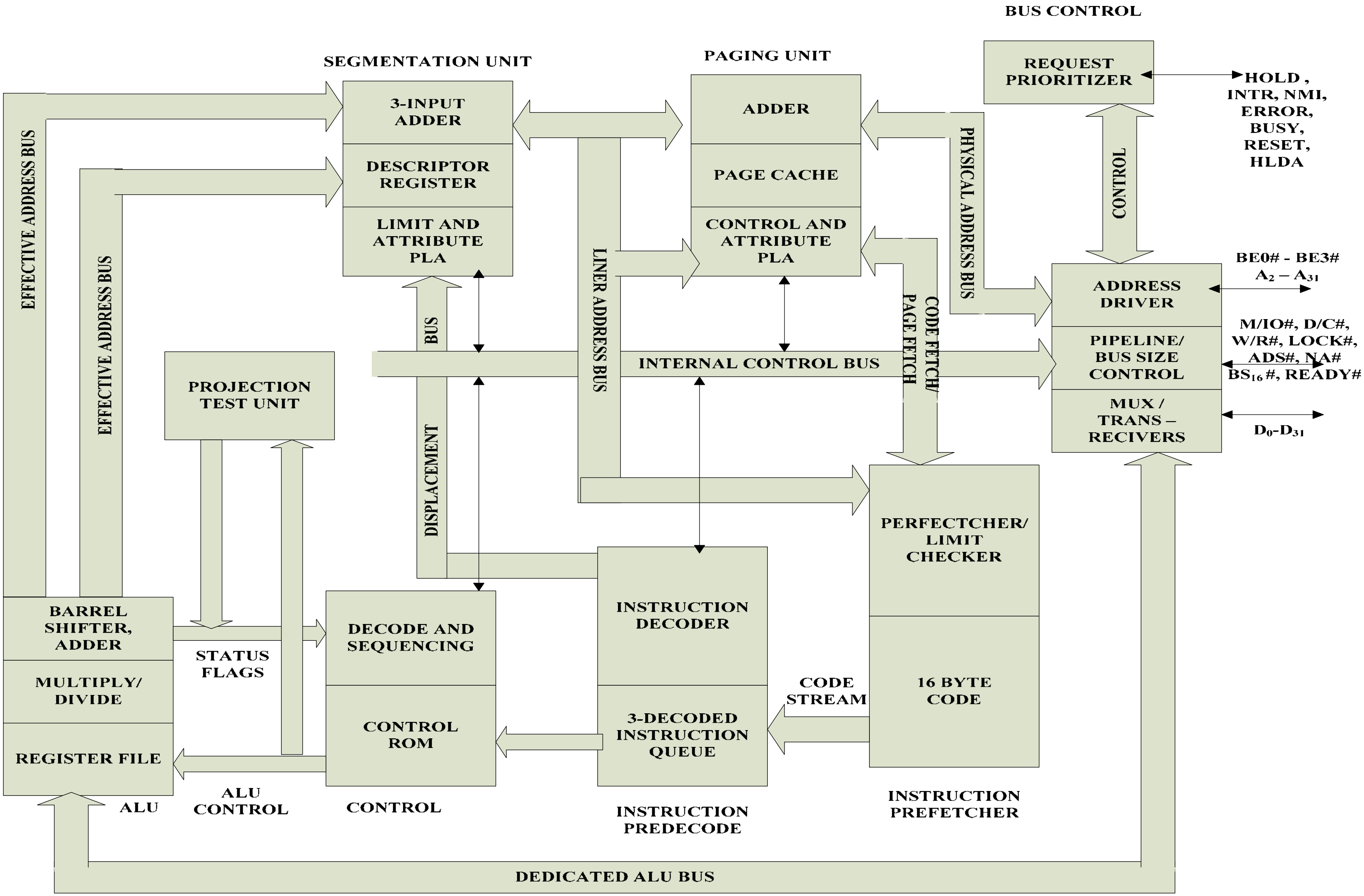
LED Interface-1

- MC14489 is a multi character LED driver
- With out additional ICs 89C51 can be interfaced to drive five 7 Segment LED displays.
- 24 bit data is serially transmitted to the the driver by the 89C51 to display five digits with decimal point option
- MC14489s can be cascaded for more number of displays
- The brightness is controlled by the external resister 3.6K

Architecture of 80386

- The Internal Architecture of 80386 is divided into 3 sections.
- Central processing unit
- Memory management unit
- Bus interface unit
- Central processing unit is further divided into Execution unit and Instruction unit
- Execution unit has 8 General purpose and 8 Special purpose registers which are either used for handling data or calculating offset addresses.

Next page



80386 ARCHITECTURE

- The **Instruction unit** decodes the opcode bytes received from the 16-byte instruction code queue and arranges them in a 3- instruction decoded instruction queue.
- After decoding them pass it to the control section for deriving the necessary control signals. The barrel shifter increases the speed of all shift and rotate operations.
- The multiply / divide logic implements the bit-shift-rotate algorithms to complete the operations in minimum time.
- Even 32- bit multiplications can be executed within one microsecond by the multiply / divide logic.

Next page

- The Memory management unit consists of a Segmentation unit and a Paging unit.
- Segmentation unit allows the use of two address components, viz. segment and offset for relocability and sharing of code and data.
- Segmentation unit allows segments of size 4Gbytes at max.
- The Paging unit organizes the physical memory in terms of pages of 4kbytes size each.
- Paging unit works under the control of the segmentation unit, i.e. each segment is further divided into pages. The virtual memory is also organizes in terms of segments and pages by the memory management unit.

Next page

- The Segmentation unit provides a 4 level protection mechanism for protecting and isolating the system code and data from those of the application program.
- Paging unit converts linear addresses into physical addresses.
- The control and attribute PLA checks the privileges at the page level. Each of the pages maintains the paging information of the task. The limit and attribute PLA checks segment limits and attributes at segment level to avoid invalid accesses to code and data in the memory segments.

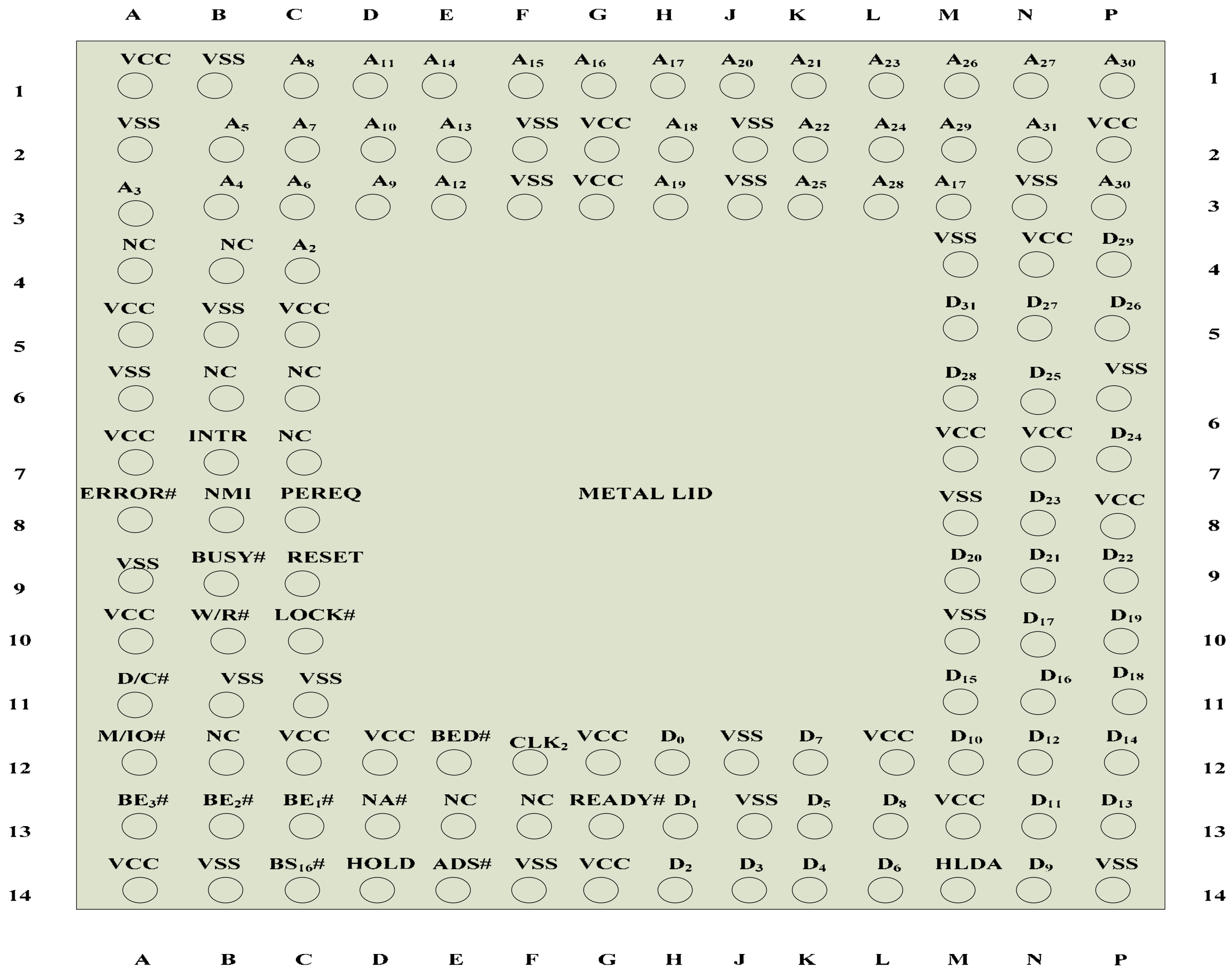
Next page

- The Bus control unit has a prioritizer to resolve the priority of the various bus requests. This controls the access of the bus. The address driver drives the bus enable and address signal A_0 – $A_{31.}$ The pipeline and dynamic bus sizing unit handle the related control signals.
- The data buffers interface the internal data bus with the system bus.

Signal Descriptions of 80386

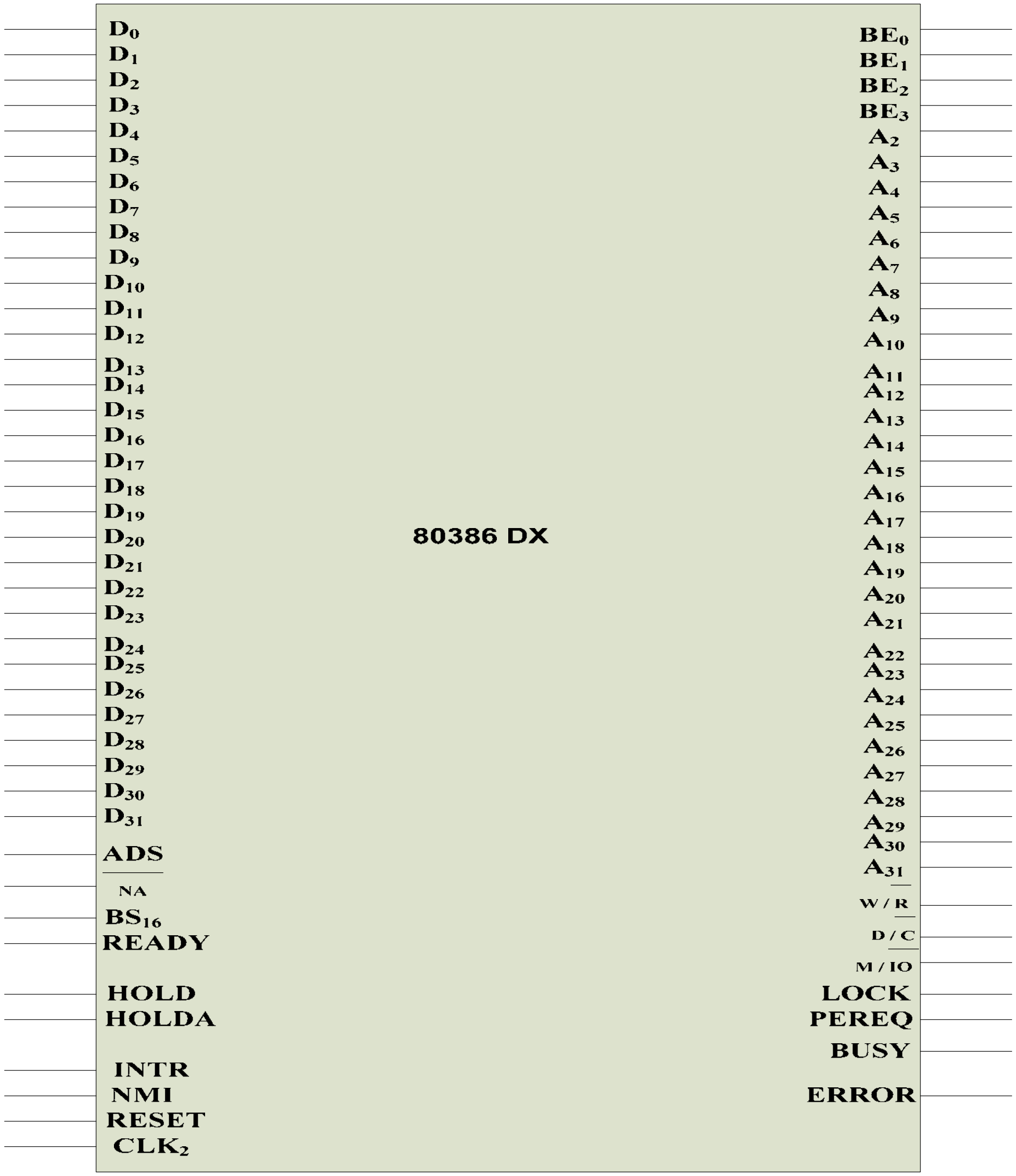
- CLK_2 :The input pin provides the basic system clock timing for the operation of 80386.
- D_0 – D_{31}:These 32 lines act as bidirectional data bus during different access cycles.
- A_{31} – A_2: These are upper 30 bit of the 32- bit address bus.
- $\overline{BE}_0$ to $\overline{BE}_3$: The 32- bit data bus supported by 80386 and the memory system of 80386 can be viewed as a 4- byte wide memory access mechanism. The 4 byte enable lines $\overline{BE}_0$ to $\overline{BE}_3$, may be used for enabling these 4 blanks. Using these 4 enable signal lines, the CPU may transfer 1 byte / 2 / 3 / 4 byte of data simultaneously.

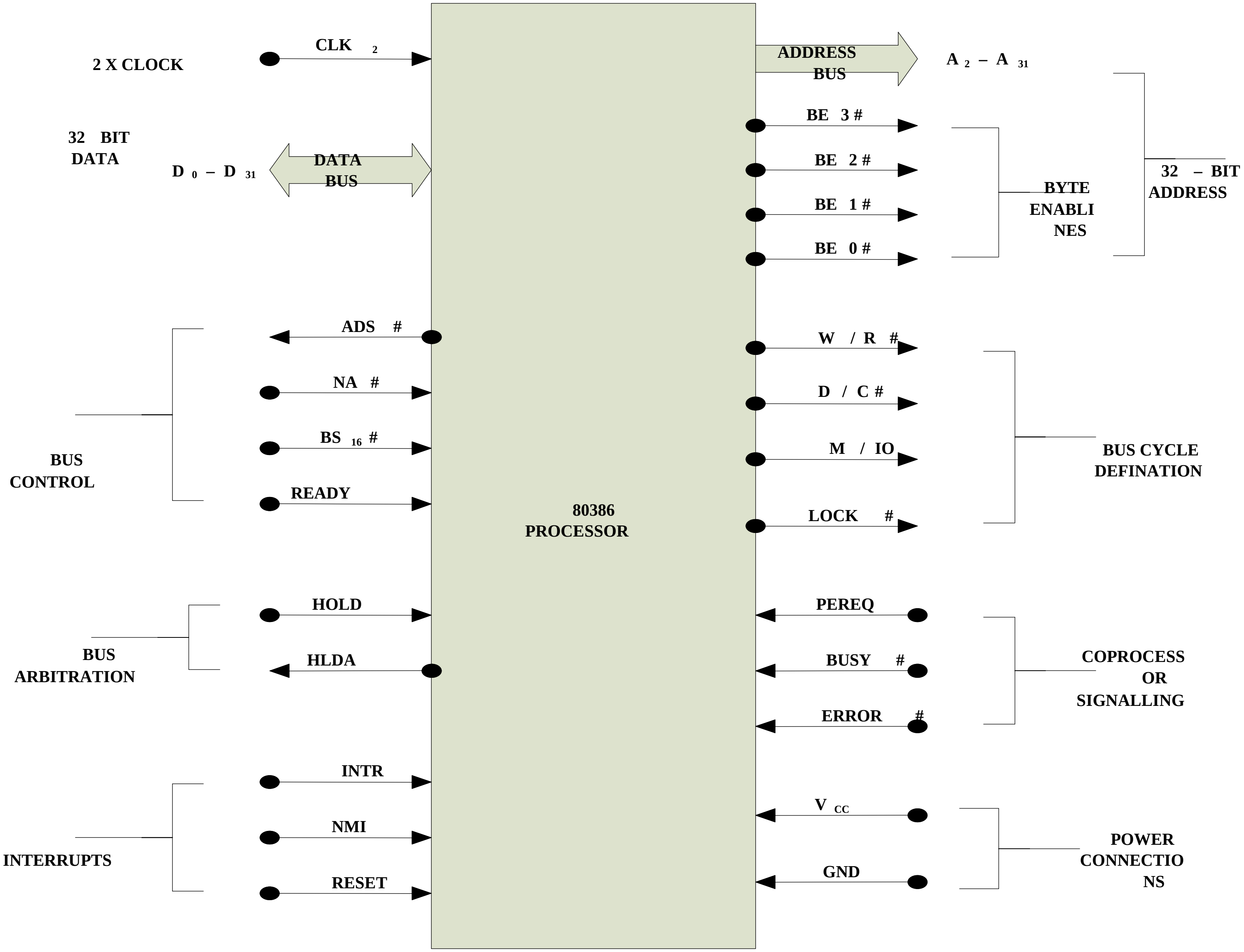
Next page



	A	B	C	D	E	F	G	H	J	K	L	M	N	P
1	VCC	VSS	A_8	A_{11}	A_{14}	A_{15}	A_{16}	A_{17}	A_{20}	A_{21}	A_{23}	A_{26}	A_{27}	A_{30}
2	VSS	A_5	A_7	A_{10}	A_{13}	VSS	VCC	A_{18}	VSS	A_{22}	A_{24}	A_{29}	A_{31}	VCC
3	A_3	A_4	A_6	A_9	A_{12}	VSS	VCC	A_{19}	VSS	A_{25}	A_{28}	A_{17}	VSS	A_{30}
4	NC	NC	A_2									VSS	VCC	D_{29}
5	VCC	VSS	VCC									D_{31}	D_{27}	D_{26}
6	VSS	NC	NC									D_{28}	D_{25}	VSS
7	VCC	INTR	NC									VCC	VCC	D_{24}
8	ERROR#	NMI	PEREQ									VSS	D_{23}	VCC
9	VSS	BUSY#	RESET									D_{20}	D_{21}	D_{22}
10	VCC	W/R#	LOCK#									VSS	D_{17}	D_{19}
11	D/C#	VSS	VSS									D_{15}	D_{16}	D_{18}
12	M/IO#	NC	VCC	VCC	BED#	CLK_2	VCC	D_0	VSS	D_7	VCC	D_{10}	D_{12}	D_{14}
13	BE_3#	BE_2#	BE_1#	NA#	NC	NC	READY#	D_1	VSS	D_5	D_8	VCC	D_{11}	D_{13}
14	VCC	VSS	BS_{16}#	HOLD	ADS#	VSS	VCC	D_2	D_3	D_4	D_6	HLDA	D_9	VSS

PIN DIAGRAM OF 80386


D_0
D_1
D_2
D_3
D_4
D_5
D_6
D_7
D_8
D_9
D_{10}
D_{11}
D_{12}
D_{13}
D_{14}
D_{15}
D_{16}
D_{17}
D_{18}
D_{19}
D_{20}
D_{21}
D_{22}
D_{23}
D_{24}
D_{25}
D_{26}
D_{27}
D_{28}
D_{29}
D_{30}
D_{31}
ADS
$\overline{NA}$
BS_{16}
READY
HOLD
HOLDA
INTR
NMI
RESET
CLK_2
80386 DX
BE_0
BE_1
BE_2
BE_3
A_2
A_3
A_4
A_5
A_6
A_7
A_8
A_9
A_{10}
A_{11}
A_{12}
A_{13}
A_{14}
A_{15}
A_{16}
A_{17}
A_{18}
A_{19}
A_{20}
A_{21}
A_{22}
A_{23}
A_{24}
A_{25}
A_{26}
A_{27}
A_{28}
A_{29}
A_{30}
A_{31}
$W/\overline{R}$
$D/\overline{C}$
$M/\overline{IO}$
LOCK
PEREQ
BUSY
ERROR


2 X CLOCK
CLK 2
32 BIT
DATA
D 0 – D 31
DATA
BUS
ADDRESS
BUS
A 2 – A 31
BE 3 #
BE 2 #
BE 1 #
BE 0 #
BYTE
ENABLI
NES
32 – BIT
ADDRESS
ADS #
NA #
BS 16 #
READY
BUS
CONTROL
80386
PROCESSOR
W / R #
D / C #
M / IO
LOCK #
BUS CYCLE
DEFINATION
HOLD
HLDA
BUS
ARBITRATION
PEREQ
BUSY #
ERROR #
COPROCESS
OR
SIGNALLING
INTR
NMI
RESET
INTERRUPTS
V CC
GND
POWER
CONNECTIO
NS

- **W/R#:** The write / read output distinguishes the write and read cycles from one another.
- **D/C#:** This data / control output pin distinguishes between a data transfer cycle from a machine control cycle like interrupt acknowledge.
- **M/IO#:** This output pin differentiates between the memory and I/O cycles.
- **LOCK#:** The LOCK# output pin enables the CPU to prevent the other bus masters from gaining the control of the system bus.
- **NA#:** The next address input pin, if activated, allows address pipelining, during 80386 bus cycles.

Next page

- **ADS#:** The address status output pin indicates that the address bus and bus cycle definition pins(W/R#, D/C#, M/IO#, BE_0# to BE_3#) are carrying the respective valid signals. The 80383 does not have any ALE signals and so this signals may be used for latching the address to external latches.
- **READY#:** The ready signals indicates to the CPU that the previous bus cycle has been terminated and the bus is ready for the next cycle. The signal is used to insert WAIT states in a bus cycle and is useful for interfacing of slow devices with CPU.
- **VCC**: These are system power supply lines.
- **VSS**: These return lines for the power supply.

Next page

- **BS_{16}#:** The bus size – 16 input pin allows the interfacing of 16 bit devices with the 32 bit wide 80386 data bus. Successive 16 bit bus cycles may be executed to read a 32 bit data from a peripheral.
- **HOLD**: The bus hold input pin enables the other bus masters to gain control of the system bus if it is asserted.
- **HLDA**: The bus hold acknowledge output indicates that a valid bus hold request has been received and the bus has been relinquished by the CPU.
- **BUSY#:** The busy input signal indicates to the CPU that the coprocessor is busy with the allocated task.

Next page

- **ERROR#:** The error input pin indicates to the CPU that the coprocessor has encountered an error while executing its instruction.
- **PEREQ**: The processor extension request output signal indicates to the CPU to fetch a data word for the coprocessor.
- **INTR**: This interrupt pin is a maskable interrupt, that can be masked using the IF of the flag register.
- **NMI:** A valid request signal at the non-maskable interrupt request input pin internally generates a non- maskable interrupt of type2.

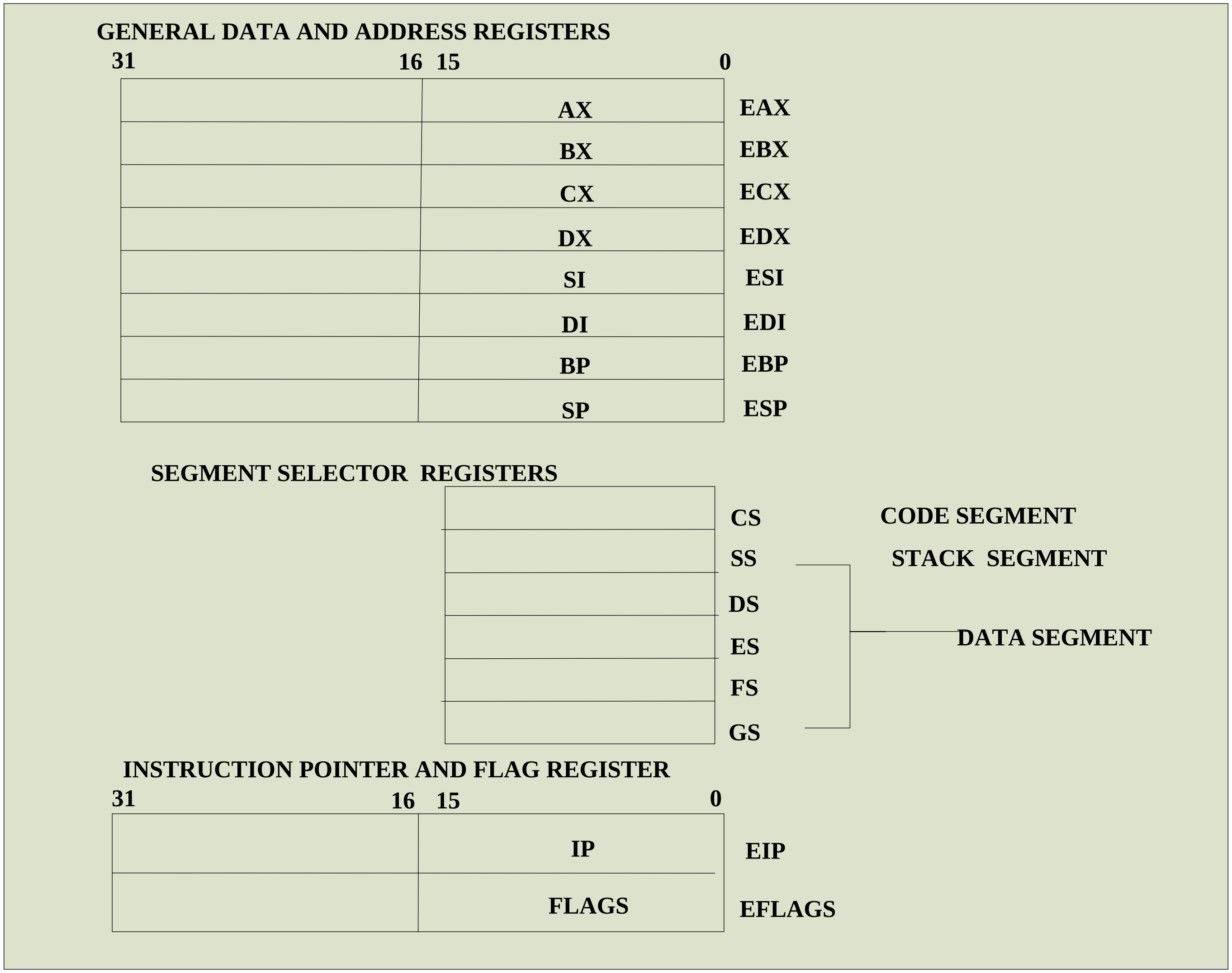
Next page

- **RESET**: A high at this input pin suspends the current operation and restart the execution from the starting location.
- **N / C** : No connection pins are expected to be left open while connecting the 80386 in the circuit.

Register Organisation

- The 80386 has eight 32 - bit general purpose registers which may be used as either 8 bit or 16 bit registers.
- A 32 - bit register known as an extended register, is represented by the register name with prefix E.
- Example : A 32 bit register corresponding to AX is EAX, similarly BX is EBX etc.
- The 16 bit registers BP, SP, SI and DI in 8086 are now available with their extended size of 32 bit and are names as EBP,ESP,ESI and EDI.
- AX represents the lower 16 bit of the 32 bit register EAX.
- BP, SP, SI, DI represents the lower 16 bit of their 32 bit counterparts, and can be used as independent 16 bit registers.

Next page



GENERAL DATA AND ADDRESS REGISTERS

31 — 16	15 — 0	
	AX	EAX
	BX	EBX
	CX	ECX
	DX	EDX
	SI	ESI
	DI	EDI
	BP	EBP
	SP	ESP

SEGMENT SELECTOR REGISTERS

CS — CODE SEGMENT

SS — STACK SEGMENT

DS

ES

FS

GS

DATA SEGMENT

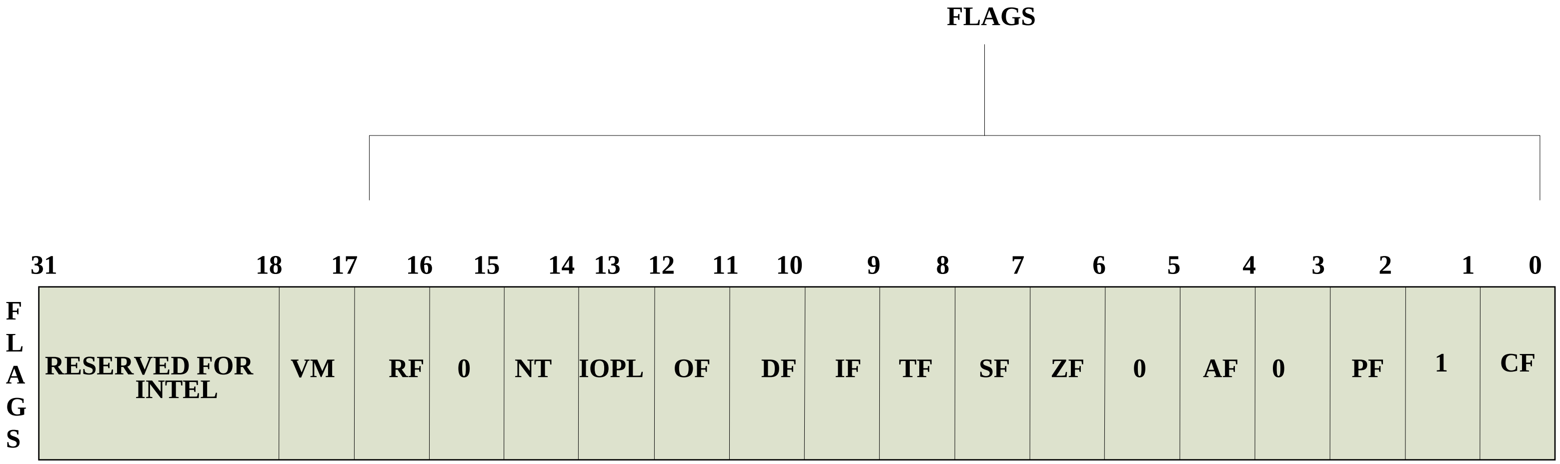
INSTRUCTION POINTER AND FLAG REGISTER

31 — 16	15 — 0	
	IP	EIP
	FLAGS	EFLAGS

Next page

- The six segment registers available in 80386 are CS, SS, DS, ES, FS and GS.
- The CS and SS are the code and the stack segment registers respectively, while DS, ES, FS, GS are 4 data segment registers.
- A 16 bit instruction pointer IP is available along with 32 bit counterpart EIP.
- ***Flag Register of 80386***: The Flag register of 80386 is a 32 bit register. Out of the 32 bits, Intel has reserved bits D_{18} to D_{31}, D_5 and D_3, while D_1 is always set at 1.Two extra new flags are added to the 80286 flag to derive the flag register of 80386. They are VM and RF flags.

Next page



	31 18	17	16	15	14	13 12	11	10	9	8	7	6	5	4	3	2	1	0
FLAGS	RESERVED FOR INTEL	VM	RF	0	NT	IOPL	OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

FLAG REGISTER OF 80386

- **VM - *Virtual Mode Flag***: If this flag is set, the 80386 enters the virtual 8086 mode within the protection mode. This is to be set only when the 80386 is in protected mode. In this mode, if any privileged instruction is executed an exception 13 is generated. This bit can be set using IRET instruction or any task switch operation only in the protected mode.
- ***RF- Resume Flag***: This flag is used with the debug register breakpoints. It is checked at the starting of every instruction cycle and if it is set, any debug fault is ignored during the instruction cycle. The RF is automatically reset after successful execution of every instruction, except for IRET and POPF instructions.

Next page

- Also, it is not automatically cleared after the successful execution of JMP, CALL and INT instruction causing a task switch. These instruction are used to set the RF to the value specified by the memory data available at the stack.
- ***Segment Descriptor Registers***: This registers are not available for programmers, rather they are internally used to store the descriptor information, like attributes, limit and base addresses of segments.
- The six segment registers have corresponding six 73 bit descriptor registers. Each of them contains 32 bit base address, 32 bit base limit and 9 bit attributes. These are automatically loaded when the corresponding segments are loaded with selectors.

Next page

- ***Control Registers***: The 80386 has three 32 bit control registers CR), CR_2 and CR_3 to hold global machine status independent of the executed task. Load and store instructions are available to access these registers.
- ***System Address Registers***: Four special registers are defined to refer to the descriptor tables supported by 80386.
- The 80386 supports four types of descriptor table, viz. global descriptor table (GDT), interrupt descriptor table (IDT), local descriptor table (LDT) and task state segment descriptor (TSS).

Next page

- ***Debug and Test Registers***: Intel has provide a set of 8 debug registers for hardware debugging. Out of these eight registers DR_0 to DR_7, two registers DR_4 and DR_5 are Intel reserved.
- The initial four registers DR_0 to DR_3 store four program controllable breakpoint addresses, while DR_6 and DR_7 respectively hold breakpoint status and breakpoint control information.
- Two more test register are provided by 80386 for page cacheing namely test control and test status register.

Next page

- ***ADDRESSING MODES***: The 80386 supports overall eleven addressing modes to facilitate efficient execution of higher level language programs.
- In case of all those modes, the 80386 can now have 32-bit immediate or 32- bit register operands or displacements.
- The 80386 has a family of scaled modes. In case of scaled modes, any of the index register values can be multiplied by a valid scale factor to obtain the displacement.
- The valid scale factor are 1, 2, 4 and 8.

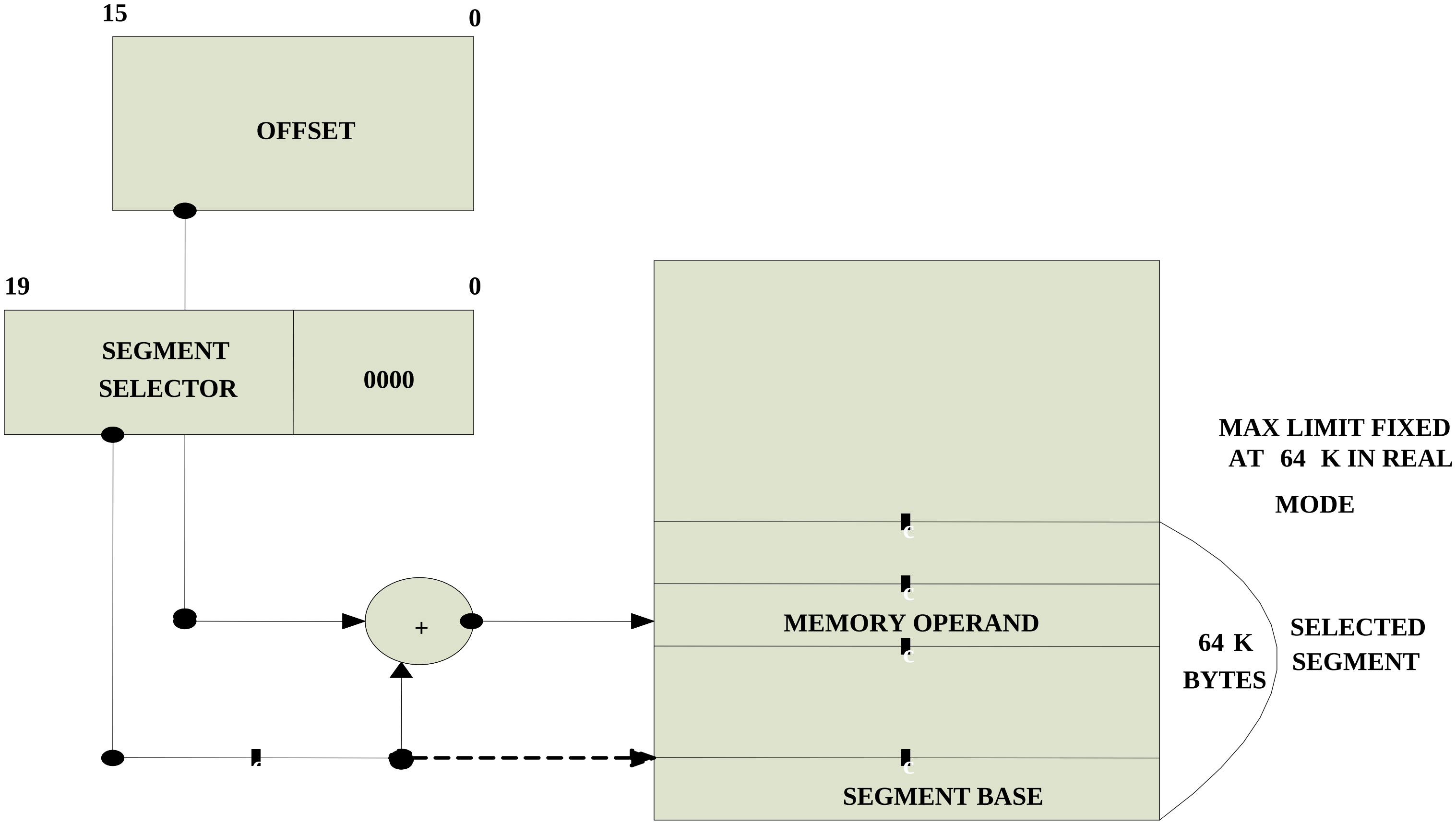
Next page

- The different scaled modes are as follows.
- ***Scaled Indexed Mode***: Contents of the an index register are multiplied by a scale factor that may be added further to get the operand offset.
- ***Based Scaled Indexed Mode***: Contents of the an index register are multiplied by a scale factor and then added to base register to obtain the offset.
- ***Based Scaled Indexed Mode with Displacement***: The Contents of the an index register are multiplied by a scaling factor and the result is added to a base register and a displacement to get the offset of an operand.

Real Address Mode of 80386

- After reset, the 80386 starts from memory location FFFFFFF0H under the real address mode. In the real mode, 80386 works as a fast 8086 with 32-bit registers and data types.
- In real mode, the default operand size is 16 bit but 32- bit operands and addressing modes may be used with the help of override prefixes.
- The segment size in real mode is 64k, hence the 32-bit effective addressing must be less than 0000FFFFFH. The real mode initializes the 80386 and prepares it for protected mode.

Next page



Physical Address Formation In Real Mode Of 80386

Next page

- ***Memory Addressing in Real Mode***: In the real mode, the 80386 can address at the most 1Mbytes of physical memory using address lines A_0-A_{19}.
- Paging unit is disabled in real addressing mode, and hence the real addresses are the same as the physical addresses.
- To form a physical memory address, appropriate segment registers contents (16-bits) are shifted left by four positions and then added to the 16-bit offset address formed using one of the addressing modes, in the same way as in the 80386 real address mode.
- The segment in 80386 real mode can be read, write or executed, i.e. no protection is available.

Next page

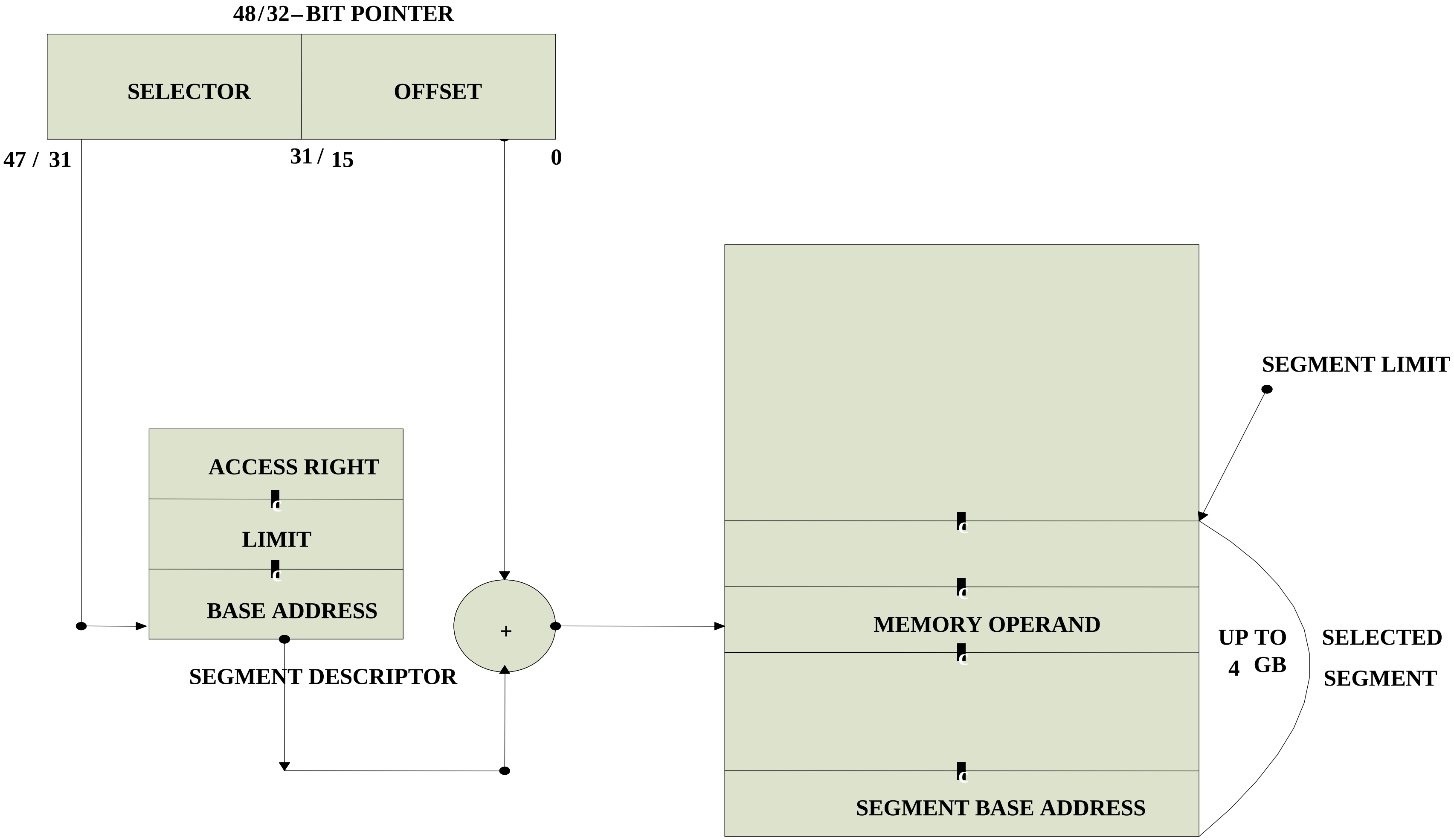
- Any fetch or access past the end of the segment limit generate exception 13 in real address mode.
- The segments in 80386 real mode may be overlapped or non-overlapped.
- The interrupt vector table of 80386 has been allocated 1Kbyte space starting from 00000H to 003FFH.

Next page

Protected Mode of 80386

- All the capabilities of 80386 are available for utilization in its protected mode of operation.
- The 80386 in protected mode support all the software written for 80286 and 8086 to be executed under the control of memory management and protection abilities of 80386.
- The protected mode allows the use of additional instruction, addressing modes and capabilities of 80386.
- ***ADDRESSING IN PROTECTED MODE***: In this mode, the contents of segment registers are used as selectors to address descriptors which contain the segment limit, base address and access rights byte of the segment.

Next page



Protected Mode Addressing Without Paging Unit

- The effective address (offset) is added with segment base address to calculate linear address. This linear address is further used as physical address, if the paging unit is disabled, otherwise the paging unit converts the linear address into physical address.
- The paging unit is a memory management unit enabled only in protected mode. The paging mechanism allows handling of large segments of memory in terms of pages of 4Kbyte size.
- The paging unit operates under the control of segmentation unit. The paging unit if enabled converts linear addresses into physical address, in protected mode.

Segmentation

- ***DESCRIPTOR TABLES***: These descriptor tables and registers are manipulated by the operating system to ensure the correct operation of the processor, and hence the correct execution of the program.
- Three types of the 80386 descriptor tables are listed as follows:
- GLOBAL DESCRIPTOR TABLE (GDT)
- LOCAL DESCRIPTOR TABLE (LDT)
- INTERRUPT DESCRIPTOR TABLE (IDT)

Next page

- ***DESCRIPTORS***: The 80386 descriptors have a 20-bit segment limit and 32-bit segment address. The descriptor of 80386 are 8-byte quantities access right or attribute bits along with the base and limit of the segments.
- ***Descriptor Attribute Bits***: The A (accessed) attributed bit indicates whether the segment has been accessed by the CPU or not.
- The TYPE field decides the descriptor type and hence the segment type.
- The S bit decides whether it is a system descriptor (S=0) or code/data segment descriptor (S=1).

Next page

<table>
<tr><td colspan="6">31</td><td colspan="6">0</td><td></td></tr>
<tr><td colspan="6">SEGMENT BASE 15..0</td><td colspan="6">SEGMENT BASE 15...0</td><td>BYTE ADDRESS 0</td></tr>
<tr><td>BASE 31..24</td><td>G</td><td>D</td><td>0</td><td>AVL</td><td>LIMIT
19...16</td><td>P</td><td>DPL</td><td>S</td><td>TYPE</td><td>A</td><td>BASE
23..26</td><td>+ 4</td></tr>
</table>

Structure of An Descriptor

BASE Base Address of the segment

LIMIT The length of the segment

P Present Bit - 1=Present ,0 = not present

S Segment Descriptor -0 = System Descriptor,
1 = Code or data segment descriptor

TYPE Type of segment

G Granularity Bit - 1= Segment length is page granular ,
0 = Segment length is byte granular

D Default Operation size

0 Bit must be zero

AVL Available field for user or OS

Next page

- The DPL field specifies the descriptor privilege level.
- The D bit specifies the code segment operation size. If D=1, the segment is a 32-bit operand segment, else, it is a 16-bit operand segment.
- The P bit (present) signifies whether the segment is present in the physical memory or not. If P=1, the segment is present in the physical memory.
- The G (granularity) bit indicates whether the segment is page addressable. The zero bit must remain zero for compatibility with future process.

Next page

- The AVL (available) field specifies whether the descriptor is for user or for operating system.
- The 80386 has five types of descriptors listed as follows:

1. Code or Data Segment Descriptors.
2. System Descriptors.
3. Local descriptors.
4. TSS (Task State Segment) Descriptors.
5. GATE Descriptors.

- The 80386 provides a four level protection mechanism exactly in the same way as the 80286 does.

Paging

- ***PAGING OPERATION***: Paging is one of the memory management techniques used for virtual memory multitasking operating system.
- The segmentation scheme may divide the physical memory into a variable size segments but the paging divides the memory into a fixed size pages.
- The segments are supposed to be the logical segments of the program, but the pages do not have any logical relation with the program.
- The pages are just fixed size portions of the program module or data.

Next page

- The advantage of paging scheme is that the complete segment of a task need not be in the physical memory at any time.
- Only a few pages of the segments, which are required currently for the execution need to be available in the physical memory. Thus the memory requirement of the task is substantially reduced, relinquishing the available memory for other tasks.
- Whenever the other pages of task are required for execution, they may be fetched from the secondary storage.
- The previous page which are executed, need not be available in the memory, and hence the space occupied by them may be relinquished for other tasks.

Next page

- Thus paging mechanism provides an effective technique to manage the physical memory for multitasking systems.
- ***Paging Unit***: The paging unit of 80386 uses a two level table mechanism to convert a linear address provided by segmentation unit into physical addresses.
- The paging unit converts the complete map of a task into pages, each of size 4K. The task is further handled in terms of its page, rather than segments.
- The paging unit handles every task in terms of three components namely page directory, page tables and page itself.

Next page

- ***Paging Descriptor Base Register***: The control register CR_2 is used to store the 32-bit linear address at which the previous page fault was detected.
- The CR_3 is used as page directory physical base address register, to store the physical starting address of the page directory.
- The lower 12 bit of the CR_3 are always zero to ensure the page size aligned directory. A move operation to CR_3 automatically loads the page table entry caches and a task switch operation, to load CR_0 suitably.

Next page

- ***Page Directory*** : This is at the most 4Kbytes in size. Each directory entry is of 4 bytes, thus a total of 1024 entries are allowed in a directory.
- The upper 10 bits of the linear address are used as an index to the corresponding page directory entry. The page directory entries point to page tables.
- ***Page Tables***: Each page table is of 4Kbytes in size and many contain a maximum of 1024 entries. The page table entries contain the starting address of the page and the statistical information about the page.

Next page

PAGE TABLE ADDRESS **31…12**	**OS** **RESERVED**	**0**	**0**	**D**	**A**	**0**	**0**	**U** **-** **S**	**R** **-** **W**	**P**

PAGE DIRECTORY ENTRY

PAGE FRAME ADDRESS **31…12**	**OS** **RESERVED**	**0**	**0**	**D**	**A**	**0**	**0**	**U** **-** **S**	**R** **-** **W**	**P**

PAGE TABLE ENTRY

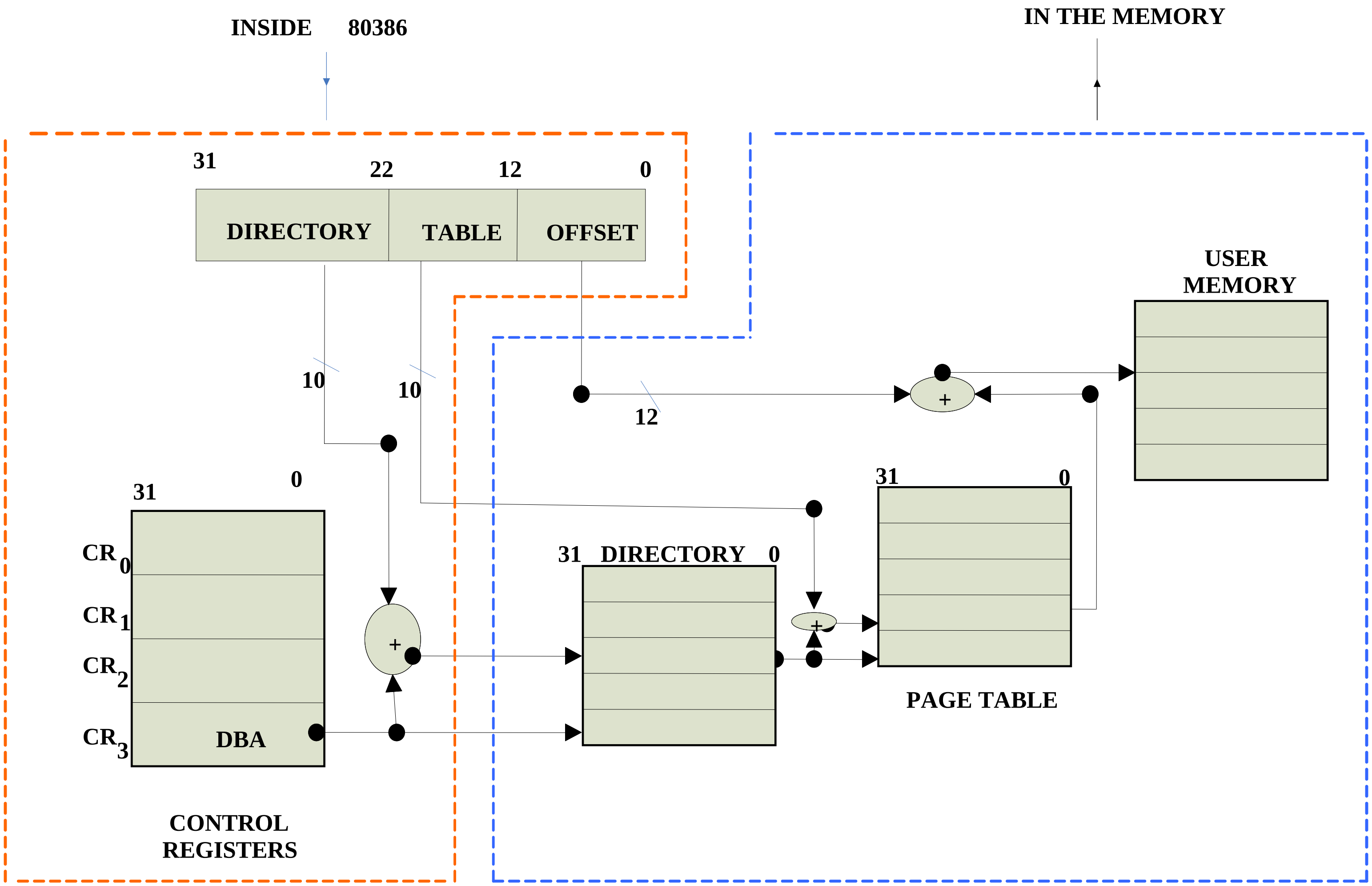
Next page

U - S	R - W	PERMITTED FOR LEVEL3	PERMITTED FOR LEVEL2 ,1 OR 0
0	0	NONE	READ / WRITE
0	1	NONE	READ / WRITE
1	0	READ ONLY	READ / WRITE
1	1	READ - WRITE	READ / WRITE

Next page

- The upper 20 bit page frame address is combined with the lower 12 bit of the linear address. The address bits A_{12}- A_{21} are used to select the 1024 page table entries. The page table can be shared between the tasks.
- The P bit of the above entries indicate, if the entry can be used in address translation.
- If P=1, the entry can be used in address translation, otherwise it cannot be used.
- The P bit of the currently executed page is always high.
- The accessed bit A is set by 80386 before any access to the page. If A=1, the page is accessed, else unaccessed.

Next page



DBA Physical directory base address

Next page

- The D bit (Dirty bit) is set before a write operation to the page is carried out. The D-bit is undefined for page director entries.
- The OS reserved bits are defined by the operating system software.
- The User / Supervisor (U/S) bit and read/write bit are used to provide protection. These bits are decoded to provide protection under the 4 level protection model.
- The level 0 is supposed to have the highest privilege, while the level 3 is supposed to have the least privilege.
- This protection provide by the paging unit is transparent to the segmentation unit.

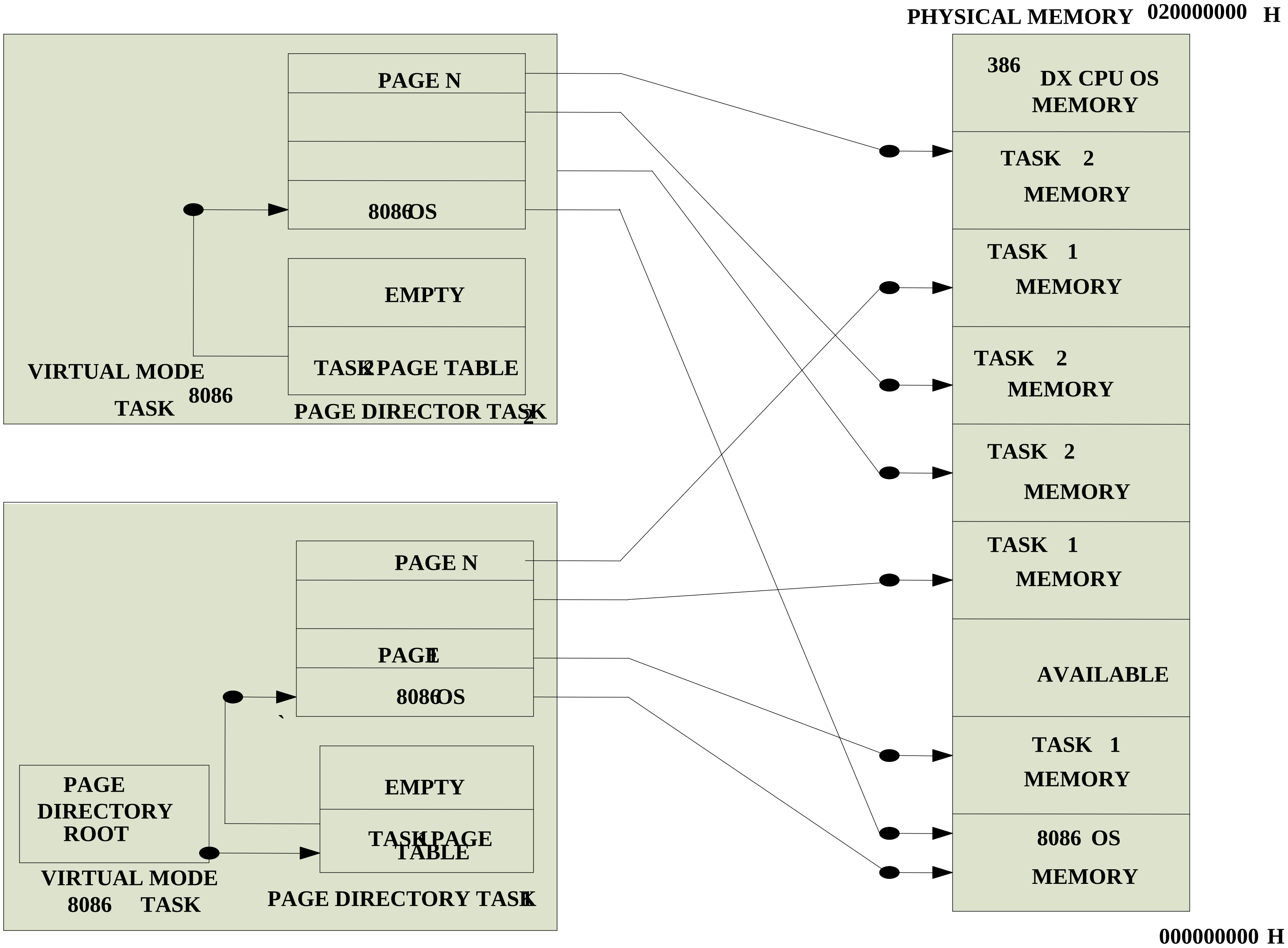
Virtual 8086 Mode

- In its protected mode of operation, 80386DX provides a virtual 8086 operating environment to execute the 8086 programs.
- The real mode can also used to execute the 8086 programs along with the capabilities of 80386, like protection and a few additional instructions.
- Once the 80386 enters the protected mode from the real mode, it cannot return back to the real mode without a reset operation.
- Thus, the virtual 8086 mode of operation of 80386, offers an advantage of executing 8086 programs while in protected mode.

Next page

- The address forming mechanism in virtual 8086 mode is exactly identical with that of 8086 real mode.
- In virtual mode, 8086 can address 1Mbytes of physical memory that may be anywhere in the 4Gbytes address space of the protected mode of 80386.
- Like 80386 real mode, the addresses in virtual 8086 mode lie within 1Mbytes of memory.
- In virtual mode, the paging mechanism and protection capabilities are available at the service of the programmers.
- The 80386 supports multiprogramming, hence more than one programmer may be use the CPU at a time.

Next page



PHYSICAL MEMORY 020000000 H

386 DX CPU OS MEMORY

TASK 2 MEMORY

TASK 1 MEMORY

TASK 2 MEMORY

TASK 2 MEMORY

TASK 1 MEMORY

AVAILABLE

TASK 1 MEMORY

8086 OS MEMORY

000000000 H

PAGE N

8086 OS

EMPTY

TASK 2 PAGE TABLE

VIRTUAL MODE 8086 TASK

PAGE DIRECTOR TASK 2

PAGE N

PAGE 1

8086 OS

EMPTY

TASK 1 PAGE TABLE

PAGE DIRECTORY ROOT

VIRTUAL MODE 8086 TASK

PAGE DIRECTORY TASK 1

Memory Management In Virtual 8086 Mode

Next page

- Paging unit may not be necessarily enable in virtual mode, but may be needed to run the 8086 programs which require more than 1Mbyts of memory for memory management function.
- In virtual mode, the paging unit allows only 256 pages, each of 4Kbytes size.
- Each of the pages may be located anywhere in the maximum 4Gbytes physical memory. The virtual mode allows the multiprogramming of 8086 applications.
- The virtual 8086 mode executes all the programs at privilege level 3.Any of the other programmes may deny access to the virtual mode programs or data.

Next page

- However, the real mode programs are executed at the highest privilege level, i.e. level 0.
- The virtual mode may be entered using an IRET instruction at CPL=0 or a task switch at any CPL, executing any task whose TSS is having a flag image with VM flag set to 1.
- The IRET instruction may be used to set the VM flag and consequently enter the virtual mode.
- The PUSHF and POPF instructions are unable to read or set the VM bit, as they do not access it.
- Even in the virtual mode, all the interrupts and exceptions are handled by the protected mode interrupt handler.

Next page

- To return to the protected mode from the virtual mode, any interrupt or execution may be used.
- As a part of interrupt service routine, the VM bit may be reset to zero to pull back the 80386 into protected mode.

Features of 80386

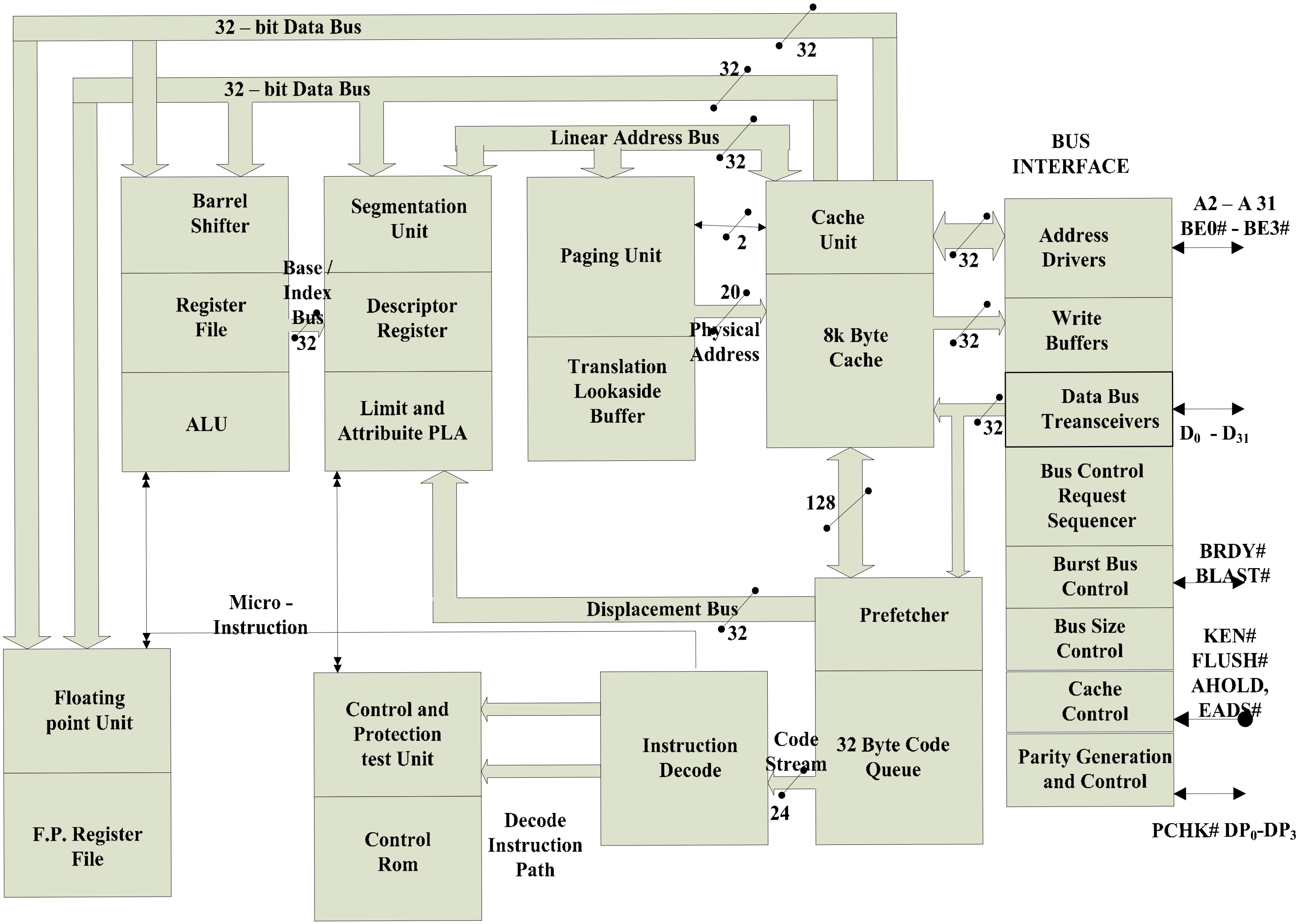
- This 80386 is a 32bit processor that supports, 8bit/32bit data operands.
- The 80386 instruction set is upward compatible with all its predecessors.
- The 80386 can run 8086 applications under protected mode in its virtual 8086 mode of operation.
- With the 32 bit address bus, the 80386 can address upto 4Gbytes of physical memory. The physical memory is organised in terms of segments of 4Gbytes at maximum.
- The 80386 CPU supports 16K number of segments and thus the total virtual space of 4Gbytes * 16K = 64 Terrabytes.

Next page

- The memory management section of 80386 supports the virtual memory, paging and four levels of protection, maintaining full compatibility with 80286.
- The 80386 offers a set of 8 debug registers DR_0-DR_7 for hardware debugging and control. The 80386 has on-chip address translation cache.
- The concept of paging is introduced in 80386 that enables it to organise the available physical memory in terms of pages of size 4Kbytes each, under the segmented memory.
- The 80386 can be supported by 80387 for mathematical data processing.

80486 Microprocessor

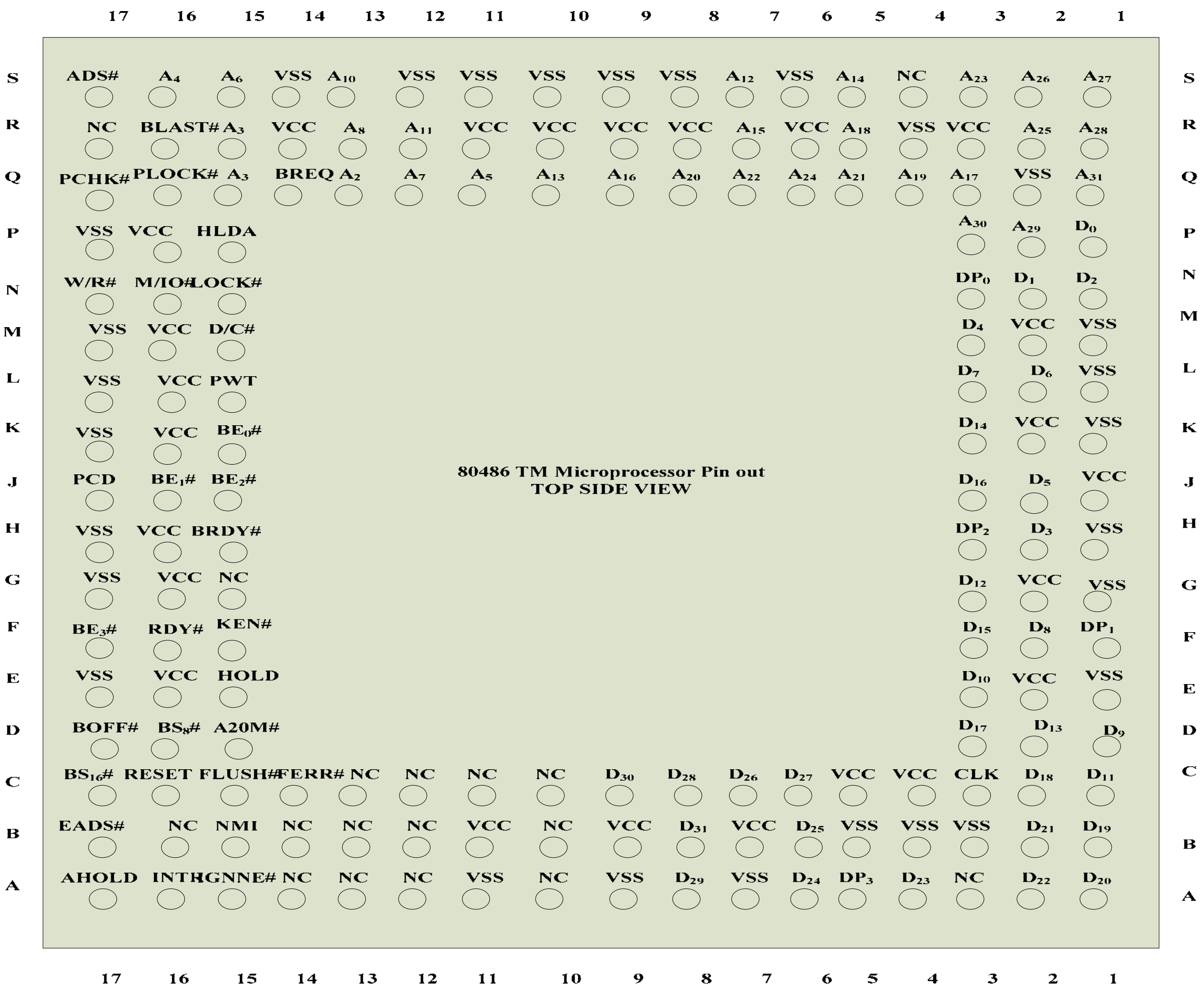
- The 32-bit 80486 is the next evolutionary step up from the 80386.
- One of the most obvious feature included in a 80486 is a built in math coprocessor. This coprocessor is essentially the same as the 80387 processor used with a 80386, but being integrated on the chip allows it to execute math instructions about three times as fast as a 80386/387 combination.
- 80486 is an 8Kbyte code and data cache.
- To make room for the additional signals, the 80486 is packaged in a 168 pin, pin grid array package instead of the 132 pin PGA used for the 80386.


32 – bit Data Bus
32
32
32 – bit Data Bus
Linear Address Bus
32
BUS INTERFACE
Barrel Shifter
Register File
ALU
Base / Index Bus
32
Segmentation Unit
Descriptor Register
Limit and Attribuite PLA
Paging Unit
Translation Lookaside Buffer
2
20
Physical Address
Cache Unit
8k Byte Cache
32
32
32
Address Drivers
Write Buffers
Data Bus Treanseivers
Bus Control Request Sequencer
Burst Bus Control
Bus Size Control
Cache Control
Parity Generation and Control
A2 – A 31
BE0# - BE3#
D$_0$ - D$_{31}$
BRDY#
BLAST#
KEN#
FLUSH#
AHOLD,
EADS#
PCHK# DP$_0$-DP$_3$
128
Micro - Instruction
Displacement Bus
32
Prefetcher
32 Byte Code Queue
Code Stream
24
Floating point Unit
F.P. Register File
Control and Protection test Unit
Control Rom
Decode Instruction Path
Instruction Decode

Pin Definitions

- **A_{31}-A_2** : Address outputs A31-A2 provide the memory and I/O with the address during normal operation. During a cache line invalidation A31-A4 are used to drive the microprocessor.
- **$\overline{A_{20}M_3}$** : The address bit 20 mask causes the 80486 to wrap its address around from location 000FFFFFH to 00000000H as in 8086. This provides a memory system that functions like the 1M byte real memory system in the 8086 processors.
- **$\overline{ADS}$** : The address data strobe become logic zero to indicate that the address bus contains a valid memory address.

Next page



80486 TM Microprocessor Pin out
TOP SIDE VIEW

	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	
S	ADS#	A_4	A_6	VSS	A_{10}	VSS	VSS	VSS	VSS	VSS	A_{12}	VSS	A_{14}	NC	A_{23}	A_{26}	A_{27}	S
R	NC	BLAST#	A_3	VCC	A_8	A_{11}	VCC	VCC	VCC	VCC	A_{15}	VCC	A_{18}	VSS	VCC	A_{25}	A_{28}	R
Q	PCHK#	PLOCK#	A_3	BREQ	A_2	A_7	A_5	A_{13}	A_{16}	A_{20}	A_{22}	A_{24}	A_{21}	A_{19}	A_{17}	VSS	A_{31}	Q
P	VSS	VCC	HLDA												A_{30}	A_{29}	D_0	P
N	W/R#	M/IO#	LOCK#												DP_0	D_1	D_2	N
M	VSS	VCC	D/C#												D_4	VCC	VSS	M
L	VSS	VCC	PWT												D_7	D_6	VSS	L
K	VSS	VCC	BE_0#												D_{14}	VCC	VSS	K
J	PCD	BE_1#	BE_2#												D_{16}	D_5	VCC	J
H	VSS	VCC	BRDY#												DP_2	D_3	VSS	H
G	VSS	VCC	NC												D_{12}	VCC	VSS	G
F	BE_3#	RDY#	KEN#												D_{15}	D_8	DP_1	F
E	VSS	VCC	HOLD												D_{10}	VCC	VSS	E
D	BOFF#	BS_8#	A20M#												D_{17}	D_{13}	D_9	D
C	BS_{16}#	RESET	FLUSH#	FERR#	NC	NC	NC	NC	D_{30}	D_{28}	D_{26}	D_{27}	VCC	VCC	CLK	D_{18}	D_{11}	C
B	EADS#	NC	NMI	NC	NC	NC	VCC	NC	VCC	D_{31}	VCC	D_{25}	VSS	VSS	VSS	D_{21}	D_{19}	B
A	AHOLD	INTR	IGNNE#	NC	NC	NC	VSS	NC	VSS	D_{29}	VSS	D_{24}	DP_3	D_{23}	NC	D_{22}	D_{20}	A
	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	

- **AHOLD**: The address hold input causes the microprocessor to place its address bus connections at their high-impedance state, with the remainder of the buses staying active. It is often used by another bus master to gain access for a cache invalidation cycle. BREQ: This bus request output indicates that the 486 has generated an internal bus request.
- $\overline{\mathbf{BE_3}}$**-**$\overline{\mathbf{BE_0}}$: Byte enable outputs select a bank of the memory system when information is transferred between the microprocessor and its memory and I/O.

  The BE_3 signal enables D_{31} – D_{24} , BE_2 enables D_{23}-D_{16}, BE_1 enables D_{15} – D_8 and BE_0 enables D_7-D_0.

Next page

- $\overline{\textbf{BLAST}}$: The burst last output shows that the burst bus cycle is complete on the next activation of BRDY# signal.

- $\overline{\textbf{BOFF}}$: The Back-off input causes the microprocessor to place its buses at their high impedance state during the next cycle. The microprocessor remains in the bus hold state until the BOFF# pin is placed at a logic 1 level.

- **NMI** : The non-maskable interrupt input requests a type 2 interrupt.

Next page

- $\overline{\text{BRDY}}$: The burst ready input is used to signal the microprocessor that a burst cycle is complete.

- $\overline{\text{KEN}}$: The cache enable input causes the current bus to be stored in the internal.

- $\overline{\text{LOCK}}$: The lock output becomes a logic 0 for any instruction that is prefixed with the lock prefix.

- W / $\overline{\text{R}}$: current bus cycle is either a read or a write.

Next page

- **IGNNE** : The ignore numeric error input causes the coprocessor to ignore floating point error and to continue processing data. The signal does not affect the state of the FERR pin.

- **FLUSH** : The cache flush input forces the microprocessor to erase the contents of its 8K byte internal cache.

- **EADS**: The external address strobe input is used with AHOLD to signal that an external address is used to perform a cache invalidation cycle.

Next page

- $\overline{\mathbf{FERR}}$: The floating point error output indicates that the floating point coprocessor has detected an error condition. It is used to maintain compatibility with DOS software.

- $\overline{\mathbf{BS_8}}$: The bus size 8, input causes the 80486 to structure itself with an 8-bit data bus to access byte-wide memory and I/O components.

- $\overline{\mathbf{BS_{16}}}$: The bus size 16, input causes the 80486 to structure itself with an 16-bit data bus to access word-wide memory and I/O components.

- $\overline{\textbf{PCHK}}$: The parity check output indicates that a parity error was detected during a read operation on the DP_3 – DP_0 pin.

- $\overline{\textbf{PLOCK}}$: The pseudo-lock output indicates that current operation requires more than one bus cycle to perform. This signal becomes a logic 0 for arithmetic coprocessor operations that access 64 or 80 bit memory data.
- **PWT**: The page write through output indicates the state of the PWT attribute bit in the page table entry or the page directory entry.

Next page

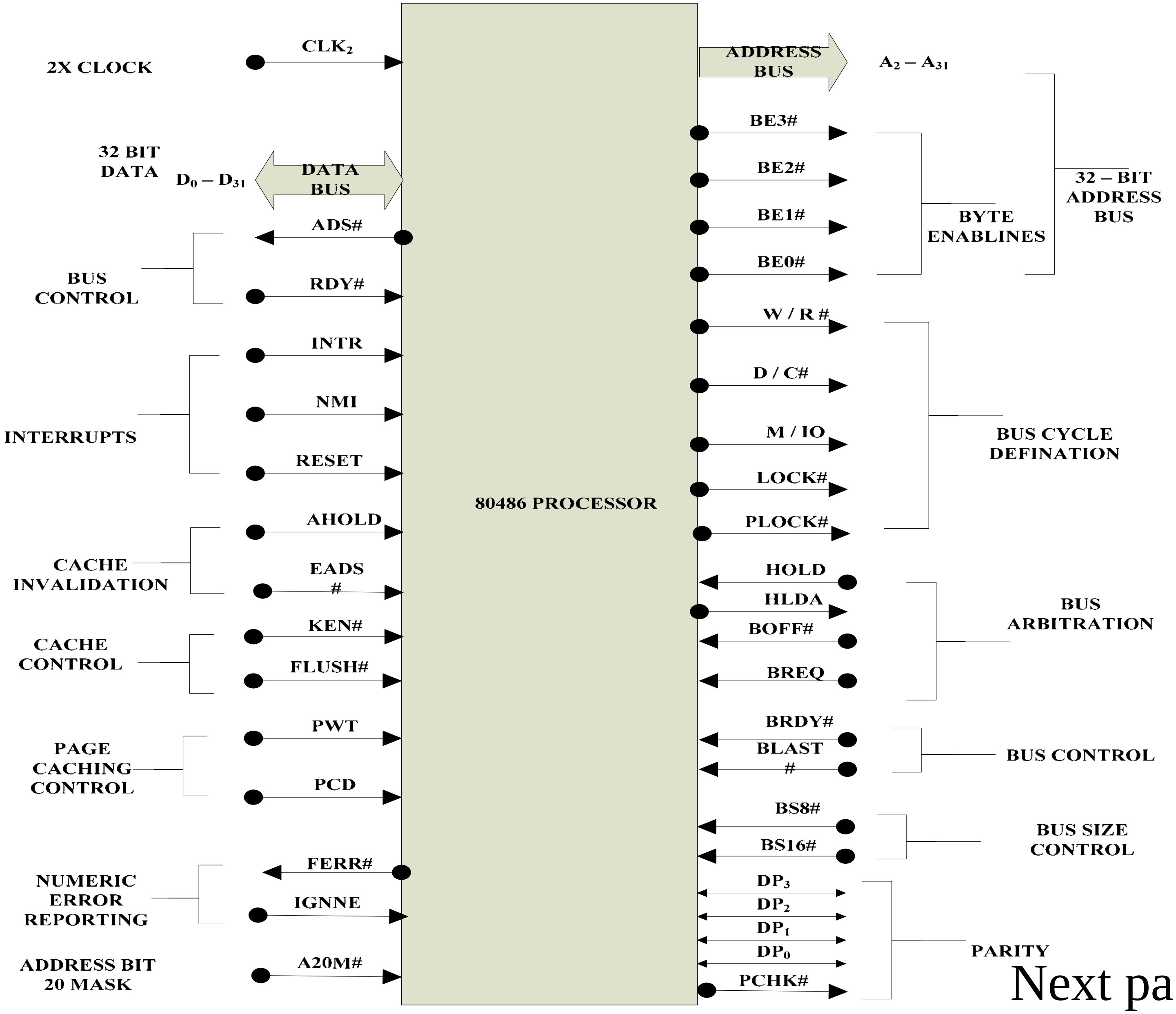
- **$\overline{\text{RDY}}$** : The ready input indicates that a non-burst bus cycle is complete. The RDY signal must be returned or the microprocessor places wait states into its timing until RDY is asserted.

- **M / $\overline{\text{IO}}$** : Memory / $\overline{\text{IO}}$ defines whether the address bus contains a memory address or an I/O port number. It is also combined with the W/ R signal to generate memory and I/O read and write control signals.

80486 Signal Group

- The 80486 data bus, address bus, byte enable, ADS#, RDY#, INTR, RESET, NMI, M/IO#, D/C#, W/R#, LOCK#, HOLD, HLDA and BS_{16}# signals function as we described for 80386.
- The 80486 requires 1 clock instead of 2 clock required by 80386.
- A new signal group on the 486 is the PARITY group DP_0-DP_3 and PCHK#.
- These signals allow the 80486 to implement parity detection / generation for memory reads and memory writes.
- During a memory write operation, the 80486 generates an even parity bit for each byte and outputs these bits on the DP_0-DP_3 lines.

Next page



80486 PROCESSOR

2X CLOCK — CLK$_2$

32 BIT DATA — $D_0 - D_{31}$ — DATA BUS

BUS CONTROL — ADS#, RDY#

INTERRUPTS — INTR, NMI, RESET

CACHE INVALIDATION — AHOLD, EADS #

CACHE CONTROL — KEN#, FLUSH#

PAGE CACHING CONTROL — PWT, PCD

NUMERIC ERROR REPORTING — FERR#, IGNNE

ADDRESS BIT 20 MASK — A20M#

ADDRESS BUS — $A_2 - A_{31}$

BYTE ENABLINES — BE3#, BE2#, BE1#, BE0#

32 – BIT ADDRESS BUS

BUS CYCLE DEFINATION — W / R #, D / C#, M / IO, LOCK#, PLOCK#

BUS ARBITRATION — HOLD, HLDA, BOFF#, BREQ

BUS CONTROL — BRDY#, BLAST #

BUS SIZE CONTROL — BS8#, BS16#

PARITY — DP$_3$, DP$_2$, DP$_1$, DP$_0$, PCHK#

Next page

- These bits will store in a separate parity memory bank.
- During a read operation the stored parity bits will be read from the parity memory and applied to the DP_0-DP_3 pins.
- The 80486 checks the parities of the data bytes read and compares them with the DP_0-DP_3 signals. If a parity error is found, the 80486 asserts the PCHK# signal.
- Another new signals group consists of the BURST ready signal BRDY# and BURST last signal BLAST#.
- These signals are used to control burst-mode memory reads and writes.

Next page

- A normal 80486 memory read operation to read a line into the cache requires 2 clock cycles. However, if a series of reads is being done from successive memory locations, the reads can be done in burst mode with only 1 clock cycle per read.
- To start the process the 80486 sends out the first address and asserts the BLAST# signal high. When the external DRAM controller has the first data bus, it asserts the BRDY# signal.
- The 80486 reads the data word and outputs the next address. Since the data words are at successive addresses, only the lower address bits need to be changed. If the DRAM controller is operating in the page or the static column modes then it will only have to output a new column address to the DRAM.

Next page

- In this mode the DRAM will be able to output the new data word within 1 clock cycle.
- When the processor has read the required number of data words, it asserts the BLAST# signal low to terminate the burst mode.
- The final signal we want to discuss here are the bus request output signal BREQ, the back-off input signal BOFF#, the HOLD signal and the hold-acknowledge signal HLDA.
- These signals are used to control sharing the local 486 bus by multiple processors (bus master).
- When a master on the bus need to use the bus, it asserts its BERQ signal .

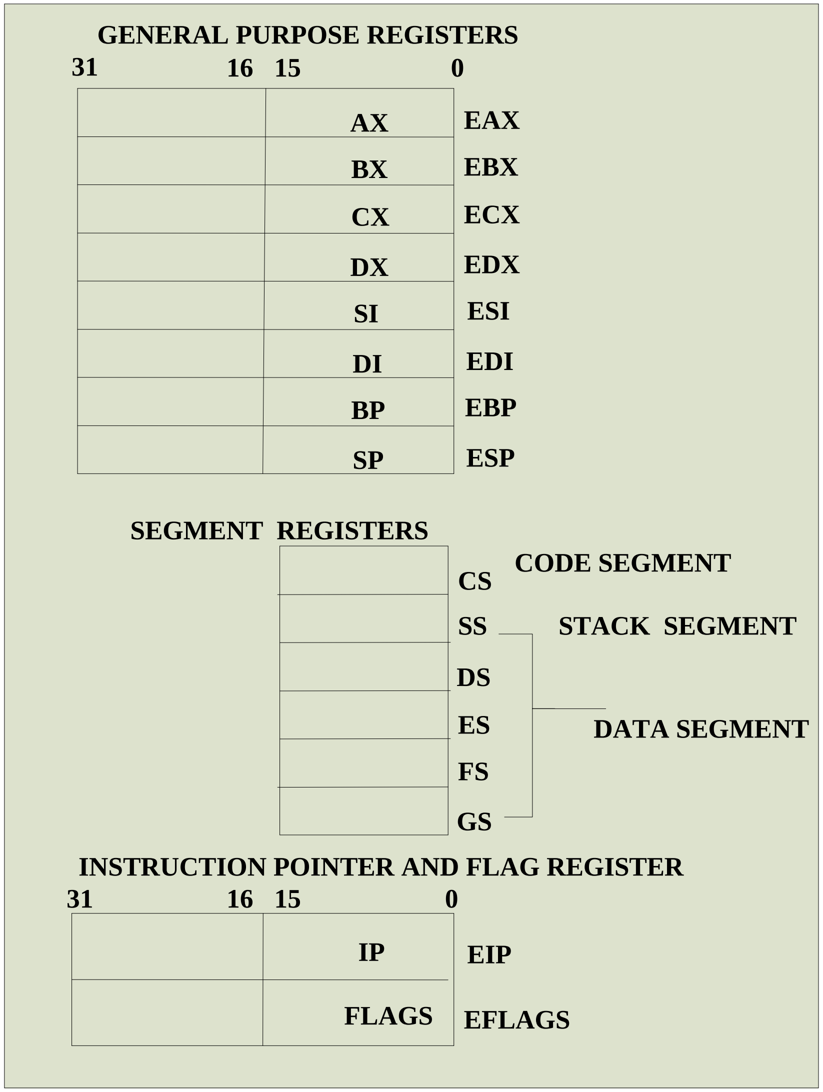
Next page

- An external parity circuit will evaluate requests to use the bus and grant bus use to the highest – priority master. To ask the 486 to release the bus , the bus controller asserts the 486 HOLD input or BOFF# input.
- If the HOLD input is asserted, the 486 will finish the current bus cycle, float its buses and assert the HLDA signal.
- To prevent another master from taking over the bus during a critical operation, the 486 can assert its LOCK# or PLOCK# signal.

EFLAG Register Of The 80486

- The extended flag register EFLAG is illustrated in the figure. The only new flag bit is the AC alignment check, used to indicate that the microprocessor has accessed a word at an odd address or a double word boundary.
- Efficient software and execution require that data be stored at word or doubleword boundaries.

Next page



GENERAL PURPOSE REGISTERS

31 16 15 0

AX EAX

BX EBX

CX ECX

DX EDX

SI ESI

DI EDI

BP EBP

SP ESP

SEGMENT REGISTERS

CS CODE SEGMENT

SS STACK SEGMENT

DS

ES DATA SEGMENT

FS

GS

INSTRUCTION POINTER AND FLAG REGISTER

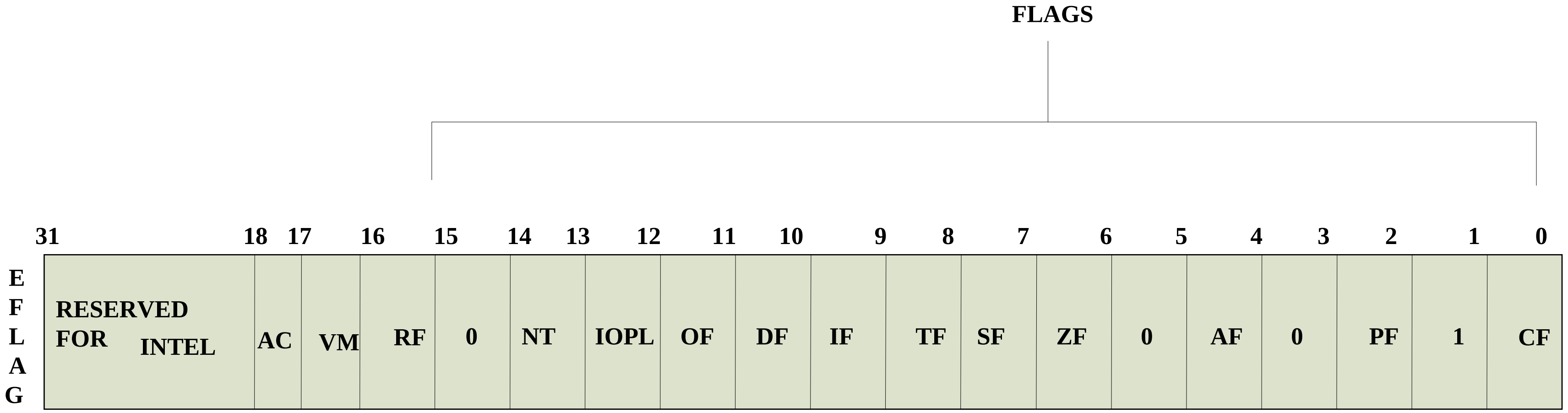
31 16 15 0

IP EIP

FLAGS EFLAGS

Next page

Flag Register of 80486



CF: Carry Flag

AF: Auxiliary carry

ZF: Zero Flag

SF : Sign Flag

TF : Trap Flag

IE : Interrupt Enable

AC : Alignment Check

DF : Direct Flag

OF : Over Flow

IOPL : I/O Privilege Level

NT : Nested Task Flag

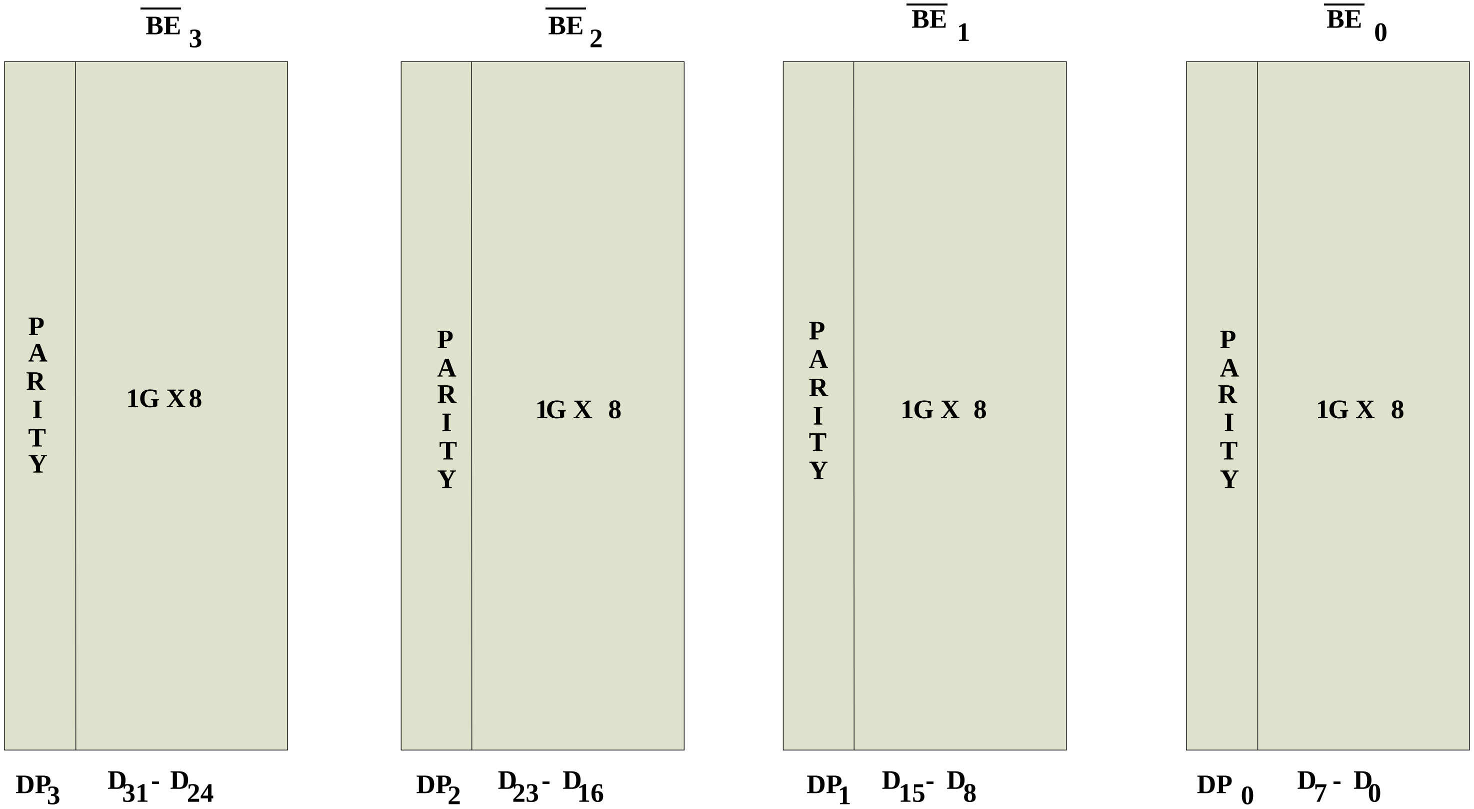
RF : Resume Flag

VM : Virtual Mode

80486 Memory System

- The memory system for the 486 is identical to 386 microprocessor. The 486 contains 4G bytes of memory beginning at location 00000000H and ending at FFFFFFFFH.
- The major change to the memory system is internal to 486 in the form of 8K byte cache memory, which speeds the execution of instructions and the acquisition of data.
- Another addition is the parity checker/ generator built into the 80486 microprocessor.
- ***Parity Checker / Generator*** : Parity is often used to determine if data are correctly read from a memory location. INTEL has incorporated an internal parity generator / decoder.

Next page



$\overline{BE}_3$ $\overline{BE}_2$ $\overline{BE}_1$ $\overline{BE}_0$

PARITY 1G X 8 | PARITY 1G X 8 | PARITY 1G X 8 | PARITY 1G X 8

DP_3 D_{31} - D_{24} | DP_2 D_{23} - D_{16} | DP_1 D_{15} - D_8 | DP_0 D_7 - D_0

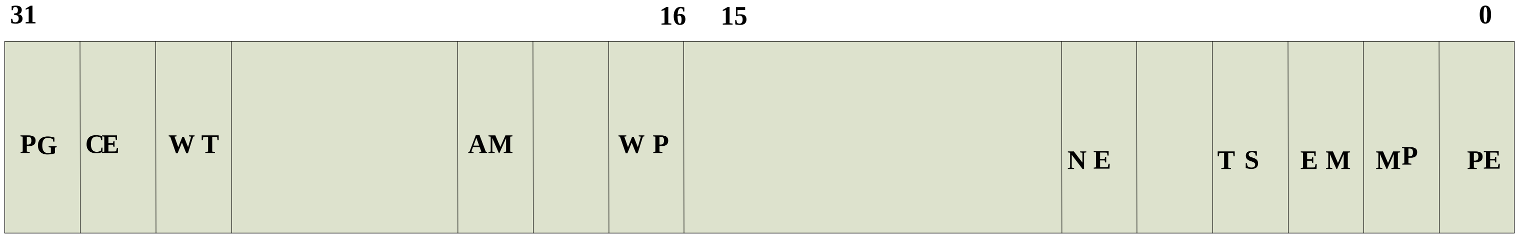
Next page

- Parity is generated by the 80486 during each write cycle. Parity is generated as even parity and a parity bit is provided for each byte of memory. The parity check bits appear on pins DP0-DP3, which are also parity inputs as well as parity outputs.
- These are typically stored in memory during each write cycle and read from memory during each read cycle.
- On a read, the microprocessor checks parity and generates a parity check error, if it occurs on the PCHK# pin. A parity error causes no change in processing unless the user applies the PCHK signal to an interrupt input.

Next page

- Interrupts are often used to signal a parity error in DS-based computer systems. This is same as 80386, except the parity bit storage.
- If parity is not used, Intel recommends that the DP0 – DP3 pins be pulled up to +5v.
- ***CACHE MEMORY****:* The cache memory system stores data used by a program and also the instructions of the program. The cache is organised as a 4 way set associative cache with each location containing 16 bytes or 4 doublewords of data.
- Control register CR0 is used to control the cache with two new control bits not present in the 80386 microprocessor.

Next page



31 16 15 0

PG CE WT AM WP NE TS EM MP PE

Control Register Zero (CR$_0$)For The **80486 Microprocessor**

- The CD (cache disable) , NW (non-cache write through) bits are new to the 80486 and are used to control the 8K byte cache.
- If the CD bit is a logic 1, all cache operations are inhibited. This setting is only used for debugging software and normally remains cleared. The NW bit is used to inhibit cache write-through operation. As with CD, cache write through is inhibited only for testing. For normal operations CD = 0 and NW = 0.
- Because the cache is new to 80486 microprocessor and the cache is filled using burst cycle not present on the 386.

80486 Memory Management

- The 80486 contains the same memory-management system as the 80386. This includes a paging unit to allow any 4K byte block of physical memory to be assigned to any 4K byte block of linear memory. The only difference between 80386 and 80486 memory-management system is paging.
- The 80486 paging system can disabled caching for section of translation memory pages, while the 80386 could not.
- If these are compared with 80386 entries, the addition of two new control bits is observed (PWT and PCD).
- The page write through and page cache disable bits control caching.

Next page

31 12	11 10 9	8	7	6	5	4	3	2	1	0
PAGE TABLE OR PAGE FRAME	OS BITS	O	O	D	A	P C D	P W T	U S	R W	P

Page Directory or Page Table Entry For The 80486 Microprocessor

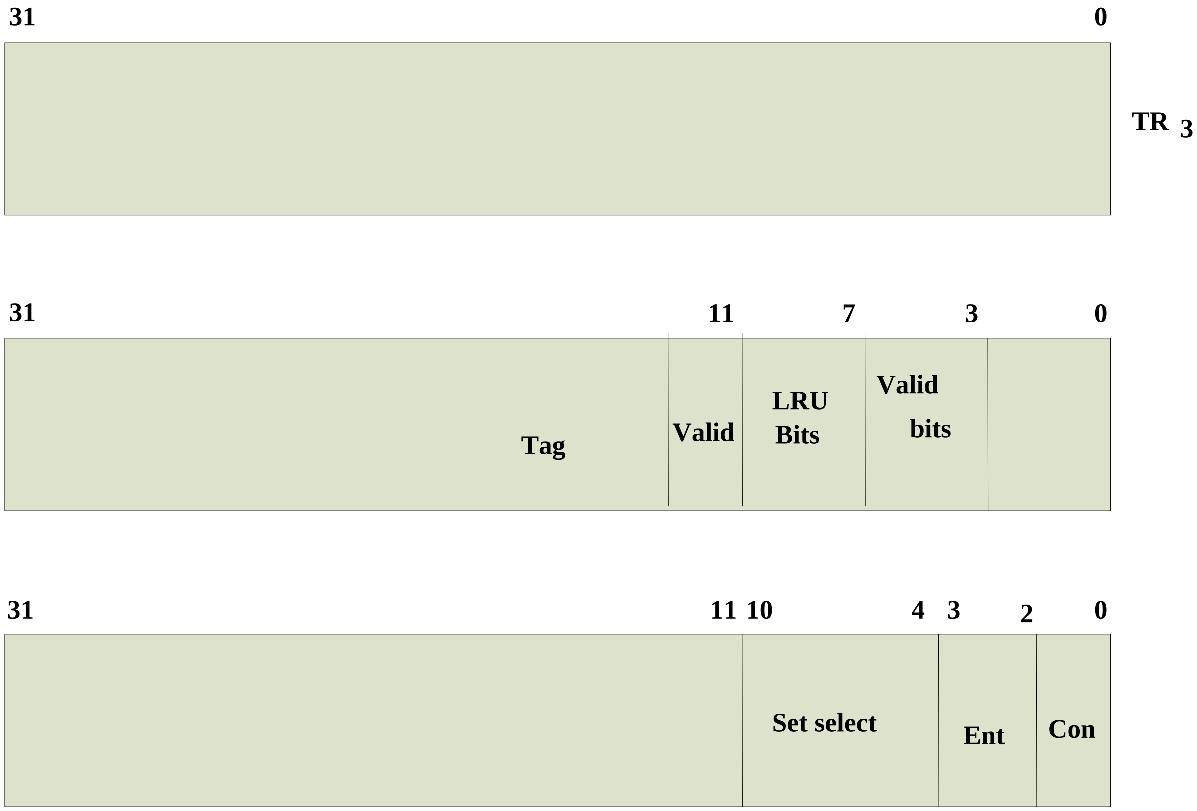
Next page

- The PWT controls how the cache functions for a write operation of the external cache memory. It does not control writing to the internal cache. The logic level of this bit is found on the PWT pin of the 80486 microprocessor. Externally, it can be used to dictate the write through policy of the external caching.
- The PCD bit controls the on-chip cache. If the PCD = 0, the on-chip cache is enabled for the current page of memory.
- Note that 80386 page table entries place a logic 0 in the PCD bit position, enabling caching. If PCD = 1, the on-chip cache is disable. Caching is disable regard less of condition of KEN#, CD, and NW.

Cache Test Registers

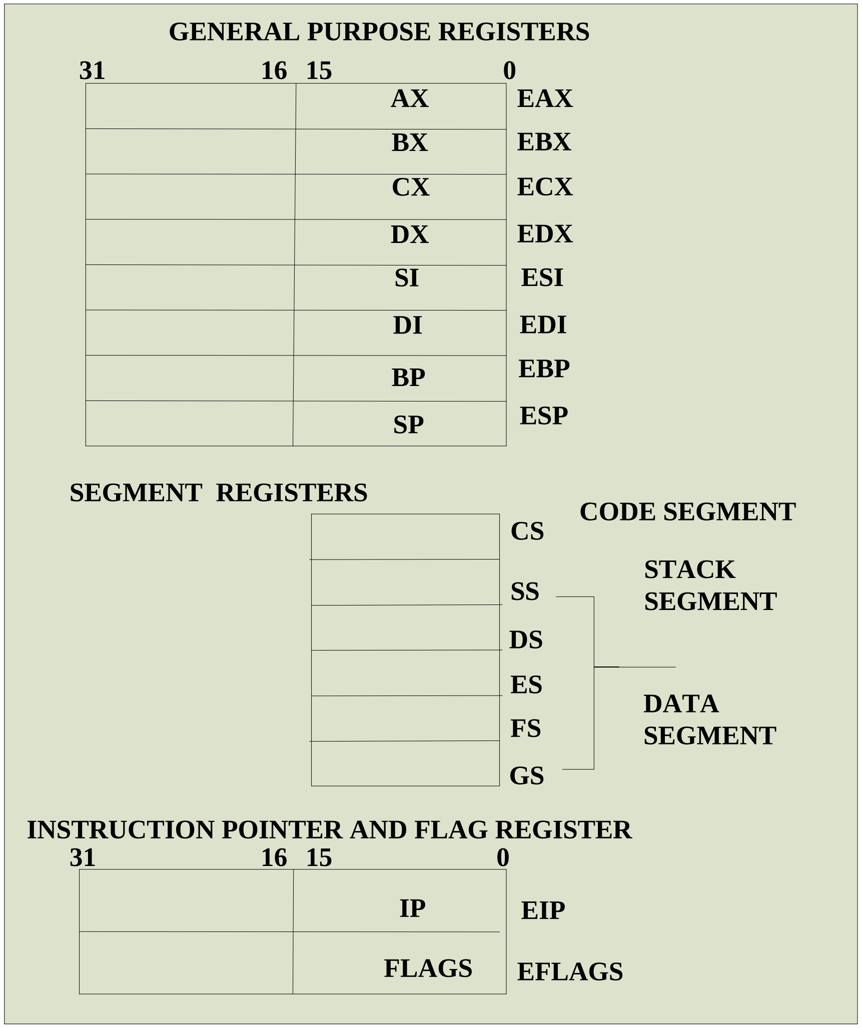
- The 80486 cache test registers are TR3, TR4, TR5.
- Cache data register (TR3) is used to access either the cache fill buffer for a write test operation or the cache read buffer for a cache read test operation.
- In order to fill or read a cache line (128 bits wide), TR3 must be written or read four times.
- The contents of the set select field in TR5 determine which internal cache line is written or read through TR3. The 7 bit test field selects one of the 128 different 16 byte wide cache lines. The entry select bits of TR5 select an entry in the set or the 32 bit location in the read buffer.

Next page



Cache test register of the 80486 microprocessor

Next page



GENERAL PURPOSE REGISTERS

31 16 15 0

AX EAX

BX EBX

CX ECX

DX EDX

SI ESI

DI EDI

BP EBP

SP ESP

SEGMENT REGISTERS

CS CODE SEGMENT

SS STACK SEGMENT

DS

ES

FS

GS

DATA SEGMENT

INSTRUCTION POINTER AND FLAG REGISTER

31 16 15 0

IP EIP

FLAGS EFLAGS

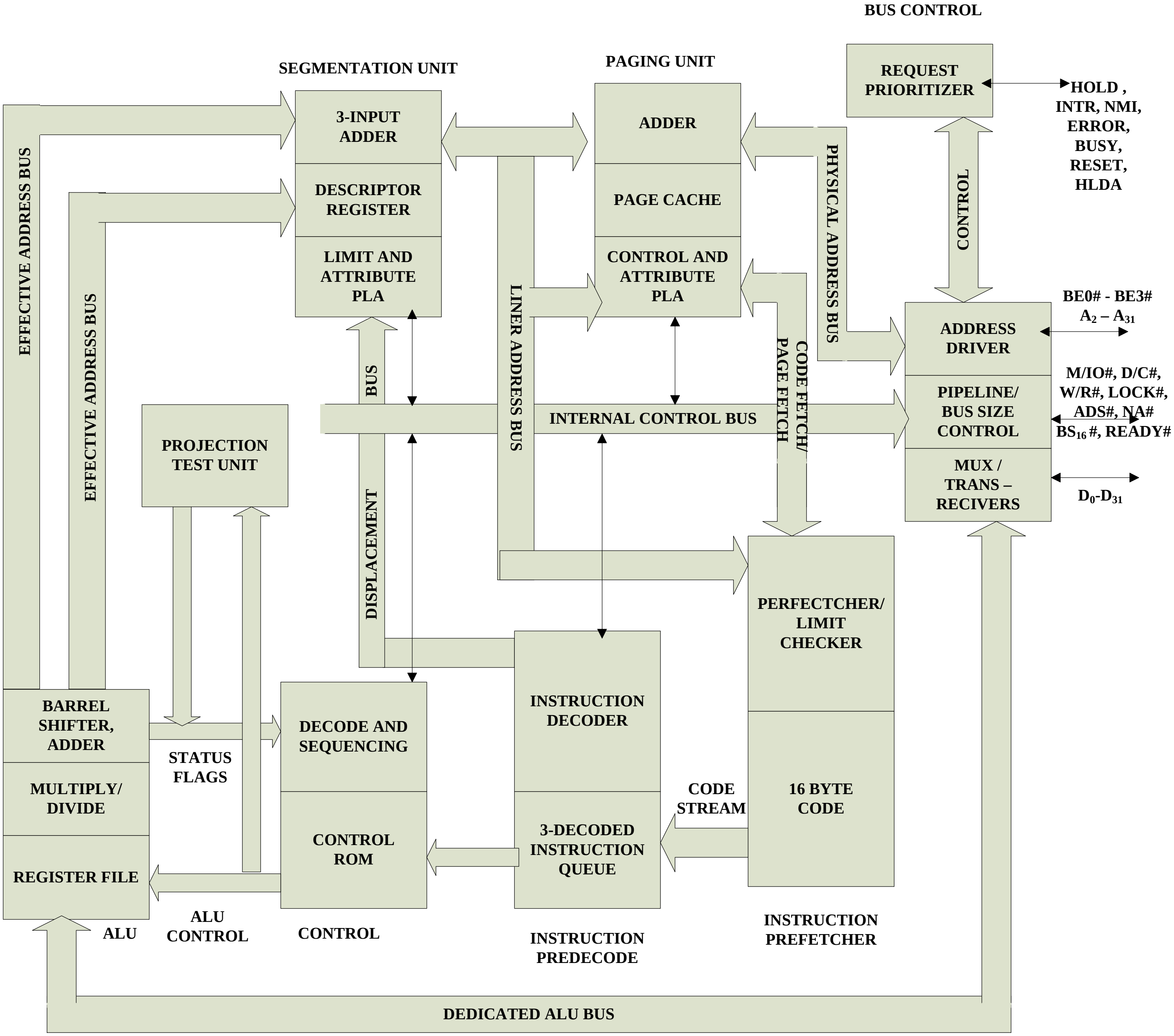
- The control bits in TR5 enable the fill buffer or read buffer operation (00)
- Perform a cache write (01), Perform a cache read (10)
- Flush the cache (11).
- The cache status register (TR4) hold the cache tag, LRU bits and a valid bit. This register is loaded with the tag and valid bit before a cache a cache write operation and contains the tag, valid bit, LRU bits, and 4 valid bits on a cache test read.
- Cache is tested each time that the microprocessor is reset if the AHOLD pin is high for 2 clocks prior to the RESET pin going low. This causes the 486 to completely test itself with a built in self test or BIST.

Next page

- The BIST uses TR3, TR4, TR5 to completely test the internal cache. Its outcome is reported in register EAX. If EAX is a zero, the microprocessor, the coprocessor and cache have passed the self test.
- The value of EAX can be tested after reset to determine if an error is detected. In most of the cases we do not directly access the test register unless we wish to perform our own tests on the cache or TLB.

Architecture of 80386

•The Internal Architecture of 80386 is divided into 3 sections.
•Central processing unit
•Memory management unit
•Bus interface unit
•Central processing unit is further divided into Execution unit and Instruction unit
•Execution unit has 8 General purpose and 8 Special purpose registers which are either used for handling data or calculating offset addresses.

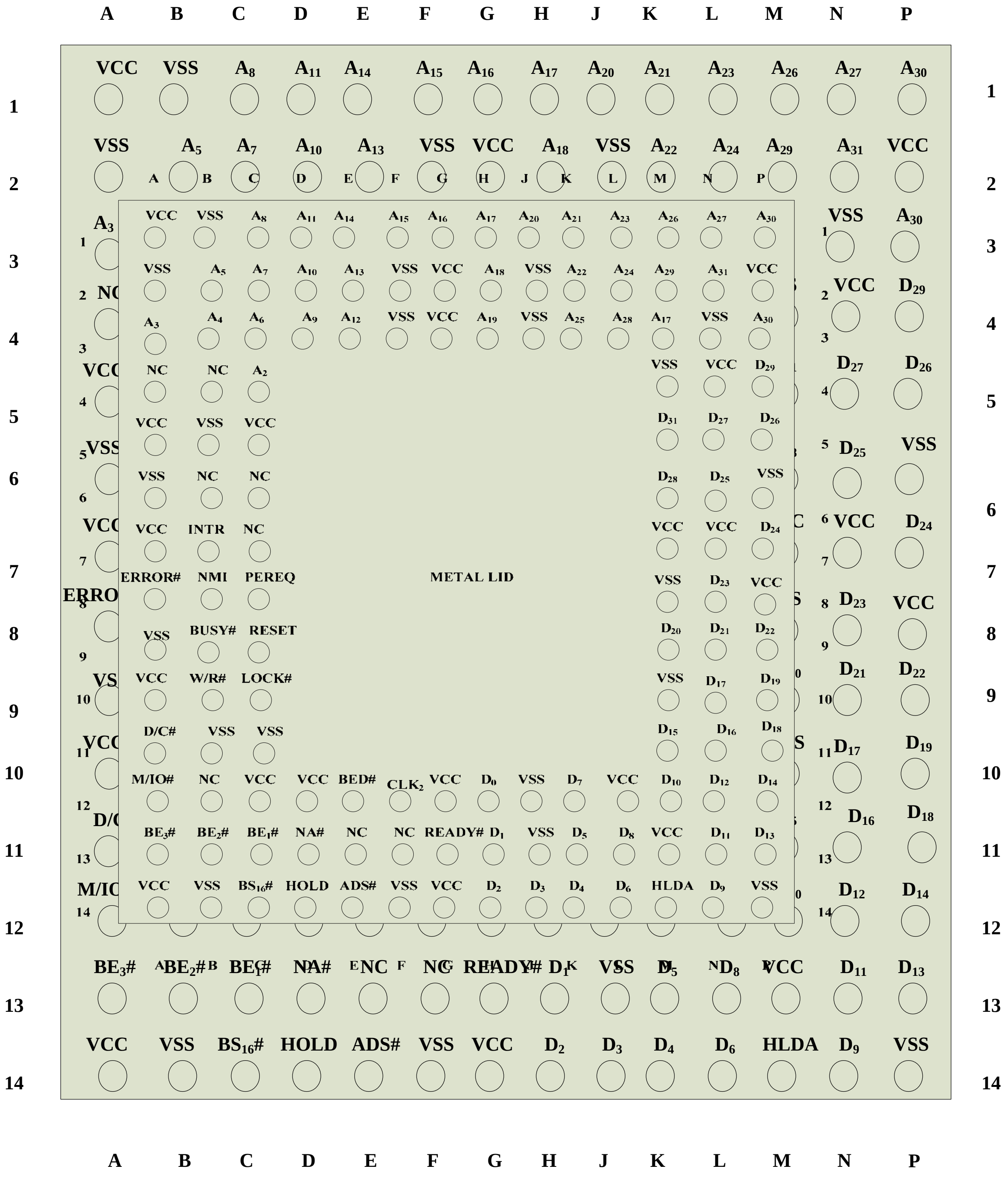


•The **Instruction unit** decodes the opcode bytes received from the 16-byte instruction code queue and arranges them in a 3- instruction decoded instruction queue.
•After decoding them pass it to the control section for deriving the necessary control signals. The barrel shifter increases the speed of all shift and rotate operations.
• The multiply / divide logic implements the bit-shift-rotate algorithms to complete the operations in minimum time.
•Even 32- bit multiplications can be executed within one microsecond by the multiply / divide logic.

•The Memory management unit consists of a Segmentation unit and a Paging unit.
•Segmentation unit allows the use of two address components, viz. segment and offset for relocability and sharing of code and data.
•Segmentation unit allows segments of size 4Gbytes at max.
•The Paging unit organizes the physical memory in terms of pages of 4kbytes size each.
•Paging unit works under the control of the segmentation unit, i.e. each segment is further divided into pages. The virtual memory is also organizes in terms of segments and pages by the memory management unit.

•The Segmentation unit provides a 4 level protection mechanism for protecting and isolating the system code and data from those of the application program.
•Paging unit converts linear addresses into physical addresses.
•The control and attribute PLA checks the privileges at the page level. Each of the pages maintains the paging information of the task. The limit and attribute PLA checks segment limits and attributes at segment level to avoid invalid accesses to code and data in the memory segments.
•The Bus control unit has a prioritizer to resolve the priority of the various bus requests. This controls the access of the bus. The address driver drives the bus enable and address signal $A_0 – A_{31.}$ The pipeline and dynamic bus sizing unit handle the related control signals.
•The data buffers interface the internal data bus with the system bus.

PIN DIAGRAM OF 80386


A B C D E F G H J K L M N P
1 VCC VSS A8 A11 A14 A15 A16 A17 A20 A21 A23 A26 A27 A30
2 VSS A5 A7 A10 A13 VSS VCC A18 VSS A22 A24 A29 A31 VCC
3 A3 A4 A6 A9 A12 VSS VCC A19 VSS A25 A28 A17 VSS A30
4 NC NC A2 VSS VCC D29
5 VCC VSS VCC D31 D27 D26
6 VSS NC NC D28 D25 VSS
7 VCC INTR NC VCC VCC D24
8 ERROR# NMI PEREQ VSS D23 VCC
METAL LID
9 VSS BUSY# RESET D20 D21 D22
10 VCC W/R# LOCK# VSS D17 D19
11 D/C# VSS VSS D15 D16 D18
12 M/IO# NC VCC VCC BED# CLK2 VCC D0 VSS D7 VCC D10 D12 D14
13 BE3# BE2# BE1# NA# NC NC READY# D1 VSS D5 D8 VCC D11 D13
14 VCC VSS BS16# HOLD ADS# VSS VCC D2 D3 D4 D6 HLDA D9 VSS
A B C D E F G H J K L M N P

Signal Descriptions of 80386

•CLK_2 :The input pin provides the basic system clock timing for the operation of 80386.
•D_0 – D_{31}:These 32 lines act as bidirectional data bus during different access cycles.
•A_{31} – A_2: These are upper 30 bit of the 32- bit address bus.
•BE_0 to BE_3: The 32- bit data bus supported by 80386 and the memory system of 80386 can be viewed as a 4- byte wide memory access mechanism. The 4 byte enable lines BE_0 to BE_3, may be used for enabling these 4 blanks. Using these 4 enable signal lines, the CPU may transfer 1 byte / 2 / 3 / 4 byte of data simultaneously.
•**ADS#:** The address status output pin indicates that the address bus and bus cycle definition pins(W/R#, D/C#, M/IO#, BE_0# to BE_3#) are carrying the respective valid signals. The 80383 does not have any ALE signals and so this signals may be used for latching the address to external latches.
•**READY#:** The ready signals indicates to the CPU that the previous bus cycle has been terminated and the bus is ready for the next cycle. The signal is used to insert WAIT states in a bus cycle and is useful for interfacing of slow devices with CPU.
•**VCC**: These are system power supply lines.
•**VSS**: These return lines for the power supply.
•**BS_{16}#:** The bus size – 16 input pin allows the interfacing of 16 bit devices with the 32 bit wide 80386 data bus. Successive 16 bit bus cycles may be executed to read a 32 bit data from a peripheral.
•**HOLD**: The bus hold input pin enables the other bus masters to gain control of the system bus if it is asserted.
•**HLDA**: The bus hold acknowledge output indicates that a valid bus hold request has been received and the bus has been relinquished by the CPU.

•**BUSY#:** The busy input signal indicates to the CPU that the coprocessor is busy with the allocated task.

•**ERROR#:** The error input pin indicates to the CPU that the coprocessor has encountered an error while executing its instruction.

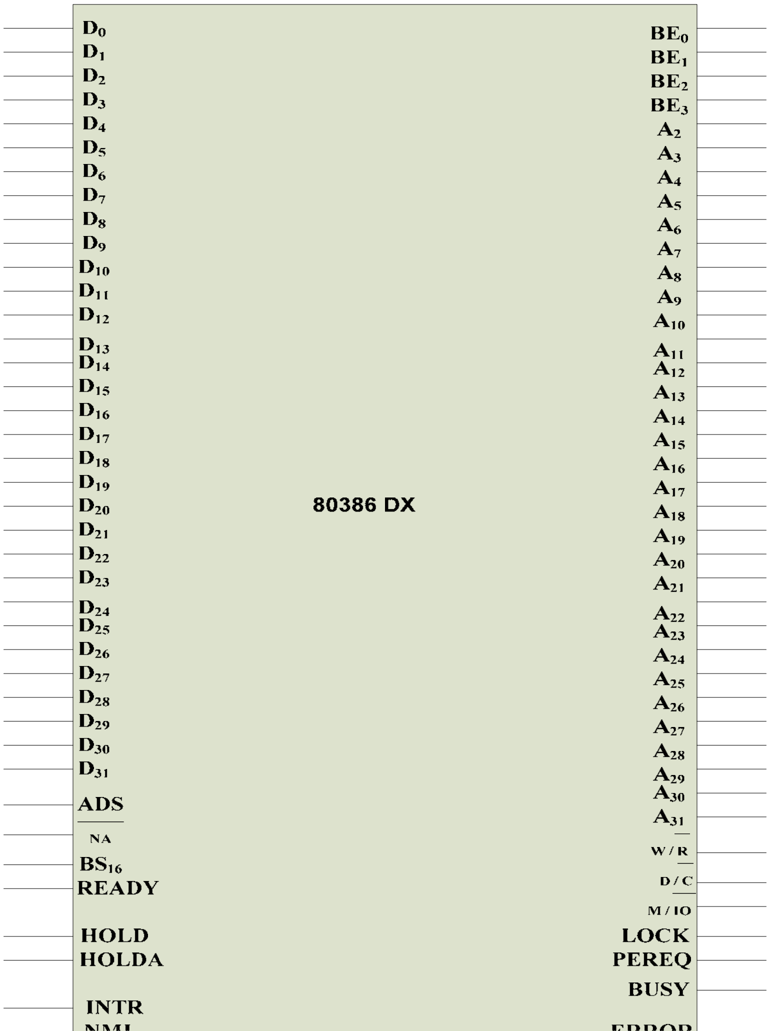
•**PEREQ**: The processor extension request output signal indicates to the CPU to fetch a data word for the coprocessor.

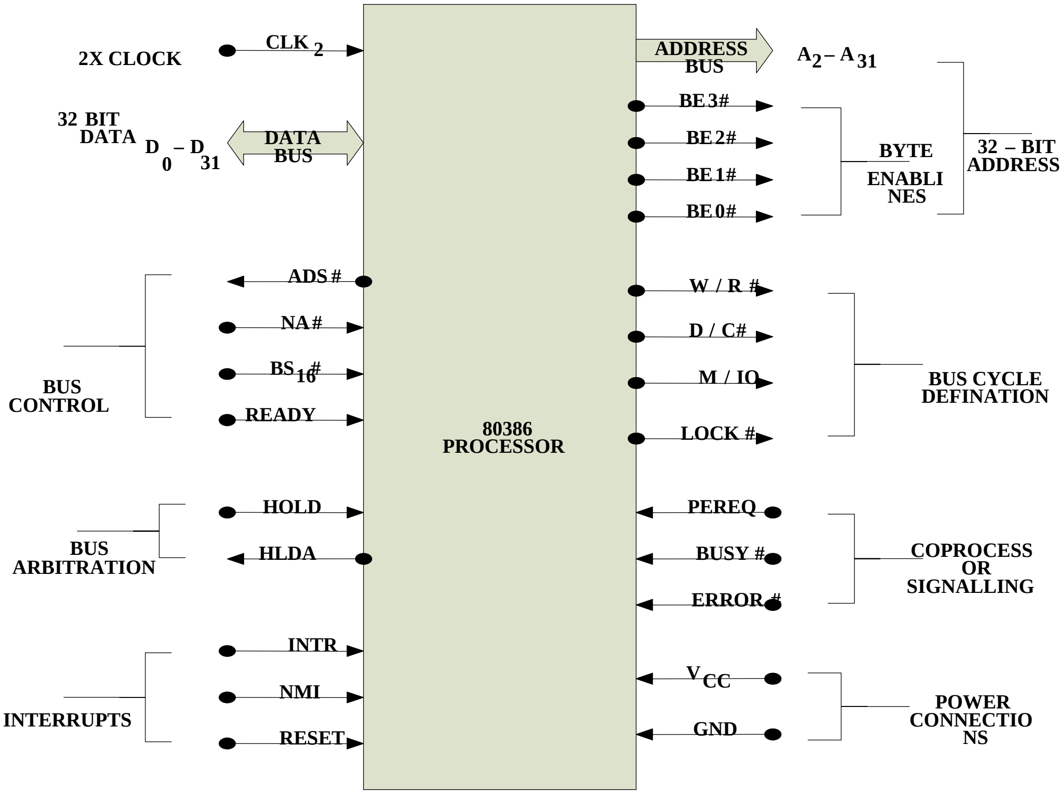
•**INTR**: This interrupt pin is a maskable interrupt, that can be masked using the IF of the flag register.

•**NMI:** A valid request signal at the non-maskable interrupt request input pin internally generates a non- maskable interrupt of type2.

•**RESET**: A high at this input pin suspends the current operation and restart the execution from the starting location.

•**N / C** : No connection pins are expected to be left open while connecting the 80386 in the circuit.

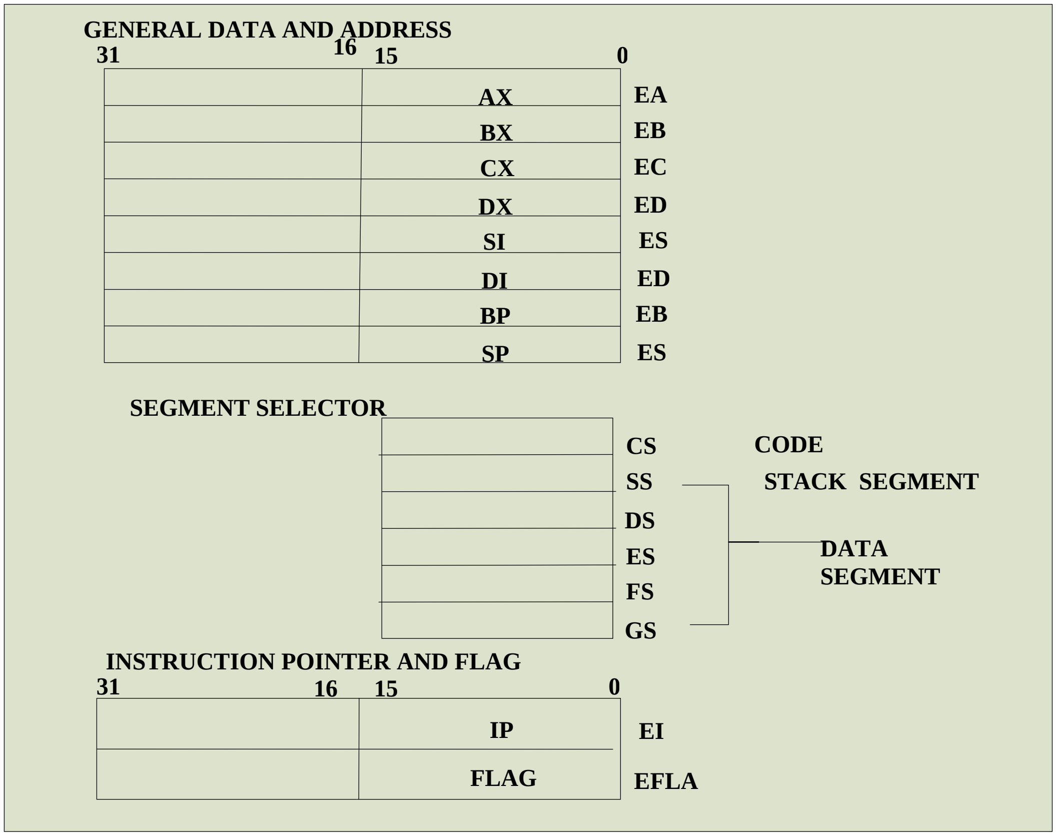


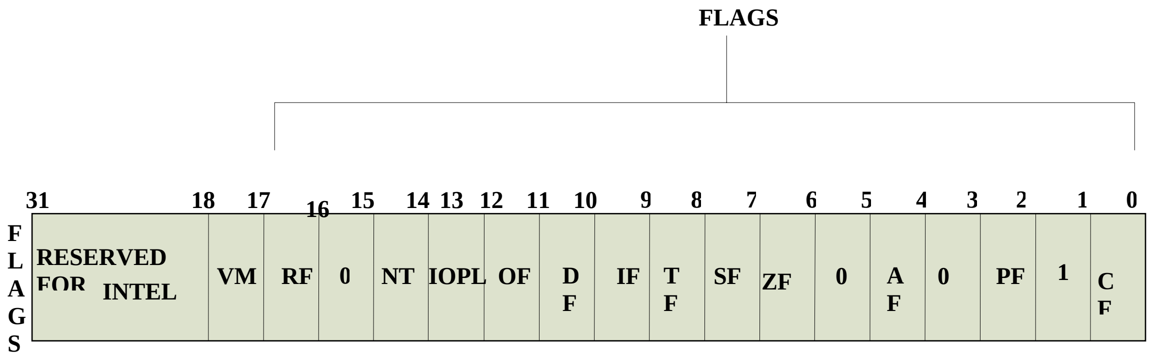


Register Organisation

•The 80386 has eight 32 - bit general purpose registers which may be used as either 8 bit or 16 bit registers.

- A 32 - bit register known as an extended register, is represented by the register name with prefix E.
- Example : A 32 bit register corresponding to AX is EAX, similarly BX is EBX etc.
- The 16 bit registers BP, SP, SI and DI in 8086 are now available with their extended size of 32 bit and are names as EBP,ESP,ESI and EDI.
- AX represents the lower 16 bit of the 32 bit register EAX.
- BP, SP, SI, DI represents the lower 16 bit of their 32 bit counterparts, and can be used as independent 16 bit registers.
- The six segment registers available in 80386 are CS, SS, DS, ES, FS and GS.
- The CS and SS are the code and the stack segment registers respectively, while DS, ES, FS, GS are 4 data segment registers.
- A 16 bit instruction pointer IP is available along with 32 bit counterpart EIP.





FLAG REGISTER OF 80386

•***Flag Register of 80386***: The Flag register of 80386 is a 32 bit register. Out of the 32 bits, Intel has reserved bits D_{18} to D_{31}, D_5 and D_3, while D_1 is always set at 1.Two extra new flags are added to the 80286 flag to derive the flag register of 80386. They are VM and RF flags.

•**VM - *Virtual Mode Flag***: If this flag is set, the 80386 enters the virtual 8086 mode within the protection mode. This is to be set only when the 80386 is in protected mode. In this mode, if any privileged instruction is executed an exception 13 is generated. This bit can be set using IRET instruction or any task switch operation only in the protected mode.

•***RF- Resume Flag***: This flag is used with the debug register breakpoints. It is checked at the starting of every instruction cycle and if it is set, any debug fault is ignored during the instruction cycle. The RF is automatically reset after successful execution of every instruction, except for IRET and POPF instructions.

•Also, it is not automatically cleared after the successful execution of JMP, CALL and INT instruction causing a task switch. These instruction are used to set the RF to the value specified by the memory data available at the stack.

•***Segment Descriptor Registers***: This registers are not available for programmers, rather they are internally used to store the descriptor information, like attributes, limit and base addresses of segments.

•The six segment registers have corresponding six 73 bit descriptor registers. Each of them contains 32 bit base address, 32 bit base limit and 9 bit attributes. These are automatically loaded when the corresponding segments are loaded with selectors.

•***Control Registers***: The 80386 has three 32 bit control registers CR), CR_2 and CR_3 to hold global machine status independent of the executed task. Load and store instructions are available to access these registers.

•***System Address Registers***: Four special registers are defined to refer to the descriptor tables supported by 80386.

•The 80386 supports four types of descriptor table, viz. global descriptor table (GDT), interrupt descriptor table (IDT), local descriptor table (LDT) and task state segment descriptor (TSS).

•***Debug and Test Registers***: Intel has provide a set of 8 debug registers for hardware debugging. Out of these eight registers DR_0 to DR_7, two registers DR_4 and DR_5 are Intel reserved.

•The initial four registers DR_0 to DR_3 store four program controllable breakpoint addresses, while DR_6 and DR_7 respectively hold breakpoint status and breakpoint control information.

•Two more test register are provided by 80386 for page cacheing namely test control and test status register.

•***ADDRESSING MODES***: The 80386 supports overall eleven addressing modes to facilitate efficient execution of higher level language programs.

•In case of all those modes, the 80386 can now have 32-bit immediate or 32- bit register operands or displacements.

•The 80386 has a family of scaled modes. In case of scaled modes, any of the index register values can be multiplied by a valid scale factor to obtain the displacement.

•The valid scale factor are 1, 2, 4 and 8.

•The different scaled modes are as follows.

•***Scaled Indexed Mode***: Contents of the an index register are multiplied by a scale factor that may be added further to get the operand offset.

•***Based Scaled Indexed Mode***: Contents of the an index register are multiplied by a scale factor and then added to base register to obtain the offset.

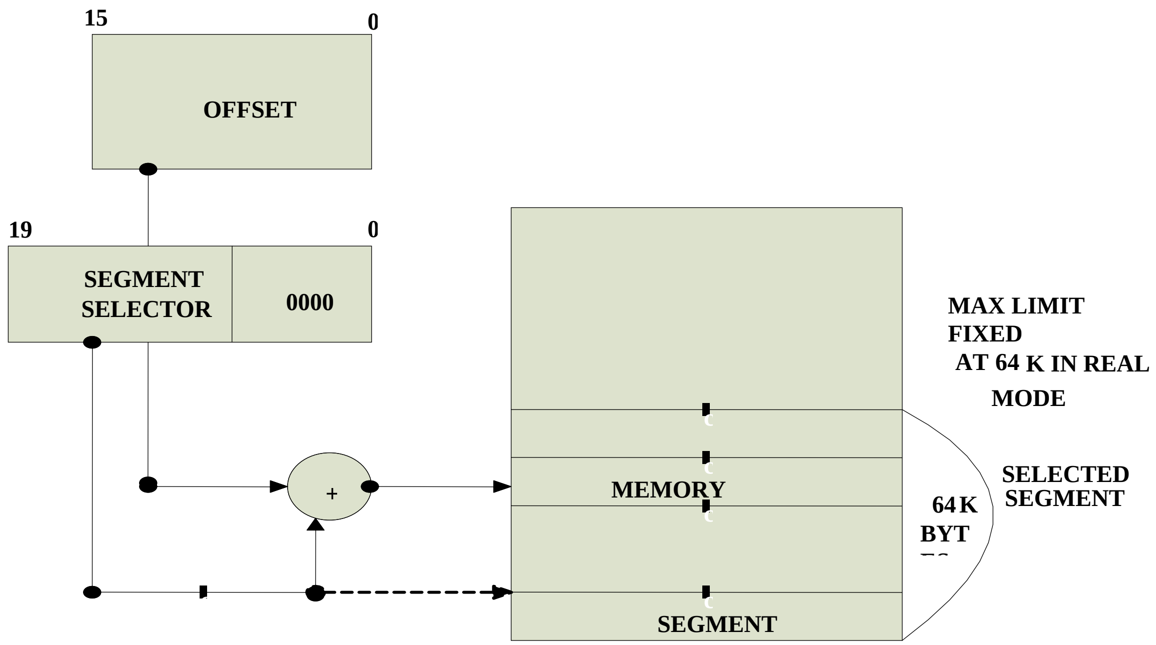
•***Based Scaled Indexed Mode with Displacement***: The Contents of the an index register are multiplied by a scaling factor and the result is added to a base register and a displacement to get the offset of an operand.

Real Address Mode of 80386

•After reset, the 80386 starts from memory location FFFFFFF0H under the real address mode. In the real mode, 80386 works as a fast 8086 with 32-bit registers and data types.

•In real mode, the default operand size is 16 bit but 32- bit operands and addressing modes may be used with the help of override prefixes.

•The segment size in real mode is 64k, hence the 32-bit effective addressing must be less than 0000FFFFFH. The real mode initializes the 80386 and prepares it for protected mode.



Physical Address Formation In Real Mode Of 80386

•***Memory Addressing in Real Mode***: In the real mode, the 80386 can address at the most 1Mbytes of physical memory using address lines A_0-A_{19}.

•Paging unit is disabled in real addressing mode, and hence the real addresses are the same as the physical addresses.

•To form a physical memory address, appropriate segment registers contents (16-bits) are shifted left by four positions and then added to the 16-bit offset address formed using one of the addressing modes, in the same way as in the 80386 real address mode.

•The segment in 80386 real mode can be read, write or executed, i.e. no protection is available.

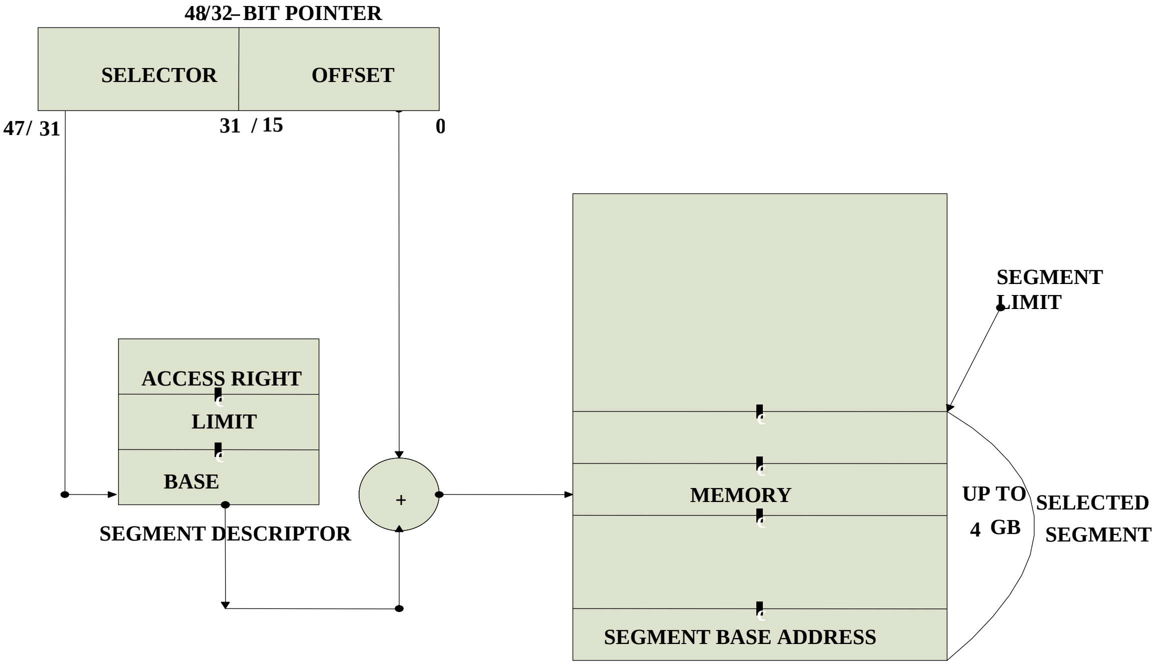
•Any fetch or access past the end of the segment limit generate exception 13 in real address mode.

•The segments in 80386 real mode may be overlapped or non-overlapped.

•The interrupt vector table of 80386 has been allocated 1Kbyte space starting from 00000H to 003FFH.

Protected Mode of 80386

•All the capabilities of 80386 are available for utilization in its protected mode of operation.
•The 80386 in protected mode support all the software written for 80286 and 8086 to be executed under the control of memory management and protection abilities of 80386.
•The protected mode allows the use of additional instruction, addressing modes and capabilities of 80386.



Protected Mode Addressing Without Paging Unit

•***ADDRESSING IN PROTECTED MODE***: In this mode, the contents of segment registers are used as selectors to address descriptors which contain the segment limit, base address and access rights byte of the segment.

•The effective address (offset) is added with segment base address to calculate linear address. This linear address is further used as physical address, if the paging unit is disabled, otherwise the paging unit converts the linear address into physical address.
•The paging unit is a memory management unit enabled only in protected mode. The paging mechanism allows handling of large segments of memory in terms of pages of 4Kbyte size.
•The paging unit operates under the control of segmentation unit. The paging unit if enabled converts linear addresses into physical address, in protected mode.

Segmentation

•***DESCRIPTOR TABLES***: These descriptor tables and registers are manipulated by the operating system to ensure the correct operation of the processor, and hence the correct execution of the program.

•Three types of the 80386 descriptor tables are listed as follows:
•GLOBAL DESCRIPTOR TABLE (GDT)
•LOCAL DESCRIPTOR TABLE (LDT)
•INTERRUPT DESCRIPTOR TABLE (IDT)
•***DESCRIPTORS***: The 80386 descriptors have a 20-bit segment limit and 32-bit segment address. The descriptor of 80386 are 8-byte quantities access right or attribute bits along with the base and limit of the segments.
•***Descriptor Attribute Bits***: The A (accessed) attributed bit indicates whether the segment has been accessed by the CPU or not.
•The TYPE field decides the descriptor type and hence the segment type.

•The S bit decides whether it is a system descriptor (S=0) or code/data segment descriptor (S=1).
•The DPL field specifies the descriptor privilege level.
•The D bit specifies the code segment operation size. If D=1, the segment is a 32-bit operand segment, else, it is a 16-bit operand segment.
•The P bit (present) signifies whether the segment is present in the physical memory or not. If P=1, the segment is present in the physical memory.
•The G (granularity) bit indicates whether the segment is page addressable. The zero bit must remain zero for compatibility with future process.

•The AVL (available) field specifies whether the descriptor is for user or for operating system.
•The 80386 has five types of descriptors listed as follows:
1.Code or Data Segment Descriptors.
2.System Descriptors.
3.Local descriptors.
4.TSS (Task State Segment) Descriptors.
5.GATE Descriptors.
•The 80386 provides a four level protection mechanism exactly in the same way as the 80286 does.

ADDRESS

BYTE

<table>
<tr><td colspan="6">31</td><td colspan="6">0</td><td></td></tr>
<tr><td colspan="6">SEGMENT BASE 15.0</td><td colspan="6">SEGMENT 15 ..0</td><td></td></tr>
<tr><td>BAS 3.2</td><td>G</td><td>D</td><td>0</td><td>AV</td><td>LIMIT
19..16</td><td>P</td><td>DPL</td><td>S</td><td>TYPE</td><td>A</td><td>BASE
23.2</td><td>+4</td></tr>
</table>

Structure of An Descriptor

BASE Base Address of the segment

LIMIT The length of the segment

P Present Bit - 1=Present ,0 = not present

S Segment Descriptor -0 = System Descriptor,
1 = Code or data segment descriptor

TYPE Type of segment

G Granularity Bit - 1=Segment length is page granular ,
0 = Segment length is byte granular

D Default Operation size

0 Bit must be zero

AVL Available field for user or OS

Paging

•***PAGING OPERATION***: Paging is one of the memory management techniques used for virtual memory multitasking operating system.
•The segmentation scheme may divide the physical memory into a variable size segments but the paging divides the memory into a fixed size pages.
•The segments are supposed to be the logical segments of the program, but the pages do not have any logical relation with the program.
•The pages are just fixed size portions of the program module or data.
•The advantage of paging scheme is that the complete segment of a task need not be in the physical memory at any time.
•Only a few pages of the segments, which are required currently for the execution need to be available in the physical memory. Thus the memory requirement of the task is substantially reduced, relinquishing the available memory for other tasks.
•Whenever the other pages of task are required for execution, they may be fetched from the secondary storage.
•The previous page which are executed, need not be available in the memory, and hence the space occupied by them may be relinquished for other tasks.
•Thus paging mechanism provides an effective technique to manage the physical memory for multitasking systems.

•***Paging Unit***: The paging unit of 80386 uses a two level table mechanism to convert a linear address provided by segmentation unit into physical addresses.
•The paging unit converts the complete map of a task into pages, each of size 4K. The task is further handled in terms of its page, rather than segments.
•The paging unit handles every task in terms of three components namely page directory, page tables and page itself.
•***Paging Descriptor Base Register***: The control register CR_2 is used to store the 32-bit linear address at which the previous page fault was detected.
•The CR_3 is used as page directory physical base address register, to store the physical starting address of the page directory.
•The lower 12 bit of the CR_3 are always zero to ensure the page size aligned directory. A move operation to CR_3 automatically loads the page table entry caches and a task switch operation, to load CR_0 suitably.
•***Page Directory*** : This is at the most 4Kbytes in size. Each directory entry is of 4 bytes, thus a total of 1024 entries are allowed in a directory.
•The upper 10 bits of the linear address are used as an index to the corresponding page directory entry. The page directory entries point to page tables.
•***Page Tables***: Each page table is of 4Kbytes in size and many contain a maximum of 1024 entries. The page table entries contain the starting address of the page and the statistical information about the page.
•The upper 20 bit page frame address is combined with the lower 12 bit of the linear address. The address bits A_{12}- A_{21} are used to select the 1024 page table entries. The page table can be shared between the tasks.
•The P bit of the above entries indicate, if the entry can be used in address translation.
•If P=1, the entry can be used in address translation, otherwise it cannot be used.
•The P bit of the currently executed page is always high.
•The accessed bit A is set by 80386 before any access to the page. If A=1, the page is accessed, else unaccessed.
•The D bit (Dirty bit) is set before a write operation to the page is carried out. The D-bit is undefined for page director entries.
•The OS reserved bits are defined by the operating system software.
•The User / Supervisor (U/S) bit and read/write bit are used to provide protection. These bits are decoded to provide protection under the 4 level protection model.
•The level 0 is supposed to have the highest privilege, while the level 3 is supposed to have the least privilege.
•This protection provide by the paging unit is transparent to the segmentation unit.

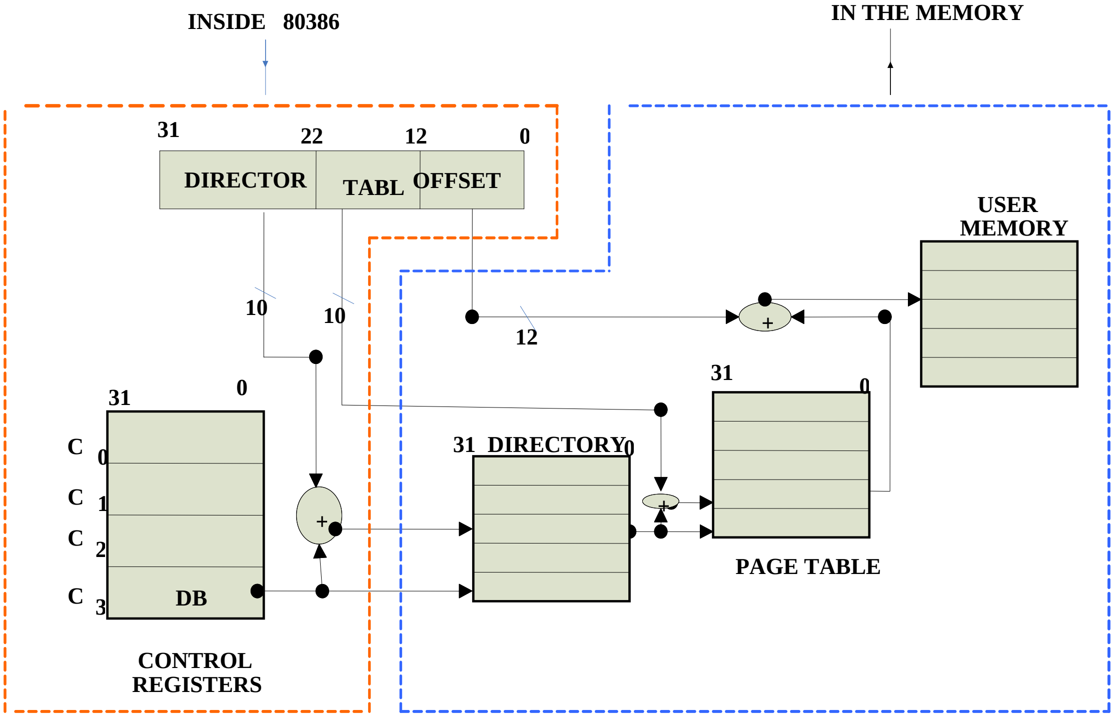
PAGE TABLE 31…1	O RESERV	0	0	D	A	0	0	U - S	R - W	P

PAGE DIRECTORY ENTRY

PAGE FRAME ADDRESS 31…1	OS RESERVE	0	0	D	A	0	0	U - S	R - W	P

PAGE TABLE ENTRY

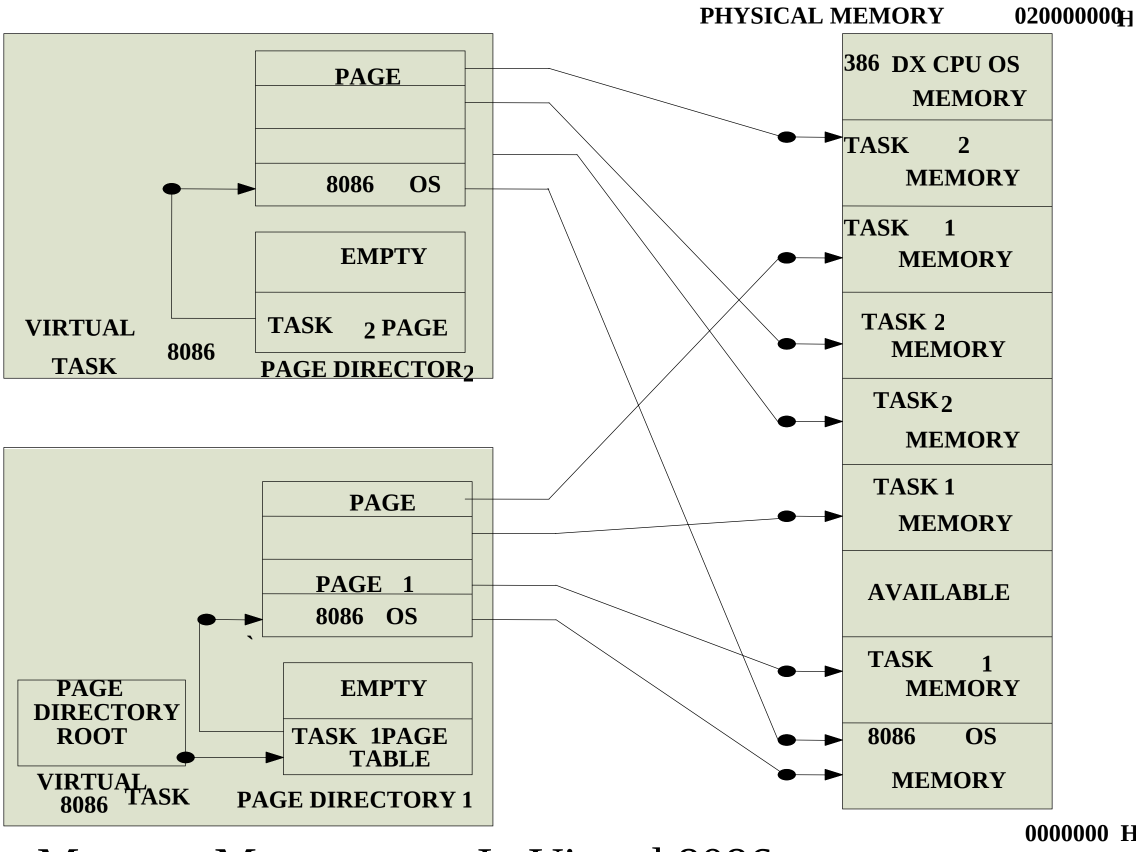
U - S	R - W	PERMITTED FOR LEVEL 3	PERMITTED LEVEL2 ,1OR 0
0	0	NONE	READ / WRITE
0	1	NONE	READ/ WRIT
1	0	READ	READ / WRITE
1	1	READ-WRITE	READ/ WRITE



DBA Physical directory base address

Virtual 8086 Mode

•In its protected mode of operation, 80386DX provides a virtual 8086 operating environment to execute the 8086 programs.
•The real mode can also used to execute the 8086 programs along with the capabilities of 80386, like protection and a few additional instructions.
•Once the 80386 enters the protected mode from the real mode, it cannot return back to the real mode without a reset operation.
•Thus, the virtual 8086 mode of operation of 80386, offers an advantage of executing 8086 programs while in protected mode.
•The address forming mechanism in virtual 8086 mode is exactly identical with that of 8086 real mode.
•In virtual mode, 8086 can address 1Mbytes of physical memory that may be anywhere in the 4Gbytes address space of the protected mode of 80386.
•Like 80386 real mode, the addresses in virtual 8086 mode lie within 1Mbytes of memory.
•In virtual mode, the paging mechanism and protection capabilities are available at the service of the programmers.
•The 80386 supports multiprogramming, hence more than one programmer may be use the CPU at a time.



Memory Management In Virtual 8086

•Paging unit may not be necessarily enable in virtual mode, but may be needed to run the 8086 programs which require more than 1Mbyts of memory for memory management function.

•In virtual mode, the paging unit allows only 256 pages, each of 4Kbytes size.

•Each of the pages may be located anywhere in the maximum 4Gbytes physical memory. The virtual mode allows the multiprogramming of 8086 applications.

•The virtual 8086 mode executes all the programs at privilege level 3.Any of the other programmes may deny access to the virtual mode programs or data.

•However, the real mode programs are executed at the highest privilege level, i.e. level 0.

•The virtual mode may be entered using an IRET instruction at CPL=0 or a task switch at any CPL, executing any task whose TSS is having a flag image with VM flag set to 1.

•The IRET instruction may be used to set the VM flag and consequently enter the virtual mode.

•The PUSHF and POPF instructions are unable to read or set the VM bit, as they do not access it.

•Even in the virtual mode, all the interrupts and exceptions are handled by the protected mode interrupt handler.
•To return to the protected mode from the virtual mode, any interrupt or execution may be used.
•As a part of interrupt service routine, the VM bit may be reset to zero to pull back the 80386 into protected mode.

Features of 80386

•This 80386 is a 32bit processor that supports, 8bit/32bit data operands.
•The 80386 instruction set is upward compatible with all its predecessors.
•The 80386 can run 8086 applications under protected mode in its virtual 8086 mode of operation.
•With the 32 bit address bus, the 80386 can address upto 4Gbytes of physical memory. The physical memory is organised in terms of segments of 4Gbytes at maximum.
•The 80386 CPU supports 16K number of segments and thus the total virtual space of 4Gbytes * 16K = 64 Terrabytes.
•The memory management section of 80386 supports the virtual memory, paging and four levels of protection, maintaining full compatibility with 80286.
•The 80386 offers a set of 8 debug registers DR_0-DR_7 for hardware debugging and control. The 80386 has on-chip address translation cache.
•The concept of paging is introduced in 80386 that enables it to organise the available physical memory in terms of pages of size 4Kbytes each, under the segmented memory.
•The 80386 can be supported by 80387 for mathematical data processing.

80486 Microprocessor

•The 32-bit 80486 is the next evolutionary step up from the 80386.
•One of the most obvious feature included in a 80486 is a built in math coprocessor. This coprocessor is essentially the same as the 80387 processor used with a 80386, but being integrated on the chip allows it to execute math instructions about three times as fast as a 80386/387 combination.
•80486 is an 8Kbyte code and data cache.
•To make room for the additional signals, the 80486 is packaged in a 168 pin, pin grid array package instead of the 132 pin PGA used for the 80386.

Pin Definitions

•**A_{31}-A_2** : Address outputs A31-A2 provide the memory and I/O with the address during normal operation. During a cache line invalidation A31-A4 are used to drive the microprocessor.

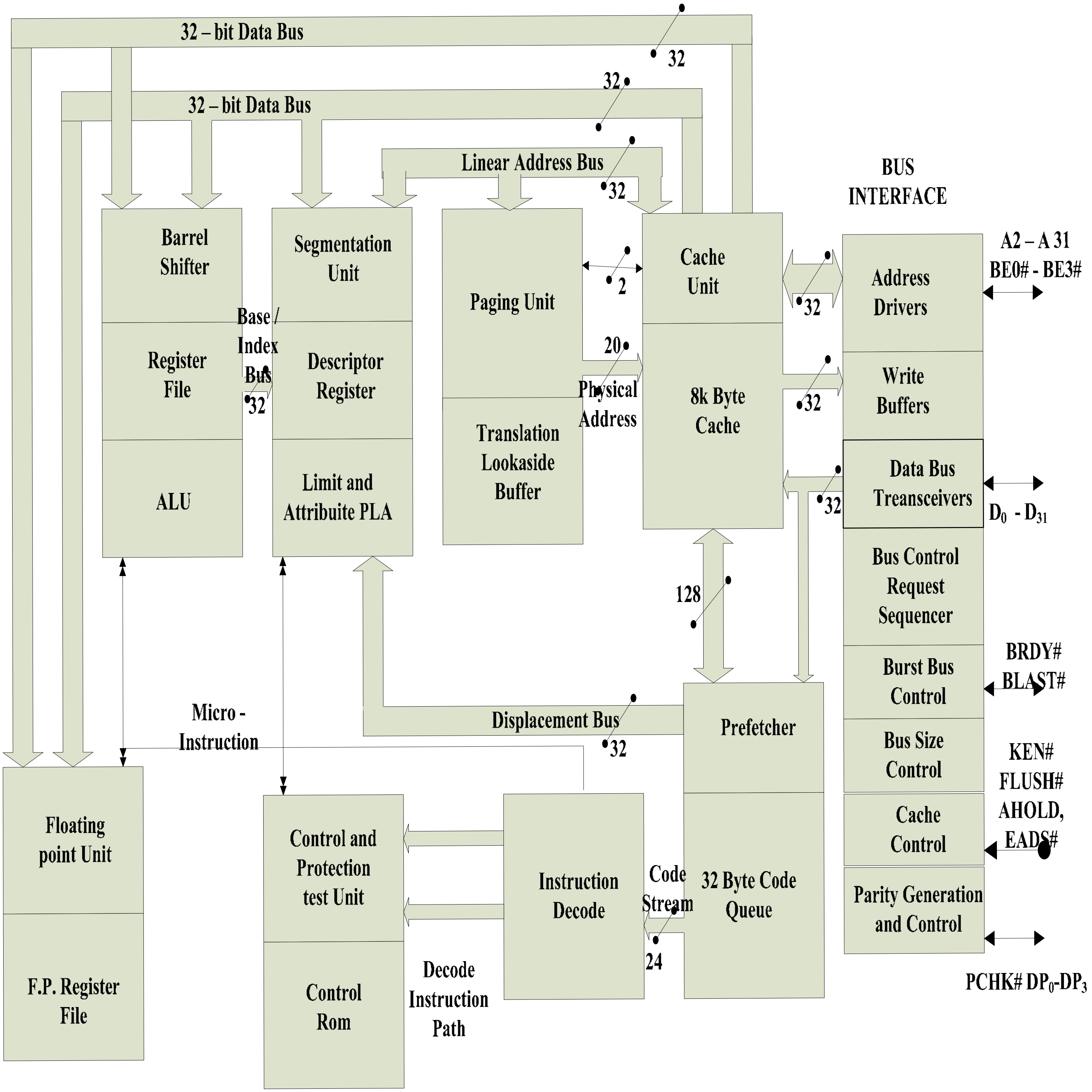
•**$\overline{A_{20}M_3}$** : The address bit 20 mask causes the 80486 to wrap its address around from location 000FFFFFH to 00000000H as in 8086. This provides a memory system that functions like the 1M byte real memory system in the 8086 processors.

•**$\overline{\text{ADS}}$** : The address data strobe become logic zero to indicate that the address bus contains a valid memory address.

•**AHOLD**: The address hold input causes the microprocessor to place its address bus connections at their high-impedance state, with the remainder of the buses staying active. It is often used by another bus master to gain access for a cache invalidation cycle.
BREQ: This bus request output indicates that the 486 has generated an internal bus request.

• **$\overline{BE_3}$-$\overline{BE_0}$** : Byte enable outputs select a bank of the memory system when information is transferred between the microprocessor and its memory and I/O.
The BE_3 signal enables D_{31} – D_{24} , BE_2 enables D_{23}-D_{16}, BE_1 enables D_{15} – D_8 and BE_0 enables D_7-D_0.

•**$\overline{\text{BLAST}}$**: The burst last output shows that the burst bus cycle is complete on the next activation of BRDY# signal.


32 – bit Data Bus
32
32
32 – bit Data Bus
Linear Address Bus
32
BUS
INTERFACE
Barrel
Shifter
Segmentation
Unit
Paging Unit
Cache
Unit
2
Address
Drivers
32
A2 – A 31
BE0# - BE3#
Base /
Index
Bus
32
Register
File
Descriptor
Register
20
Physical
Address
8k Byte
Cache
Write
Buffers
32
Translation
Lookaside
Buffer
ALU
Limit and
Attribute PLA
Data Bus
Treansceivers
32
D0 - D31
Bus Control
Request
Sequencer
128
BRDY#
BLAST#
Burst Bus
Control
Micro -
Instruction
Displacement Bus
32
Prefetcher
Bus Size
Control
KEN#
FLUSH#
AHOLD,
EADS#
Floating
point Unit
Control and
Protection
test Unit
Instruction
Decode
Code
Stream
32 Byte Code
Queue
Cache
Control
Parity Generation
and Control
24
PCHK# DP0-DP3
F.P. Register
File
Control
Rom
Decode
Instruction
Path

•$\overline{\mathbf{BOFF}}$: The Back-off input causes the microprocessor to place its buses at their high impedance state during the next cycle. The microprocessor remains in the bus hold state until the BOFF# pin is placed at a logic 1 level.

•**NMI** : The non-maskable interrupt input requests a type 2 interrupt.

•$\overline{\mathbf{BRDY}}$: The burst ready input is used to signal the microprocessor that a burst cycle is complete.

•$\overline{\mathbf{KEN}}$: The cache enable input causes the current bus to be stored in the internal.

•$\overline{\mathbf{LOCK}}$: The lock output becomes a logic 0 for any instruction that is prefixed with the lock prefix.

•$\mathbf{W\ /\ \overline{R}}$: current bus cycle is either a read or a write.

•$\overline{\mathbf{IGNNE}}$: The ignore numeric error input causes the coprocessor to ignore floating point error and to continue processing data. The signal does not affect the state of the FERR pin.

•$\overline{\mathbf{FLUSH}}$: The cache flush input forces the microprocessor to erase the contents of its 8K byte internal cache.

•$\overline{\mathbf{EADS}}$: The external address strobe input is used with AHOLD to signal that an external address is used to perform a cache invalidation cycle.

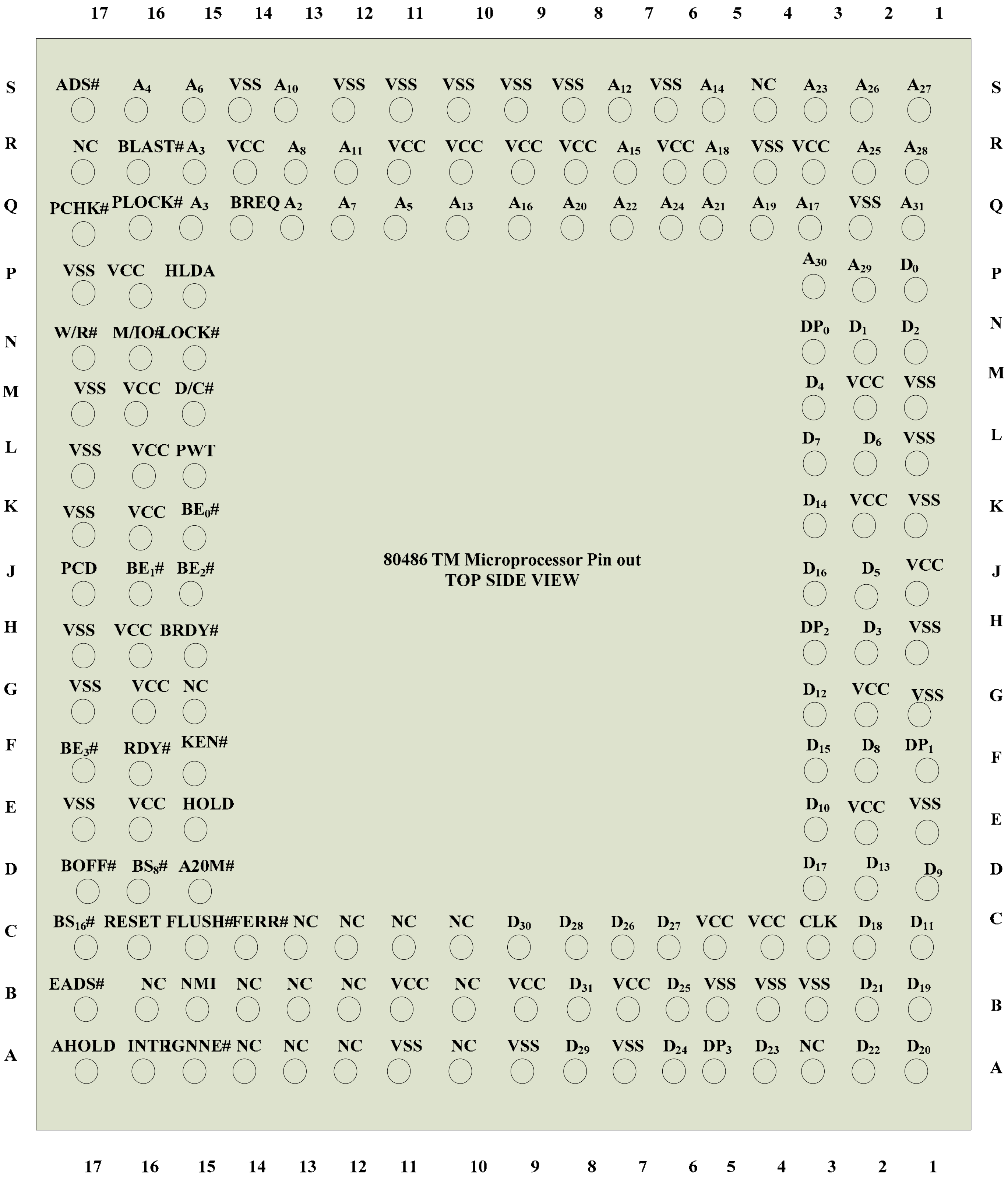
•$\overline{\mathbf{FERR}}$: The floating point error output indicates that the floating point coprocessor has detected an error condition. It is used to maintain compatibility with DOS software.

•$\overline{\mathbf{BS_8}}$: The bus size 8, input causes the 80486 to structure itself with an 8-bit data bus to access byte-wide memory and I/O components.

•$\overline{\mathbf{BS_{16}}}$: The bus size 16, input causes the 80486 to structure itself with an 16-bit data bus to access word-wide memory and I/O components.

•$\overline{\mathbf{PCHK}}$: The parity check output indicates that a parity error was detected during a read operation on the DP_3 – DP_0 pin.

•$\overline{\mathbf{PLOCK}}$: The pseudo-lock output indicates that current operation requires more than one bus cycle to perform. This signal becomes a logic 0 for arithmetic coprocessor operations that access 64 or 80 bit memory data.



80486 TM Microprocessor Pin out
TOP SIDE VIEW

	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	
S	ADS#	A_4	A_6	VSS	A_{10}	VSS	VSS	VSS	VSS	VSS	A_{12}	VSS	A_{14}	NC	A_{23}	A_{26}	A_{27}	S
R	NC	BLAST#	A_3	VCC	A_8	A_{11}	VCC	VCC	VCC	VCC	A_{15}	VCC	A_{18}	VSS	VCC	A_{25}	A_{28}	R
Q	PCHK#	PLOCK#	A_3	BREQ	A_2	A_7	A_5	A_{13}	A_{16}	A_{20}	A_{22}	A_{24}	A_{21}	A_{19}	A_{17}	VSS	A_{31}	Q
P	VSS	VCC	HLDA												A_{30}	A_{29}	D_0	P
N	W/R#	M/IO#	LOCK#												DP_0	D_1	D_2	N
M	VSS	VCC	D/C#												D_4	VCC	VSS	M
L	VSS	VCC	PWT												D_7	D_6	VSS	L
K	VSS	VCC	BE_0#												D_{14}	VCC	VSS	K
J	PCD	BE_1#	BE_2#												D_{16}	D_5	VCC	J
H	VSS	VCC	BRDY#												DP_2	D_3	VSS	H
G	VSS	VCC	NC												D_{12}	VCC	VSS	G
F	BE_3#	RDY#	KEN#												D_{15}	D_8	DP_1	F
E	VSS	VCC	HOLD												D_{10}	VCC	VSS	E
D	BOFF#	BS_8#	A20M#												D_{17}	D_{13}	D_9	D
C	BS_{16}#	RESET	FLUSH#	FERR#	NC	NC	NC	NC	D_{30}	D_{28}	D_{26}	D_{27}	VCC	VCC	CLK	D_{18}	D_{11}	C
B	EADS#	NC	NMI	NC	NC	NC	VCC	NC	VCC	D_{31}	VCC	D_{25}	VSS	VSS	VSS	D_{21}	D_{19}	B
A	AHOLD	INTR	IGNNE#	NC	NC	NC	VSS	NC	VSS	D_{29}	VSS	D_{24}	DP_3	D_{23}	NC	D_{22}	D_{20}	A
	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	

•**PWT**: The page write through output indicates the state of the PWT attribute bit in the page table entry or the page directory entry.

•**$\overline{\text{RDY}}$** : The ready input indicates that a non-burst bus cycle is complete. The RDY signal must be returned or the microprocessor places wait states into its timing until RDY is asserted.

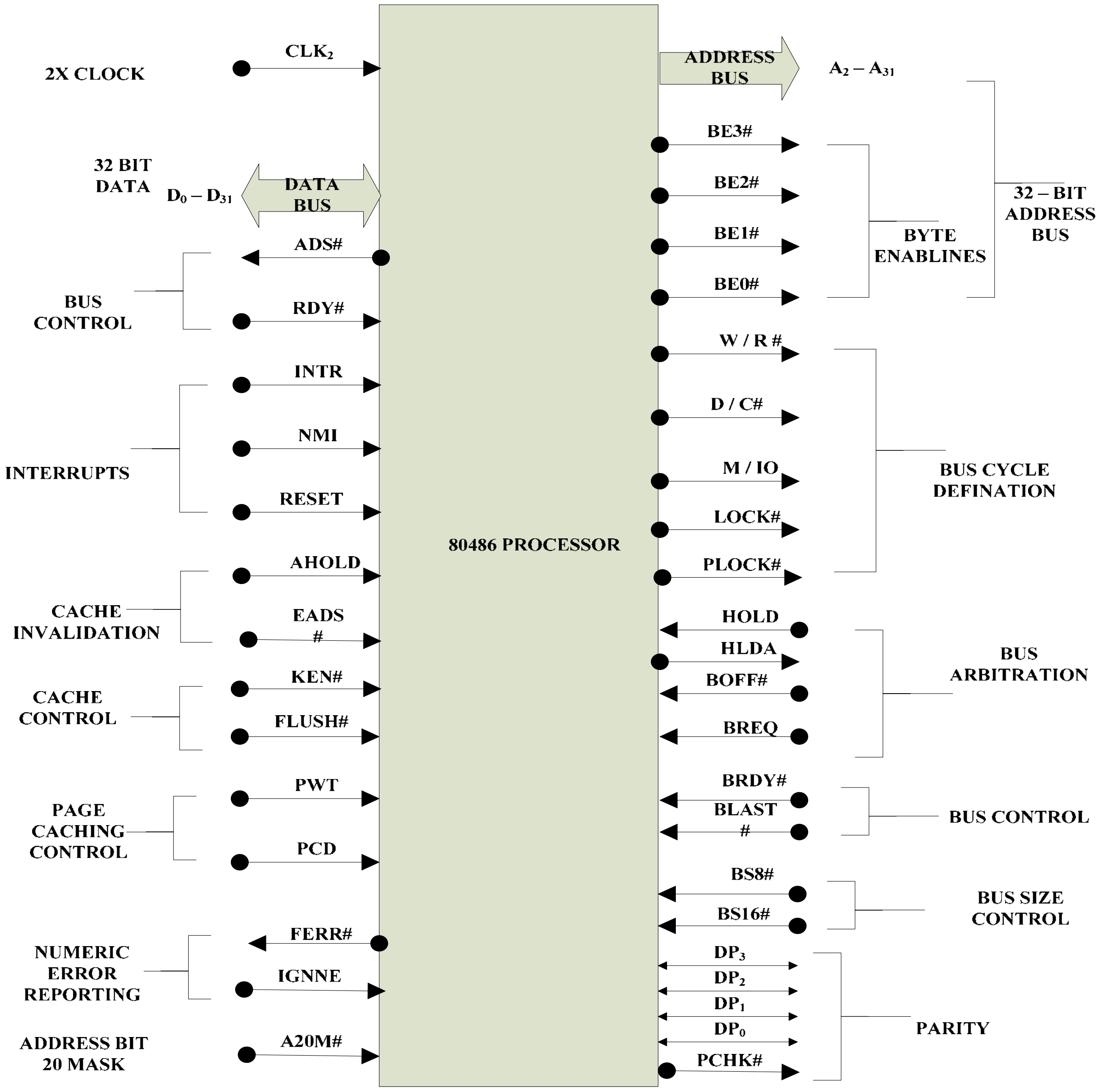
•**$\overline{\text{M}}$ / IO** : Memory / $\overline{\text{IO}}$ defines whether the address bus contains a memory address or an I/O port number. It is also combined with the W/ R signal to generate memory and I/O read and write control signals.

80486 Signal Group

•The 80486 data bus, address bus, byte enable, ADS#, RDY#, INTR, RESET, NMI, M/IO#, D/C#, W/R#, LOCK#, HOLD, HLDA and BS$_{16}$# signals function as we described for 80386.
•The 80486 requires 1 clock instead of 2 clock required by 80386.
•A new signal group on the 486 is the PARITY group DP$_0$-DP$_3$ and PCHK#.
•These signals allow the 80486 to implement parity detection / generation for memory reads and memory writes.
•During a memory write operation, the 80486 generates an even parity bit for each byte and outputs these bits on the DP$_0$-DP$_3$ lines.

•These bits will store in a separate parity memory bank.
•During a read operation the stored parity bits will be read from the parity memory and applied to the DP$_0$-DP$_3$ pins.
•The 80486 checks the parities of the data bytes read and compares them with the DP$_0$-DP$_3$ signals. If a parity error is found, the 80486 asserts the PCHK# signal.
•Another new signals group consists of the BURST ready signal BRDY# and BURST last signal BLAST#.
•These signals are used to control burst-mode memory reads and writes.
•A normal 80486 memory read operation to read a line into the cache requires 2 clock cycles. However, if a series of reads is being done from successive memory locations, the reads can be done in burst mode with only 1 clock cycle per read.
•To start the process the 80486 sends out the first address and asserts the BLAST# signal high. When the external DRAM controller has the first data bus, it asserts the BRDY# signal.
•The 80486 reads the data word and outputs the next address. Since the data words are at successive addresses, only the lower address bits need to be changed. If the DRAM controller is operating in the page or the static column modes then it will only have to output a new column address to the DRAM.

•In this mode the DRAM will be able to output the new data word within 1 clock cycle.



•When the processor has read the required number of data words, it asserts the BLAST# signal low to terminate the burst mode.

•The final signal we want to discuss here are the bus request output signal BREQ, the back-off input signal BOFF#, the HOLD signal and the hold-acknowledge signal HLDA.

•These signals are used to control sharing the local 486 bus by multiple processors (bus master).

•When a master on the bus need to use the bus, it asserts its BERQ signal .

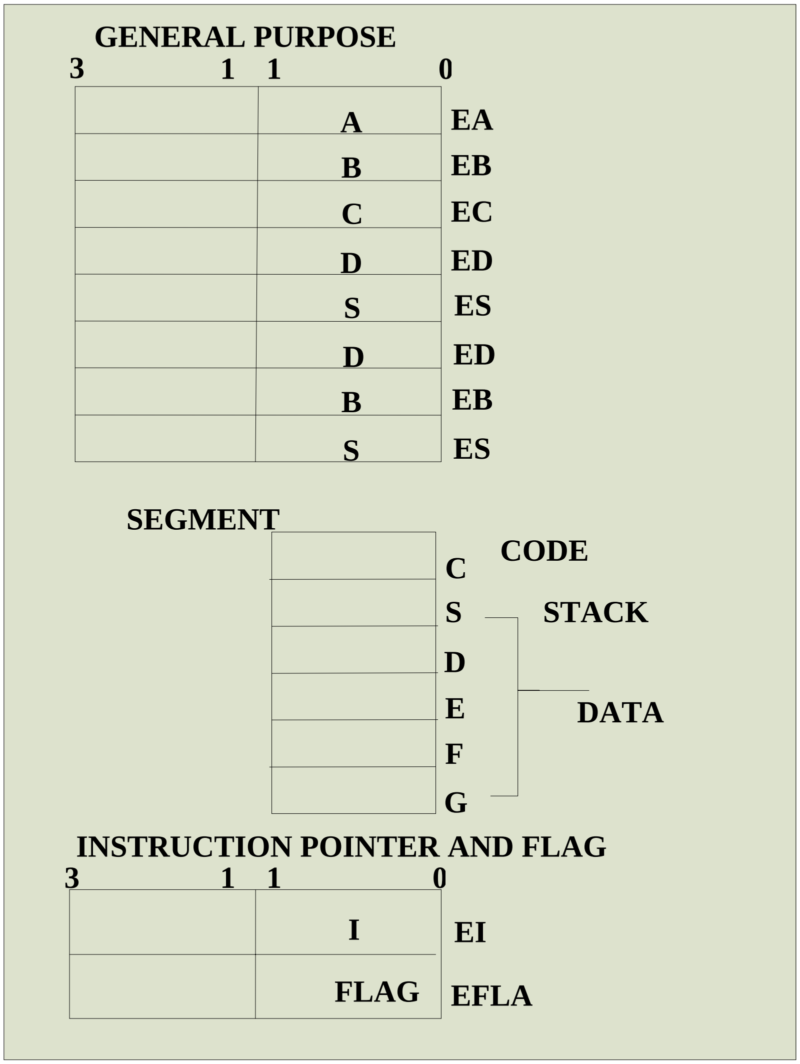
•An external parity circuit will evaluate requests to use the bus and grant bus use to the highest – priority master. To ask the 486 to release the bus , the bus controller asserts the 486 HOLD input or BOFF# input.

•If the HOLD input is asserted, the 486 will finish the current bus cycle, float its buses and assert the HLDA signal.

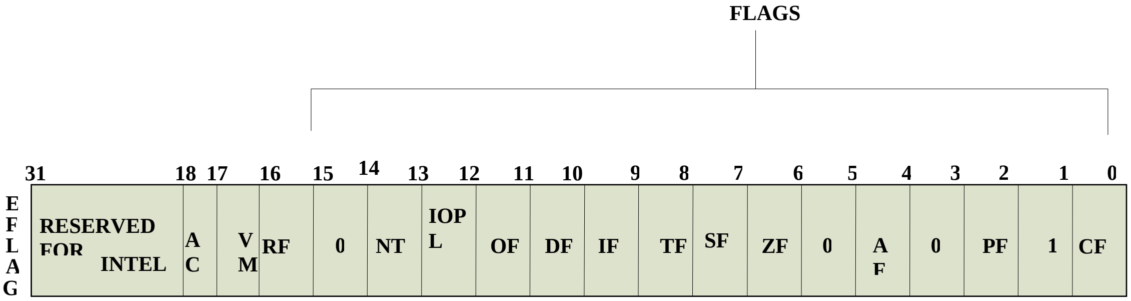
•To prevent another master from taking over the bus during a critical operation, the 486 can assert its LOCK# or PLOCK# signal.

EFLAG Register of The 80486

•The extended flag register EFLAG is illustrated in the figure. The only new flag bit is the AC alignment check, used to indicate that the microprocessor has accessed a word at an odd address or a double word boundary.
•Efficient software and execution require that data be stored at word or doubleword boundaries.


GENERAL PURPOSE
3
1
1
0
A
EA
B
EB
C
EC
D
ED
S
ES
D
ED
B
EB
S
ES
SEGMENT
C
CODE
S
STACK
D
E
DATA
F
G
INSTRUCTION POINTER AND FLAG
3
1
1
0
I
EI
FLAG
EFLA

Flag Register of 80486



	31	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EFLAG	RESERVED FOR INTEL		AC	VM	RF	0	NT	IOPL	OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

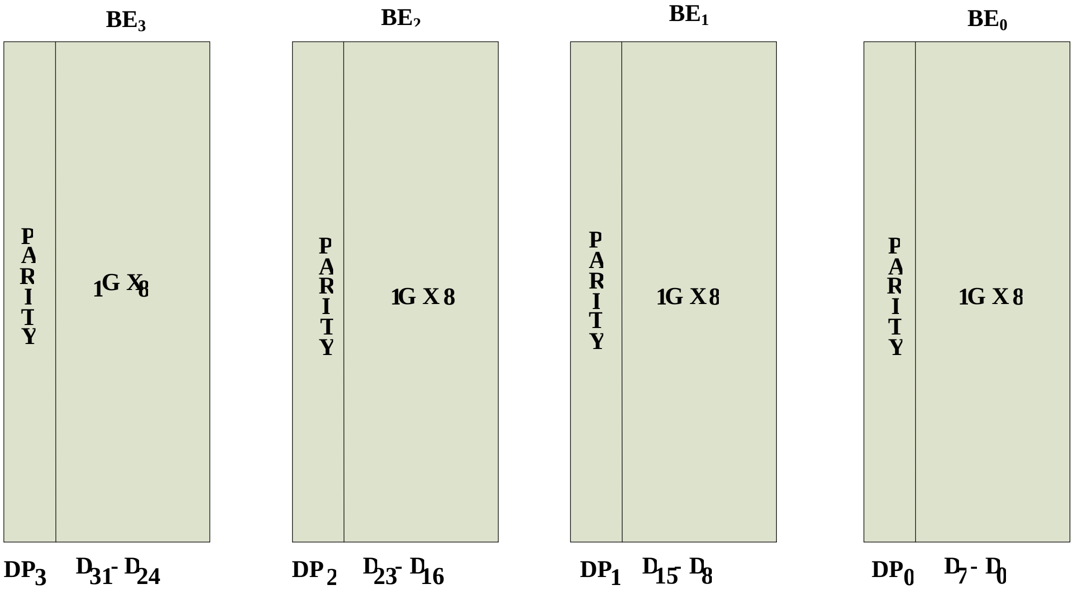
CF: Carry Flag
AF: Auxiliary carry
ZF: Zero Flag
SF : Sign Flag
TF : Trap Flag
IE : Interrupt Enable
AC : Alignment Check
DF : Direct Flag

OF : Over Flow
IOPL : I/O Privilege Level
NT : Nested Task Flag
RF : Resume Flag
VM : Virtual Mode

80486 Memory System

•The memory system for the 486 is identical to 386 microprocessor. The 486 contains 4G bytes of memory beginning at location 00000000H and ending at FFFFFFFFH.
•The major change to the memory system is internal to 486 in the form of 8K byte cache memory, which speeds the execution of instructions and the acquisition of data.
•Another addition is the parity checker/ generator built into the 80486 microprocessor.
•***Parity Checker / Generator*** : Parity is often used to determine if data are correctly read from a memory location. INTEL has incorporated an internal parity generator / decoder.

•Parity is generated by the 80486 during each write cycle. Parity is generated as even parity and a parity bit is provided for each byte of memory. The parity check bits appear on pins DP0-DP3, which are also parity inputs as well as parity outputs.
•These are typically stored in memory during each write cycle and read from memory during each read cycle.
•On a read, the microprocessor checks parity and generates a parity check error, if it occurs on the PCHK# pin. A parity error causes no change in processing unless the user applies the PCHK signal to an interrupt input.
•Interrupts are often used to signal a parity error in DS-based computer systems. This is same as 80386, except the parity bit storage.
•If parity is not used, Intel recommends that the DP0 – DP3 pins be pulled up to +5v.



•***CACHE MEMORY****:* The cache memory system stores data used by a program and also the instructions of the program. The cache is organised as a 4 way set associative cache with each location containing 16 bytes or 4 doublewords of data.
•Control register CR0 is used to control the cache with two new control bits not present in the 80386 microprocessor.
•The CD (cache disable) , NW (non-cache write through) bits are new to the 80486 and are used to control the 8K byte cache.
•If the CD bit is a logic 1, all cache operations are inhibited. This setting is only used for debugging software and normally remains cleared. The NW bit is used to inhibit cache write-through operation. As with CD, cache write through is inhibited only for testing. For normal operations CD = 0 and NW = 0.
•Because the cache is new to 80486 microprocessor and the cache is filled using burst cycle not present on the 386.

80486 Memory Management

•The 80486 contains the same memory-management system as the 80386. This includes a paging unit to allow any 4K byte block of physical memory to be assigned to any 4K byte block of linear memory. The only difference between 80386 and 80486 memory-management system is paging.
•The 80486 paging system can disabled caching for section of translation memory pages, while the 80386 could not.

31 12	11 10 9	8	7	6	5	4	3	2	1	0
PAGE TABLE OR PAGE FRAME	OS BITS	O	O	D	A	P C D	P W T	U S	R W	P

Page Directory or Page Table Entry For The 80486 Microprocessor

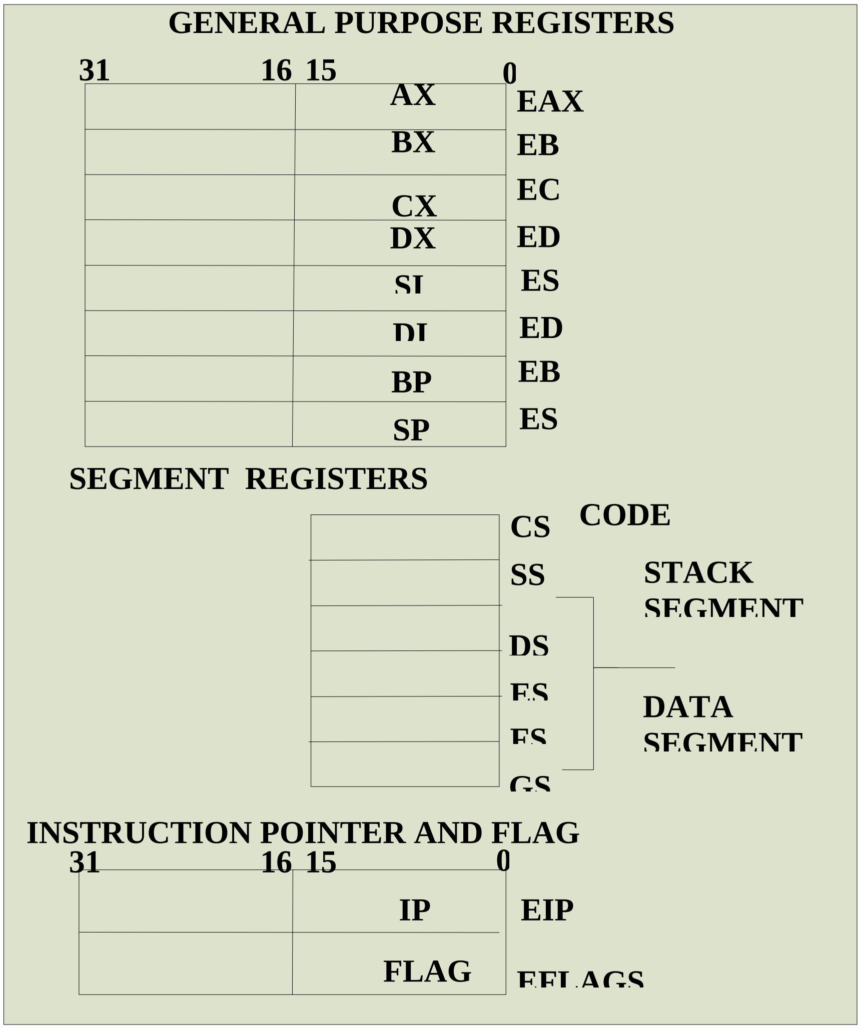
•If these are compared with 80386 entries, the addition of two new control bits is observed (PWT and PCD).
•The page write through and page cache disable bits control caching.
•The PWT controls how the cache functions for a write operation of the external cache memory. It does not control writing to the internal cache. The logic level of this bit is found on the PWT pin of the 80486 microprocessor. Externally, it can be used to dictate the write through policy of the external caching.
•The PCD bit controls the on-chip cache. If the PCD = 0, the on-chip cache is enabled for the current page of memory.
•Note that 80386 page table entries place a logic 0 in the PCD bit position, enabling caching. If PCD = 1, the on-chip cache is disable. Caching is disable regard less of condition of KEN#, CD, and NW.

Cache Test Registers

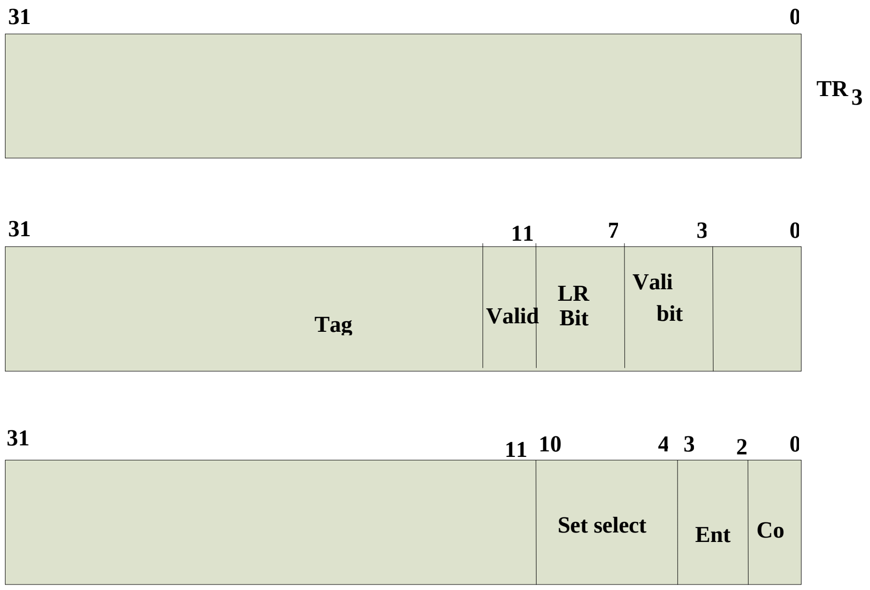
•The 80486 cache test registers are TR3, TR4, TR5.
•Cache data register (TR3) is used to access either the cache fill buffer for a write test operation or the cache read buffer for a cache read test operation.
•In order to fill or read a cache line (128 bits wide), TR3 must be written or read four times.

•The contents of the set select field in TR5 determine which internal cache line is written or read through TR3. The 7 bit test field selects one of the 128 different 16 byte wide cache lines. The entry select bits of TR5 select an entry in the set or the 32 bit location in the read buffer.

•The control bits in TR5 enable the fill buffer or read buffer operation (00)
•Perform a cache write (01), Perform a cache read (10)
•Flush the cache (11).
•The cache status register (TR4) hold the cache tag, LRU bits and a valid bit. This register is loaded with the tag and valid bit before a cache a cache write operation and contains the tag, valid bit, LRU bits, and 4 valid bits on a cache test read.
•Cache is tested each time that the microprocessor is reset if the AHOLD pin is high for 2 clocks prior to the RESET pin going low. This causes the 486 to completely test itself with a built in self test or BIST.



- •The BIST uses TR3, TR4, TR5 to completely test the internal cache. Its outcome is reported in register EAX. If EAX is a zero, the microprocessor, the coprocessor and cache have passed the self test.
- •The value of EAX can be tested after reset to determine if an error is detected. In most of the cases we do not directly access the test register unless we wish to perform our own tests on the cache or TLB.



Cache test register of the 80486 microprocessor

M1 ARCHITECTURE OF MICROPROCESSORS

a. General definitions of mini computers etc,.

Q1. What are the advantages and the limitations microcontroller over a microprocessor?

Q2. Describe the main blocks in a digital signal processor that are not in a general microprocessor

b. Overview of 8085 Microprocessor

Q1. List the internal registers in 8085 microprocessor and their abbreviations and lengths. Describe the primary function of each register.

Q2. Differentiate between NMI and MI interrupts

Q3. Explain how with external hardware TRAP can be masked

Q4. Interface a Speaker to SOD pin of 8085 Microprocessor.

Q5. Explain DMA function in 8085 microprocessor with timing diagrams

Q6. Explain the timing diagrams of 8085 when it is executing Memory mapped I/O and I/O mapped I/O instructions

c. Overview of 8086 Microprocessor

Q1. List all the registers associated with the four segment registers

Q2. List the internal registers in 8086 Microprocessor

Q3. What are the main blocks in BIU and EU

Q4. Explain the coordination between BIU an EU

d. Signals and pins of 8086 microprocessor

Q1. How do you configure 8086 into minimum and maximum modes

Q2. Bring out the differences between 8086 and 8088 processors

Q3. Explain all the features in 8284

Q4. Why and when wait states are required. How do you insert wait states

M2. Assembly language of 8086

a. Description of Instructions

Q1. If BH = 0F3H what is the value of BH in hex after the instruction SAR BH, 1

Q2. IF AL = 78H and BL=73H explain how DAS instruction (after subtracting BL from AL) adjusts to the BCD result

Q3. If CL=78H what is the value of CL after the instruction ROL CL, 3

Q4. Why AAD is to be executed before DIV instruction while converting unpacked BCD to Binary number

Q4. Under what conditions REPE MOVS executes

Q5. Explain XLAT instruction to linearize transducer characteristics

Q6. Explain intra segment and inter segment branch instructions with examples the instructions related to arithmetic and logical shift.

Q7. Explain all addressing modes with the assembler syntax and how effective address is calculated

b. Assembly directives.

Q1. Explain EQU directive with example

Q2. Explain SEGMENT directives with examples

Q3. Explain coding template for 8086 instruction

c. Algorithms with assembly software programs

Q1. Write an algorithm to compute Fibonacci numbers using a recursive procedure. Write 8086 assembly program for the above

Q2. Write an algorithm and assembly program to convert an unpacked 4 digit number to Binary number.

Q3. Write an algorithm and assembly program to convert a 16 bit number to a maximum of 5 unpacked digits

Q4. Write an algorithm and assembly program to convert an unpacked 4 digit number to Binary number.

Q5. Write an algorithm and assembly program to find the square root of a 16 bit number using shift and subtract method.

Q6. Write an algorithm and assembly program to reverse the bits in a 16 bit number and check whether it is a palindrome.

Q7. Write an algorithm and assembly program for a cash bill of n materials. Rupees is a 4 digit and paisa is a 2 digit number which are stored in two different arrays. Find the total amount for the n materials. Subtract 10% discount on the total and give the actual amount to be paid. Hint Shift the total amount by one digit to get the 10% discount and get the actual amount.

M3. Interfacing with 8086

a Interfacing with RAM's and ROM's

Q1. What are the differences in interfacing RWMs while 8086 is in minimum and maximum modes

Q2. Sketch and explain the interface of 32K x 16 RWMs using a decoder in minimum mode. What is the maximum access time of ROMs such that it does not require wait states when 8086 operates at 8 MHz

Q3. Sketch and explain the timing diagrams in the above interface Question 2

Q4. Sketch and explain the 8086 bus activities during write machine cycle

b Interfacing with peripheral IC's like 8255 etc,.

Q1. What are the steps in interfacing peripherals with the micro processor

Q2. Sketch and explain the interface of PPI 8255 to the 8086 microprocessor in minimum mode. Interface 4 7 segment LEDs to display as a BCD counter

Q3. In the above question Q2 interface two keys UP and DOWN to the PPI. Write an 8086 assembly program segment such that when UP is pressed the counter counts up every second. Similarly when DOWN key is pressed the counter decrements every second

Q4. Sketch and explain the interface of 8279 to the 8086 microprocessor in minimum mode. Interface 8x8 key pad and 16x 7 Seg LED display. Write an 8086 assembly program to read the key codes of keys and display **-NPTEL-INDIA**

Q5. Sketch and explain the interface of PIT 8254 to the 8086 microprocessor in minimum mode. Cascade two counters in the PIT. Write a program segment two get one minute delay

M4. Coprocessor 8087

a. Architecture of 8087

Q1. Give five differences between the main processor and the coprocessor

Q2. How does 8086 distinguish its instructions from 8087 instructions as it fetches from memory

Q3. What is the role of busy pin 8087, when it is interfaced with 8086

Q4. In which mode 8086 is interfaced with 8087 and why

Q5. Explain major blocks of 8087

b. Data types, instructions and programming

Q1. What are the minimum and maximum values can be represented in all types of data

Q2. What are the different steps involved in converting a short real number to a decimal number

Q3. What are the different steps involved in converting a decimal number to a long real number

Q4. Write an 8086/87 assembly program to compute the total surface area of a sphere. The formula is 4*PI* R^2 where R is a real number

Q5. What are the differences between rounding and truncation. Explain with examples

M5. Architecture of Microcontrollers

a Overview of 8051 micro controller architecture

Q1. What are the advantages and disadvantages of using Harvard architecture in 8051

Q2. How much maximum external program memory can be interfaced

Q3. Explain PSW SFR. Give the application differences between Carry and Overflow flags

Q4. What are the power consumptions in power down and idle modes

Q5. Explain Quasi Bidirectional ports of 8051

Q6. What is the status of all registers on reset

Q7. What is the maximum delay the Timer0 produces when 8051 is operated at 12MHz

Q8. Explain how in Serial communication mode 0 expands I/O lines with the help of shift

b Overview of 8096 micro controller architecture

Q1. How many bytes are there in the internal memory

Q2. Explain all the bits in the PSW

Q3. How much program memory is in the chip and how much more can be interfaced externally

Q4. List and explain all SFRs

Q5. How DAC is realized using PWM output of 8096

Q6. What is the maximum delay the Timer0 produces when 8096 is operated at 12MHz

M6. Assembly language of 8051

a. Description of instructions

Q1. On what condition JZ quit become true

Q2. What are the values of RS0 and RS1 of PSW when 19H location is treated as a register

Q3. Explain JBC bit, exit instruction

Q4. Does DA A instruction converts binary number to BCD number? Explain under what conditions the BCD numbers gets adjusted after BCD addition

Q5. Differentiate between RET and RETI instructions

Q6. What are program branch ranges of SJMP, AJMP and LJMP instructions

b. Assembly Directives

Q1. List all the header files required to cross compile C program

Q2. Using the directives initialize a Look Up Table

c. Algorithms with 8051 assembly language programs

Q1. Write an 8051 assembly program to check a byte is a palindrome. Palindrome is a byte or a word or words when read left or right it will be the same. Like for e.g. C3H (11000011b) or MALAYALAM.

Q2. Write an 8051 assembly program to produce a software delay of 1 minute.

Q3. Write an 8051 assembly program to multiply two 16 bit numbers, using shift left and add algorithm

Q4. Write an 8051 assembly program to compute the square root of a 16 bit number using shift left and subtract method

Q5. Write an 8051 assembly program to find LCM of two 16 bit numbers.

Q6. Write an 8051 assembly program to search a key in a array of 16 bit numbers using Binary search algorithm

M7. Interfacing with 8051

a. Interfacing with peripherals like keyboards, LEDs, 7 segment LEDs, LCDs, ADCs, etc,.

Q1. Sketch and explain the interface of 5x4 key matrix using 74923.

Write an 8051 assembly program segment to input the code of keys.

Q2. Sketch and explain the interface of 4X7Segment LEDs in multiplexed mode.

Write an 8051 assembly program segment to flash CEDT at 5Hz

Q3. Sketch the interface of a 16ch x 1line LCD to the 8051 microcontroller. Write an 8051 assembly program segment to display any Logo

Q4. Sketch the interface of a serial ADC MAX 192 to the 8051 microcontroller. Write an 8051 assembly program segment to read an analog signal through the ADC

Q5. Sketch the interface of a dual DAC 7303 to the 8051 microcontroller. Output the DAC outputs are connected to an analog comparator. A LED is connected to the output of the comparator for indication. Write an 8051 assembly program segment to flash if one analog input is greater than the other.

Q6. Sketch the interface of a RTC 1302 to the 8051 microcontroller. Write an 8051 assembly program segment to read and write real time into the RTC.

M8. High end processors

a. Introduction to 80386 and 80486

Q1. List all the additional features that the 80386 microprocessor has over 8086

Q2. What is the main difference between the 80386 DX and 80386 SX microprocessor

Q3. How much the physical memory can 80386 address in real mode and in protected mode?

Q4. How are the tasks in 80386 system protected from each other.

Q5. How is an 80386 switched into virtual 8086 mode during task switch.

Q6. Describe three major additions or improvements that the 80486 processor has over 80386 processor.

M1 ARCHITECTURE OF MICROPROCESSORS

a. General definitions of mini computers etc,.

Q1. Differentiate between a microprocessor and a micro controller
Q2. Differentiate between a microprocessor and digital signal processor

b. Overview of 8085 Microprocessor

Q1. List the internal registers in 8085 microprocessor and their abbreviations and lengths. Describe the primary function of each register.
Q2. List five levels of interrupts in 8085 microprocessor with priority.
Q3. Interface a key to SID pin of 8085 Microprocessor.
Q4. Interface a LED to SOD pin of 8085 Microprocessor.
Q5. In 8085 microprocessor which has higher the priority NMI or DMA
Q6. What are the differences between Memory mapped I/O and I/O mapped I/O

c. Overview of 8086 Microprocessor

Q1. Explain the need of segmentation
Q2. List the internal registers in 8086 Microprocessor
Q3. Explain the roles of BIU and EU
Q4. What are the advantages of pipelining
Q5. Explain all the flags in 8086

d. Signals and pins of 8086 microprocessor

Q1. List the signals in minimum and maximum modes
Q2. Explain the roles of pins $\overline{\text{TEST}}$, $\overline{\text{LOCK}}$
Q3. Which are the pins of 8086 that are to be connected to interface 8284 and explain their functions
Q4. Which are the pins of 8086 that are to be connected to 8087 and explain their functions

M2. Assembly language of 8086

a. Description of Instructions

Q1. If AL = -9 and BL = 47_{10} after IDIV BL what are the values of AL and AH.
Q2. IF AX = -200_{10} and CX = 670H after IMUL CX what are the values of AX and DX
Q3. Explain AAA, AAD, AAM, AAS instructions with examples.
Q4. Explain DAA, DAS instructions with examples.
Q5. Explain the instructions related to the fixed and variable ports.
Q6. Explain the instructions related to arithmetic and logical shift.
Q7. How REP instruction is used along with string instructions.

Q8. Explain different types of CALL instructions

b. Assembly directives.

Q1. What is the length of bytes reserved for the following directive STORE DW 100 DUP(0)

Q2. What is the difference between ENDS and ENDP directives.

Q3. Explain PTR directive

c. Algorithms with assembly software programs

Q1. Write an algorithm to convert BCD to Binary numbers. Write 8086 assembly program to convert two digit BCD number to hexadecimal number

Q2. Write an algorithm to convert Binary number to BCD number. Write 8086 assembly program to convert one byte Binary number to BCD.

Q3. Write an algorithm to evaluate a factorial of an integer number N. Write an assembly program using recursive procedure.

Q4. Write an algorithm to find GCD of two numbers. Write an assembly program to find GCD of two words.

Q5. Write an algorithm and assembly program to sort the numbers in an array in descending order using bubble sort method

M3. Interfacing with 8086

a Interfacing with RAM’s and ROM’s

Q1. Sketch and explain the interface of 32K x 16 ROMs using a decoder in minimum mode. What is the maximum access time of ROMs such that it does not require wait states when 8086 operates at 8 MHz

Q2. Sketch and explain the interface of 8K x 16 RAMs using a decoder in minimum mode. What is the maximum access time of RAMs such that it does not require wait states when 8086 operates at 8 MHz

Q3. Sketch and explain the 8086 bus activities during read machine cycle

b Interfacing with peripheral IC’s like 8255 etc,.

Q1. Sketch and explain the interface of PPI 8255 to the 8086 microprocessor in minimum mode. Interface 8 LEDs to the port B of 8255. Interface 8 keys to the port A. Write an 8086 assembly program to read the key status and output on to the 8 LEDs

i) Interface an 8 bit ADC 808 to port A. Derive control signals from port C. Write an 8086 assembly program segment to read an analog signal.

ii) Interface an 8 bit DAC 08 to port A. Write an 8086 assembly program segment to output a ramp.

iii) Interface 16 ch x 1Line LCD to port A. Derive control signals from port C. Write an 8086 assembly program segment to flash **WELCOME TO CEDT**

Q2. Sketch and explain the interface of PIT 8254 to the 8086 microprocessor in minimum mode. Write an 8086 assembly program to generate a clock of 10 Hz on the OUT 0 pin. Write an 8086 assembly program to generate a hardware triggerable mono-shot of 1 msec pulse width.

Q3. Sketch and explain the interface of 8279 to the 8086 microprocessor in minimum mode. Interface 8x8 key pad and 16x 7 Seg LED display. Write an 8086 assembly program to read the key codes of keys and display **-IISc-BANGALORE-**

Q4. Sketch and explain the interface of PIC 8259 to the 8086 microprocessor in minimum mode. Show the cascading of additional eight 8259s to provide 64 external interrupts. Write an 8086 assembly program to initialize master 8259 and slaves.

M4. Coprocessor 8087

a. Architecture of 8087

Q1. Explain the stack in 8087

Q2. How 8086 and 8087 instructions stored in programming are executed simultaneously in 8086/8087 system

Q3. Explain precision and rounding controls

Q4. What are the exception flags in status register

Q5. Explain how 8087 is interfaced to 8086

b. Data types, instructions and programming

Q1. List all data types supported by 8087 and their ranges

Q2. Convert 278.375 to short real

Q3. Convert 4332A000H to real number

Q4. List all the mathematical instructions in 8087

Q5. Write an 8086/87 assembly program to compute X^{Y} where X and Y are real numbers

Q6. Write an 8086/87 assembly program to find the hypotenuse of a right angled triangle with two sides given.

M5. Architecture of Microcontrollers

a Overview of 8051 micro controller architecture

Q1. How Harvard architecture is implemented in 8051

Q2. List the features of 8051

Q3. What are the alternate functions of Port 3, port 2, and Port 0

Q4. List all SRF's

Q5. How do you differentiate between SFR bits and internal memory bits

Q6. What are the advantages of register banks
Q7. What is the maximum program memory and data memory can be interfaced externally
Q8. List all interrupt vectors.

b Overview of 8096 micro controller architecture

Q1. What is RALU architecture in 8096
Q2. List all the features of 8096
Q3. What is reset location of 8096
Q4. Explain HSI and HSO in 8096
Q5. How DAC is realized using PWM output of 8096
Q6. Explain three operands instruction in 8096
Q7. Explain watch dog timer in 8096
Q8. Explain ADC in 8096
Q9. What is the maximum baud rate possible in 8096
Q10. Bring out the differences between T1 and T2 timers in 8096.

M6. Assembly language of 8051

a. Description of instructions

Q1. Explain MUL AB and DIV AB instructions
Q2. Explain two indirect indexed instructions
Q3. Explain all flags in 8051
Q4. Explain all Boolean instructions
Q5. Explain all compare instructions
Q6. Write a program to decrement the value of DPTR

b. Assembly Directives

Q1. Explain 8051 assembler directives EQU, SET and BIT with one example each.
Q2. Explain 8051 assembler directives DBIT and DS with one example each.

c. Algorithms with 8051 assembly language programs

Q1. Write an 8051 assembly program to exchange the data using PUSH and POP instructions.
Q2. Write an 8051 assembly program to convert two digit BCD number to hexadecimal number
Q3. Write an 8051 assembly program to convert one byte Binary number to BCD.
Q4. Write an 8051 assembly program to evaluate the factorial of an integer number N using recursive procedure.
Q5. Write an 8051 assembly program to find GCD of two numbers.

Q6. Write an 8051 assembly program to sort the number in an array using bubble sort method

M7. Interfacing with 8051

a. Interfacing with peripherals like keyboards, LEDs, 7 segment LEDs, LCDs, ADCs, etc,.

Q1. Sketch and explain the interface of 8x4 key matrix using 8:1 multiplexer, and a 3:8 decoder to 8051 microcontroller. Write an 8051 assembly program segment to input the code of keys.
Q2. Switch the interface of a single red LED to 8051 microcontroller. Write an 8051 assembly program segment to blink the LED at 5Hz.
Q3. Sketch the interface of a 4x7 segment LEDs to the 8051 microcontroller. Write an 8051 assembly program segment to display **I.I.Sc**
Q4. Sketch the interface of a 16ch x 1line LCD to the 8051 microcontroller. Write an 8051 assembly program segment to display **NPTEL**
Q5. Sketch the interface of an ADC 808 to the 8051 microcontroller. Write an 8051 assembly program segment to read an analog signal through the ADC
Q6. Sketch the interface of a DAC 08 to the 8051 microcontroller. Write an 8051 assembly program segment to output a ramp signal through the DAC.
Q7. Sketch the interface of a RTC 1307 to the 8051 microcontroller. Write an 8051 assembly program segment to read and write real time into the RTC.

M8. High end processors

a. Introduction to 80386 and 80486

Q1. List all the additional features that the 80386 microprocessor has over 8086
Q2. What is the main difference between the 80386 DX and 80386 SX microprocessor
Q3. How much the physical memory can 80386 address in real mode and in protected mode.
Q4. How are the tasks in 80386 system protected from each other.
Q5. How is an 80386 switched into virtual 8086 mode during task switch.
Q6. Describe three major additions or improvements that the 80486 processor has over 80386 processor.