University of Global Village (UGV), Barishal

Department of Electrical And Electronics Engineering

EEE 0714-3106 Microprocessor & Interfacing

Course Content

Edited By :
Abdul Hai Utsho
Lecturer
Dept of EEE, UGV

Basic Course Information:

- **Course Title :** Electronics – 1
- **Course Code:** EEE 0714 – 3106
- **CIE Marks:** 90
- **SEE Marks:** 60
- **Exam Hours:** 2 hours (Mid term Examination)

  3 hours (Semester End Examination)
- **Semester:** 3rd Semester
- **Academic Session:** Winter 2025

Assessment Pattern

CIE – Continuous Internal Evaluation (90 Marks)

Bloom's Category Marks (out of 90)	Mid Term (45)	Assignment (15)	Quiz (15)	Attendance & External
Remember	05		05	
Understand	05	05	05	
Apply	10		05	15
Analyze	10			
Evaluate	10			
Create	05	05		

Assessment Pattern

SEE – Semester End Examination (60 Marks)

Bloom's Category Marks (out of 90)	Final Examination Term (60)
Remember	15
Understand	10
Apply	10
Analyze	10
Evaluate	10
Create	05

Course Learning Outcome (CLO's):

CLO1	Explain the architecture, instruction set, memory and input/output interface for 8086/8088 microprocessor
CLO2	Relate microprocessor working principle, instruction set execution and peripheral connection for specific applications
CLO3	Program in assembly language for executing microprocessor instructions set
CLO4	Investigate microprocessor-based systems by designing and conducting experiments
CLO5	Design a microprocessor-based system that meets specified requirements

Course Rationales:

The course familiarizes the architecture of the 8086 processor, assembling language programming and interfacing with various modules. The student can also understand 8051 Microcontroller concepts, architecture, programming, and application of Microcontrollers. Student able to do any type of VLSI, embedded systems, industrial and real time applications by knowing the concepts of Microprocessor and Microcontrollers.

Course Objectives:

- To develop an in-depth understanding of the operation of microprocessors.
- To master assembly language programming using concepts like assembler directives, procedures, macros, software interrupts etc.
- To create an exposure to basic peripherals, its programming and interfacing techniques.
- To understand the concept of Interrupts and interfacing details of 8086.
- To impart the basic concepts of serial communication in 8086.

Course Contents Summary

Serial No	Course Contents	Hours
1	Microprocessor Introduction, Microprocessor based System, Assembly Language, memory basic, 8085 basic, Block diagram, registers, Demultiplexing, Flag Registers, Flags	10
2	8086, Pin diagrams, Block Diagram, Pin Working Principle, addressing modes, instruction sets, basic programs	12
3	***Peripherals:*** Basic of 8255, 8255 block diagram, Bus, modes, control words, basic programs, 8259 basic, Block diagram, Registers Master slave operation, Control Words, USART, Synchronous , Asynchronous communication, Serial, Parallel Communication	12

COURSE PLAN MAPPED WITH CLO

Week	Content of Course	Teaching-Learning Strategy	Assessment Strategy	Corresponding CLOs
1	Introduction to Microprocessors: Basic concepts, microprocessor-based systems, history, and applications	Lecture, Visual Demonstration	Quiz, Discussion	CLO-1
2	8085 Microprocessor Overview: Block diagram, registers, addressing modes	Lecture, Interactive Discussion	Problem Solving, Quiz	CLO-1
3	8085 Instruction Set: Data transfer, arithmetic, logical, and branching instructions	Lecture, Assembly Language Example	Problem Solving, Assignment	CLO-2
4	Demultiplexing and Flag Registers: Understanding the operation of flag registers and demultiplexing of address/data bus	Lecture, Circuit Demonstration	Quiz, Problem Solving	CLO-1
5	Basic Programs in 8085: Writing and debugging simple programs using the 8085-instruction set	Practical, Hands-On Lab	Lab Report, Assignment	CLO-2
6	Introduction to 8086: Block diagram, registers, pin diagram, and working principle	Lecture, Visual Aids, Demonstration	Quiz, Assignment	CLO-1

COURSE PLAN MAPPED WITH CLO

Week	Course Content	Teaching-Strategy	Assessment Strategy	Corresponding CLOs
7	**Addressing Modes and Instruction Set of 8086: Direct, indirect, register, and immediate addressing modes**	**Lecture, Problem Solving**	**Quiz, Problem Solving**	**CLO-2**
8	8086 Programming: Writing basic programs using assembly language	Lab Session, Practical Demonstration	Practical Exam, Quiz	CLO-2
9	Introduction to 8255: Block diagram, operation, and modes (input/output, bidirectional, handshake)	Lecture, Circuit Demonstration	Quiz, Assignment	CLO-3
10	8255 Control Words and Basic Programming: Understanding control words and programming 8255 for I/O operations	Lecture, Hands-On Lab	Lab Report, Problem Solving	CLO-3
11	8259 Interrupt Controller: Basic operation, master-slave configuration, and control words	Lecture, Circuit Simulation	Quiz, Assignment	CLO-3
12	USART: Introduction to synchronous and asynchronous communication, serial data transmission	Lecture, Practical Examples	Problem Solving, Written Exam	CLO-4

COURSE PLAN MAPPED WITH CLO

Week	Course Content	Teaching-Strategy	Assessment Strategy	Corresponding CLOs
13	Synchronous and Asynchronous Communication: Working principles and applications	Lecture, Demonstration	Quiz, Problem Solving	CLO-4
14	Serial and Parallel Communication: Overview of different communication methods and interfaces	Lecture, Circuit Simulation	Assignment, Quiz	CLO-4
15	Interfacing Microprocessors: Interfacing 8085/8086 with peripherals (LED, 7-segment display, switches)	Lab Session, Practical Circuit Design	Lab Report, Problem Solving	CLO-4
16	Course Review and Integration: Review of microprocessor architecture, programming, and interfacing with peripherals	Group Problem Solving, Review	Quiz, Problem Solving	CLO-1, CLO-2, CLO-3, CLO-4
17	Viva and Presentation: Comprehensive evaluation through oral exams and student presentations	Viva-Voce, Student Presentations	Viva, Presentation	CLO-1, CLO-2, CLO-3, CLO-4

References:

- Ramesh S. Goankar, “Microprocessor Architecture, Programming and Applications with 8085”, 5th Edition, Prentice Hall

- Bharat Acharya Education - YouTube

Week - 1

Basic Concepts of Microprocessors

- Differences between:
  - Microcomputer – a computer with a microprocessor as its CPU. Includes memory, I/O etc.
  - Microprocessor – silicon chip which includes ALU, register circuits & control circuits
  - Microcontroller – silicon chip which includes microprocessor, memory & I/O in a single package.

What is a Microprocessor?

- The word comes from the combination micro and processor.
  - Processor means a device that processes whatever. In this context processor means a device that processes numbers, specifically binary numbers, 0’s and 1’s.
    - To process means to manipulate. It is a general term that describes all manipulation. Again in this content, it means to perform certain operations on the numbers that depend on the microprocessor’s design.

What about micro?

- Micro is a new addition.
  - In the late 1960’s, processors were built using discrete elements.
    - These devices performed the required operation, but were too large and too slow.

  - In the early 1970’s the microchip was invented. All of the components that made up the processor were now placed on a single piece of silicon. The size became several thousand times smaller and the speed became several hundred times faster. The “Micro”Processor was born.

Was there ever a “mini”-processor?

- No.
  - It went directly from discrete elements to a single chip. However, comparing today’s microprocessors to the ones built in the early 1970’s you find an extreme increase in the amount of integration.

- So, What is a microprocessor?

Definition of the Microprocessor

The microprocessor is a programmable device that takes in numbers, performs on them arithmetic or logical operations according to the program stored in memory and then produces other numbers as a result.

Definition (Contd.)

- Takes in: The data that the microprocessor manipulates must come from somewhere.
    - It comes from what is called “input devices”.
    - These are devices that bring data into the system from the outside world.
    - These represent devices such as a keyboard, a mouse, switches, and the like.

Definition (Contd.)

- Numbers: The microprocessor has a very narrow view on life. It only understands binary numbers.

  A binary digit is called a bit (which comes from **b**inary dig**it**).

  The microprocessor recognizes and processes a group of bits together. This group of bits is called a “word”.

  The number of bits in a Microprocessor’s word, is a measure of its “abilities”.

Definition (Contd.)

- Words, Bytes, etc.
  - The earliest microprocessor (the Intel 8088 and Motorola's 6800) recognized 8-bit words.
    - They processed information 8-bits at a time. That's why they are called "8-bit processors". They can handle large numbers, but in order to process these numbers, they broke them into 8-bit pieces and processed each group of 8-bits separately.
  - Later microprocessors (8086 and 68000) were designed with 16-bit words.
    - A group of 8-bits were referred to as a "half-word" or "byte".
    - A group of 4 bits is called a "nibble".
    - Also, 32 bit groups were given the name "long word".
  - Today, all processors manipulate at least 32 bits at a time and there exists microprocessors that can process 64, 80, 128 bits

Definition (Contd.)

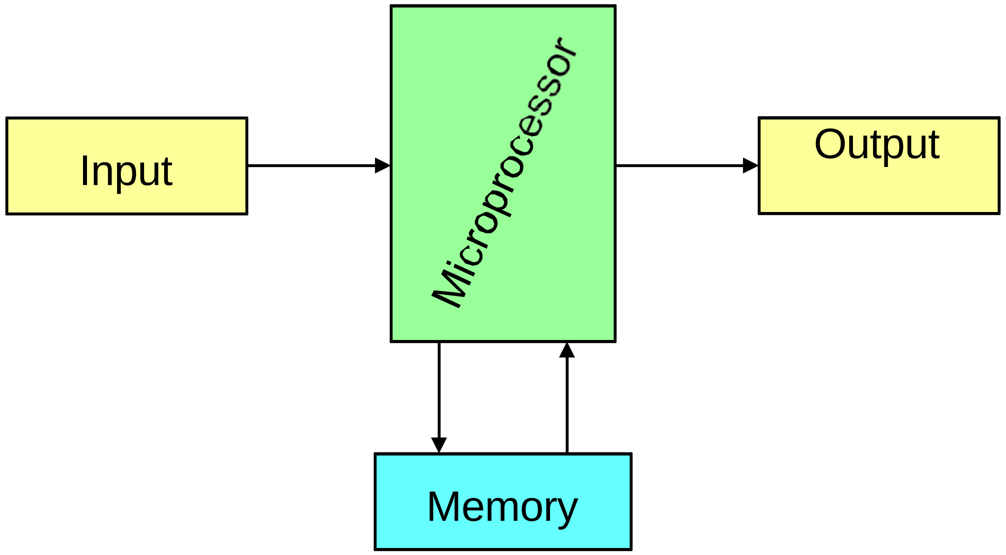
- Stored in memory:
  - First, what is memory?
    - Memory is the location where information is kept while not in current use.
    - Memory is a collection of storage devices. Usually, each storage device holds one bit. Also, in most kinds of memory, these storage devices are grouped into groups of 8. These 8 storage locations can only be accessed together. So, one can only read or write in terms of bytes to and form memory.
    - Memory is usually measured by the number of bytes it can hold. It is measured in Kilos, Megas and lately Gigas. A Kilo in computer language is 2^{10} =1024. So, a KB (KiloByte) is 1024 bytes. Mega is 1024 Kilos and Giga is 1024 Mega.

Definition (Contd.)

- Stored in memory:
  - When a program is entered into a computer, it is stored in memory. Then as the microprocessor starts to execute the instructions, it brings the instructions from memory one at a time.
  - Memory is also used to hold the data.
    - The microprocessor reads (brings in) the data from memory when it needs it and writes (stores) the results into memory when it is done.

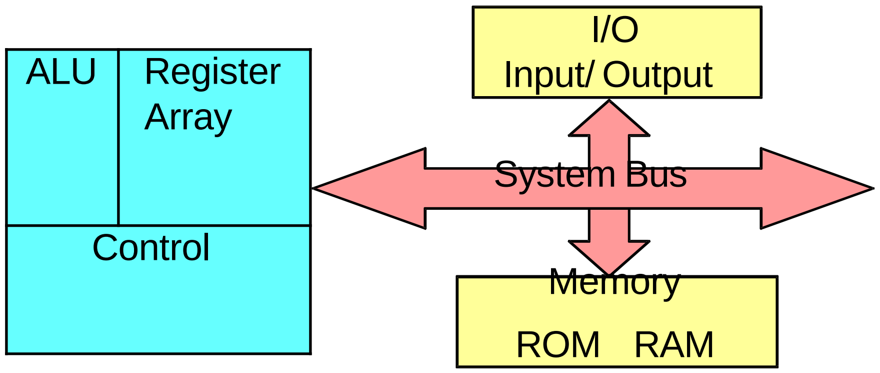
A Microprocessor-based system

From the above description, we can draw the following block diagram to represent a microprocessor-based system:



Organization of a microprocessor-based system

- Let’s expand the picture a bit.



Microprocessor

Fifth Generation **Pentium**

Fourth Generation
During 1980s
Low power version of HMOS technology (HCMOS)
32 bit processors
Physical memory space 2^{24} bytes = 16 Mb
Virtual memory space 2^{40} bytes = 1 Tb
Floating point hardware
Supports increased number of addressing modes

Intel 80386

Third Generation
During 1978
HMOS technology ⇒ Faster speed, Higher packing density
16 bit processors ⇒ 40/ 48/ 64 pins
Easier to program
Dynamically relatable programs
Processor has multiply/ divide arithmetic hardware
More powerful interrupt handling capabilities
Flexible I/O port addressing

Intel 8086 (16 bit processor)

Second Generation
During 1973
NMOS technology ⇒ Faster speed, Higher density, Compatible with TTL
4 / 8/ 16 bit processors ⇒ 40 pins
Ability to address large memory spaces and I/O ports
Greater number of levels of subroutine nesting
Better interrupt handling capabilities

Intel 8085 (8 bit processor)

First Generation
Between 1971 – 1973
PMOS technology, non compatible with TTL
4 bit processors ⇒ 16 pins
8 and 16 bit processors ⇒ 40 pins
Due to limitations of pins, signals are multiplexed

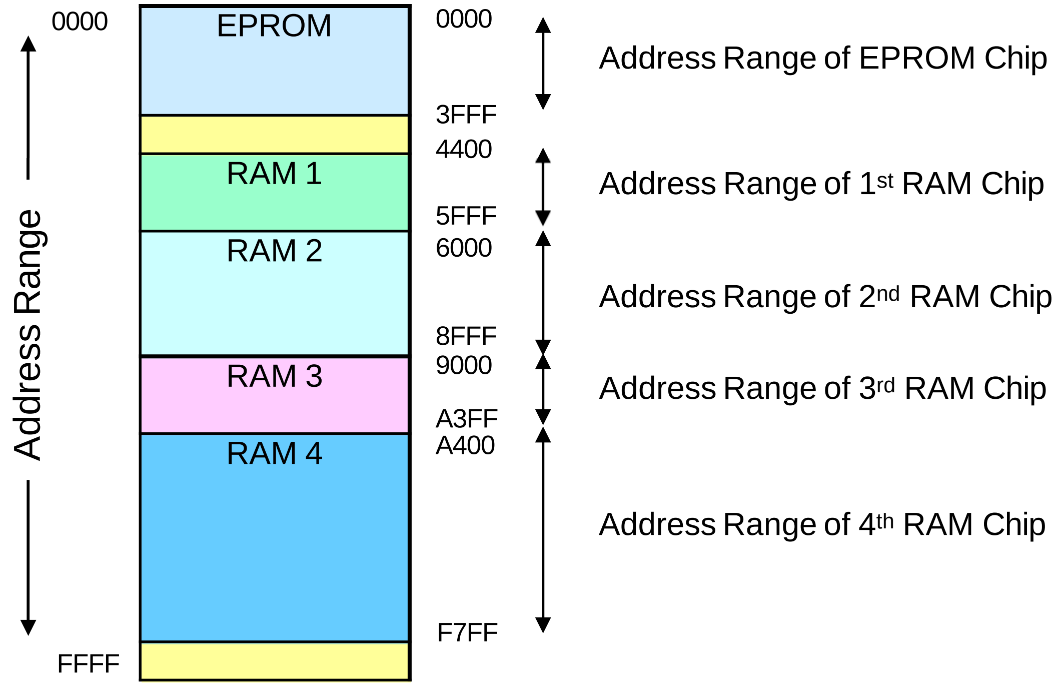
Memory

- Memory stores information such as instructions and data in binary format (0 and 1). It provides this information to the microprocessor whenever it is needed.

- Usually, there is a memory “sub-system” in a microprocessor-based system. This sub-system includes:
  - The registers inside the microprocessor
  - Read Only Memory (ROM)
    - used to store information that does not change.
  - Random Access Memory (RAM) (also known as Read/Write Memory).
    - used to store information supplied by the user. Such as programs and data.

Memory Map and Addresses

- The memory map is a picture representation of the address range and shows where the different memory chips are located within the address range.



0000
FFFF
Address Range
EPROM
RAM 1
RAM 2
RAM 3
RAM 4
0000
3FFF
4400
5FFF
6000
8FFF
9000
A3FF
A400
F7FF
Address Range of EPROM Chip
Address Range of 1st RAM Chip
Address Range of 2nd RAM Chip
Address Range of 3rd RAM Chip
Address Range of 4th RAM Chip

Machine Language

- The number of bits that form the "word" of a microprocessor is fixed for that particular processor.
  - These bits define a maximum number of combinations.
    - For example an 8-bit microprocessor can have at most $2^8 = 256$ different combinations.

- However, in most microprocessors, not all of these combinations are used.
  - Certain patterns are chosen and assigned specific meanings.
  - Each of these patterns forms an instruction for the microprocessor.
  - The complete set of patterns makes up the microprocessor's machine language.

Week - 2

The 8085 Machine Language

- The 8085 (from Intel) is an 8-bit microprocessor.
  - The 8085 uses a total of 246 bit patterns to form its instruction set.
  - These 246 patterns represent only 74 instructions.
    - The reason for the difference is that some (actually most) instructions have multiple different formats.
  - Because it is very difficult to enter the bit patterns correctly, they are usually entered in hexadecimal instead of binary.
    - For example, the combination 0011 1100 which translates into “increment the number in the register called the accumulator”, is usually entered as 3C.

Assembly Language

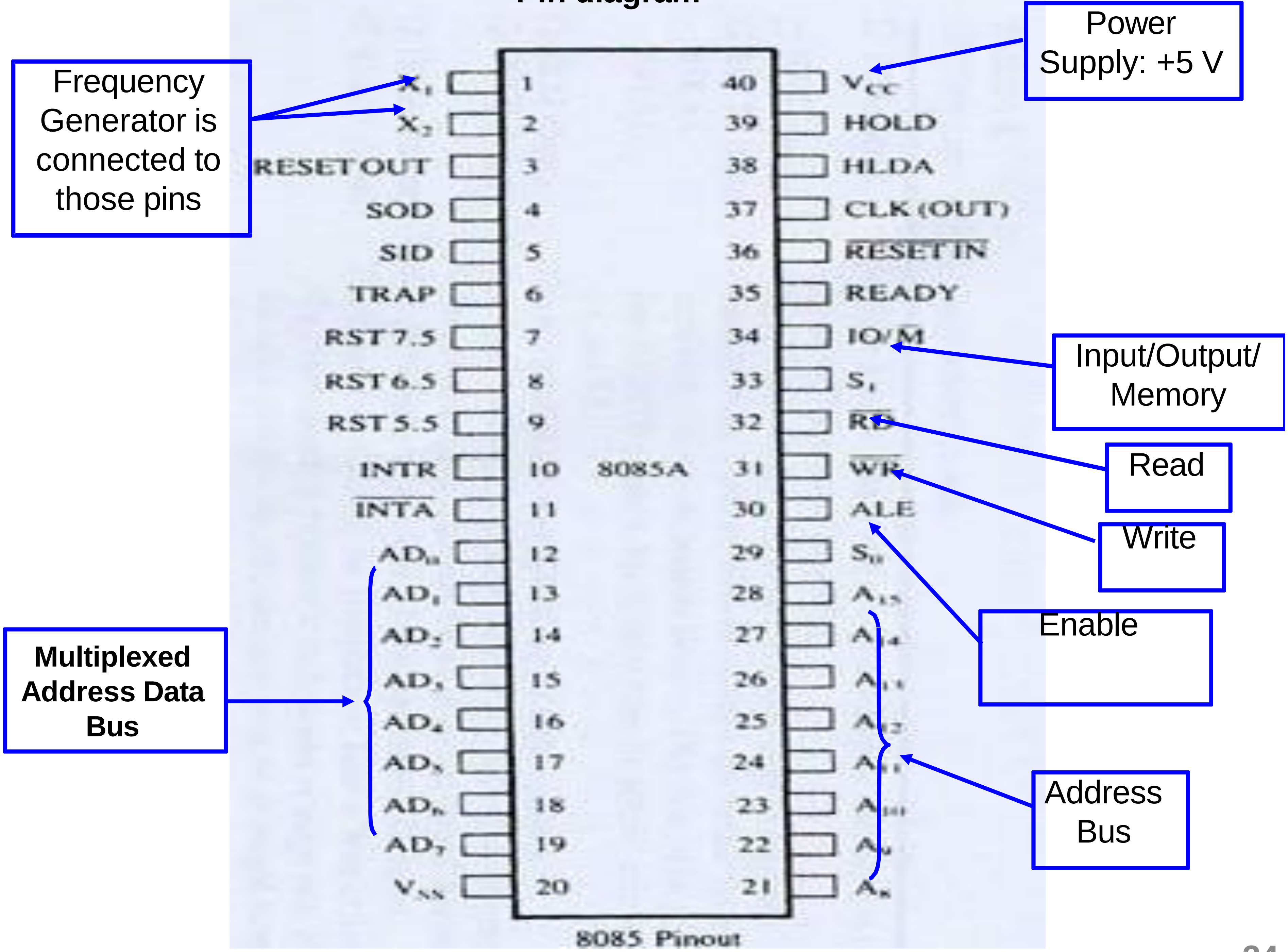
- Using the same example from before,
  - 00111100 translates to 3C in hexadecimal (OPCODE)
  - Its mnemonic is: “INR A”.
  - INR stands for “increment register” and A is short for accumulator.

- Another example is: 1000 0000,
  - Which translates to 80 in hexadecimal.
  - Its mnemonic is “ADD B”.
  - “Add register B to the accumulator and keep the result in the accumulator”.

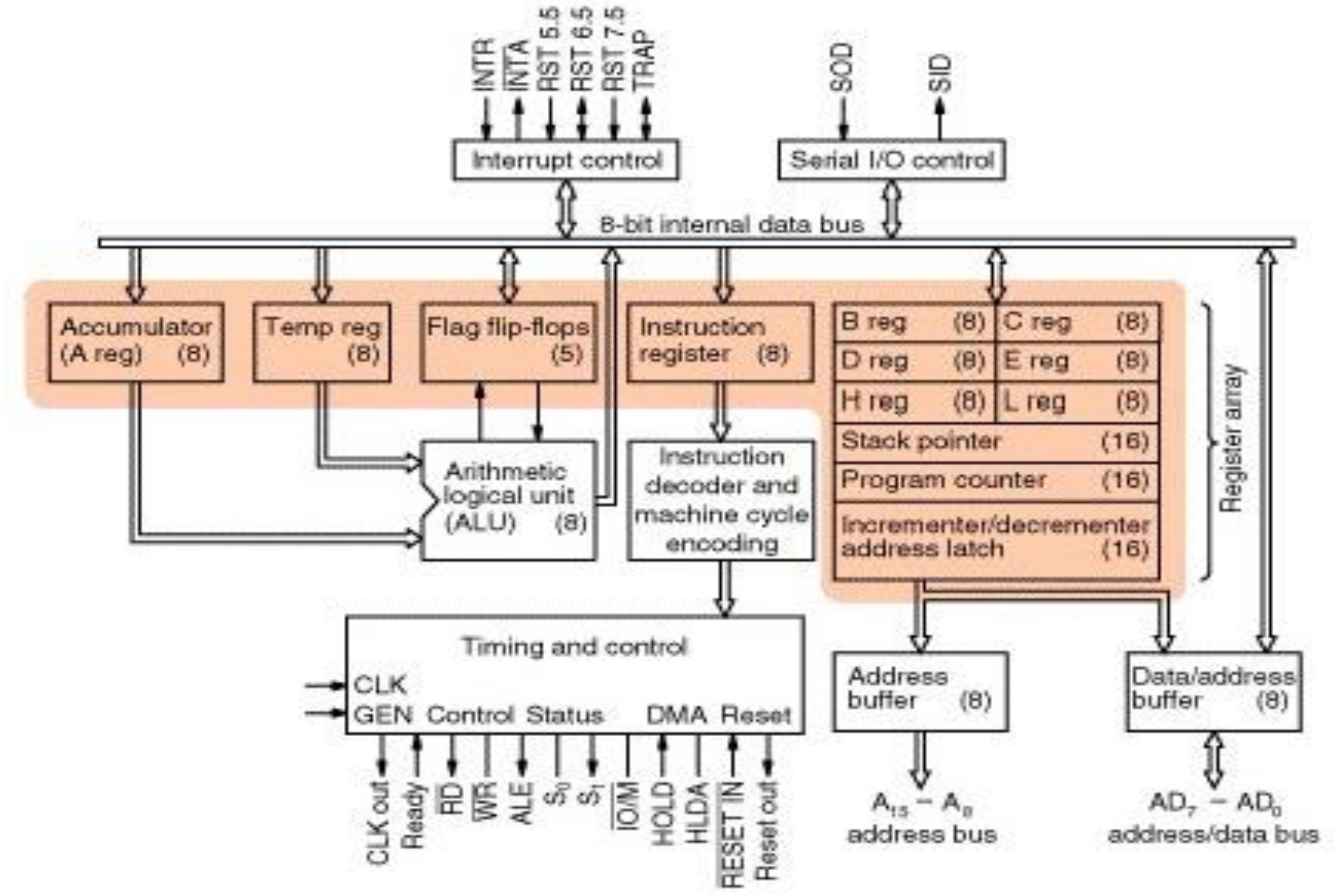
8085 Microprocessor Architecture

- 8-bit general purpose µp
- Capable of addressing 64 k of memory
- Has 40 pins
- Requires +5 v power supply
- Can operate with 3 MHz clock

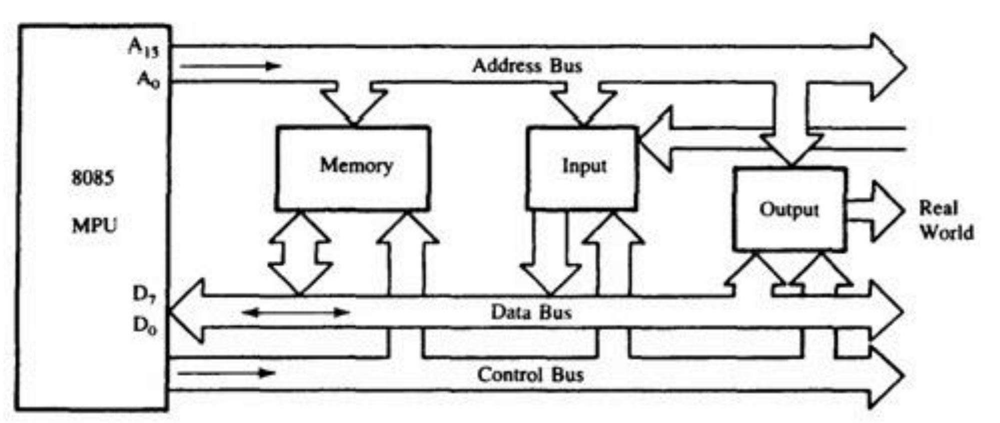
Pin diagram


Frequency Generator is connected to those pins
Power Supply: +5 V
Input/Output/ Memory
Read
Write
Enable
Address Bus
Multiplexed Address Data Bus
X1
X2
RESET OUT
SOD
SID
TRAP
RST 7.5
RST 6.5
RST 5.5
INTR
INTA
AD0
AD1
AD2
AD3
AD4
AD5
AD6
AD7
VSS
VCC
HOLD
HLDA
CLK (OUT)
RESET IN
READY
IO/M
S1
RD
WR
ALE
S0
A15
A14
A13
A12
A11
A10
A9
A8
8085A
8085 Pinout

Architecture of Intel 8085 Microprocessor



- System Bus – wires connecting memory & I/O to microprocessor
  - Address Bus
    - Unidirectional
    - Identifying peripheral or memory location
  - Data Bus
    - Bidirectional
    - Transferring data
  - Control Bus
    - Synchronization signals
    - Timing signals
    - Control signal



Address Bus

- The address bus consists of 16, 20, 24 or 32 parallel signal lines.
- On these lines the CPU sends out the address of the memory location that is to be written to or read from. The no of memory location that the CPU can address is determined by the number of address lines.
- If the CPU has N address lines, then it can directly address 2N memory locations i.e. CPU with 16 address lines can address 216 or 65536 memory locations.

Data Bus

- The data bus consists of 8, 16 or 32 parallel signal lines.
- The data bus lines are bi-directional.
- This means that the CPU can read data in from memory or it can send data out to memory

Control Bus

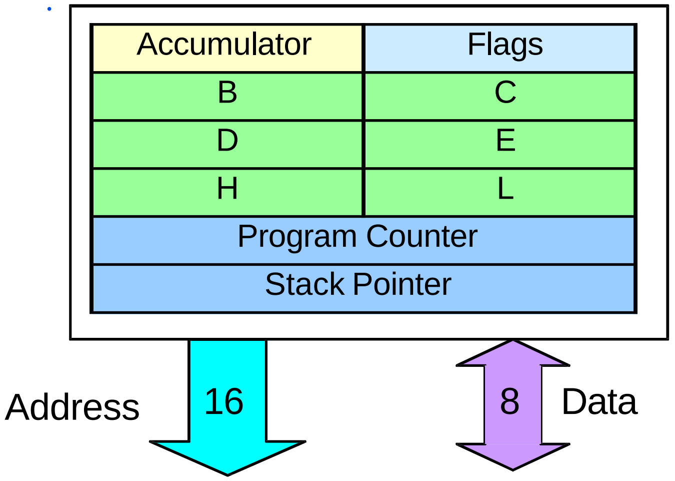
- The control bus consists of 4 to 10 parallel signal lines.
- The CPU sends out signals on the control bus to enable the output of addressed memory devices or port devices.
- Typical control bus signals are Memory Read, Memory Write, I/O Read and I/O Write.

The ALU

- In addition to the arithmetic & logic circuits, the ALU includes the accumulator, which is part of every arithmetic & logic operation.

- Also, the ALU includes a temporary register used for holding data temporarily during the execution of the operation. This temporary register is not accessible by the programmer.

- Registers
  - General Purpose Registers
    - B, C, D, E, H & L (8 bit registers)
    - Can be used singly
    - Or can be used as 16 bit register pairs
      - BC, DE, HL
    - H & L can be used as a data pointer (holds memory address)
  - Special Purpose Registers
    - Accumulator (8 bit register)
      - Store 8 bit data
      - Store the result of an operation
      - Store 8 bit data during I/O transfer



Week - 3

Flag Register

- 8 bit register – shows the status of the microprocessor before/after an operation
- S (sign flag), Z (zero flag), AC (auxillary carry flag), P (parity flag) & CY (carry flag)

D7	D6	D5	D4	D3	D2	D1	D0
S	Z		AC		P		CY

- Sign Flag
  - Used for indicating the sign of the data in the accumulator
  - The sign flag is set if negative (1 – negative)
  - The sign flag is reset if positive (0 –positive)

- Zero Flag
  - Is set if result obtained after an operation is 0
  - Is set following an increment or decrement operation of that register

```
     10110011
+    01001101
    ---------------
1    00000000
```

- Carry Flag
  - Is set if there is a carry or borrow from arithmetic operation

```
           1011 0101              1011 0101
       +   0110 1100          -   1100 1100
          ---------------        ---------------
Carry 1    0010 0001    Borrow 1  1110 1001
```

- Auxillary Carry Flag
  - Is set if there is a carry out of bit 3
- Parity Flag
  - Is set if parity is even
  - Is cleared if parity is odd

- The Program Counter (PC)
  - This is a register that is used to control the sequencing of the execution of instructions.
  - This register always holds the address of the next instruction.
  - Since it holds an address, it must be 16 bits wide.

- The Stack pointer
  - The stack pointer is also a 16-bit register that is used to point into memory.
  - The memory this register points to is a special area called the stack.
  - The stack is an area of memory used to hold data that will be retreived soon.
  - The stack is usually accessed in a Last In First Out (LIFO) fashion.

Week - 4

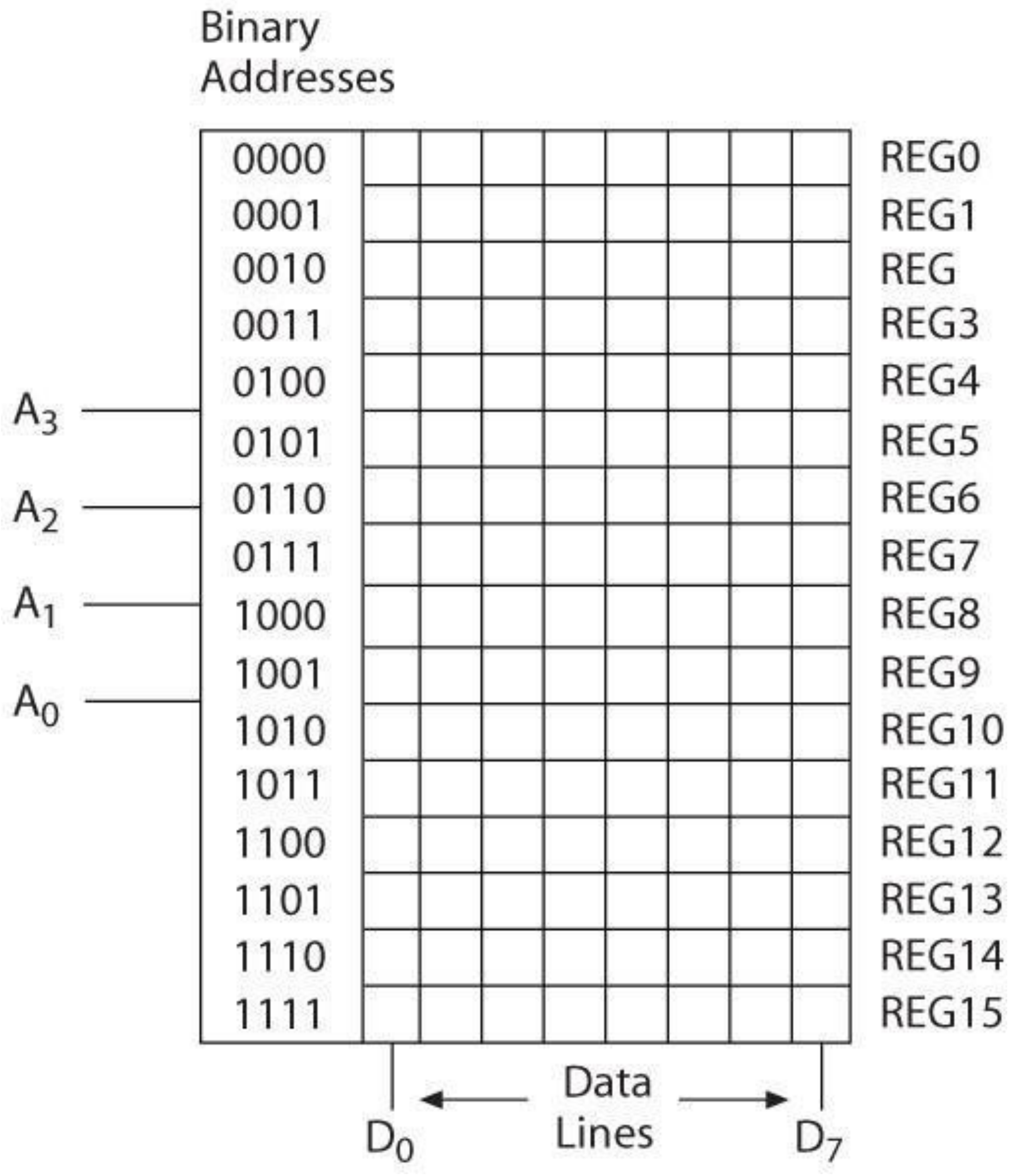
Memory

- Memory consist of RAM and ROM, the purpose of memory is to store binary codes for the sequences of instructions you want the computer to carry out.
- The second purpose of the memory is to store the binary-coded data with which the computer is going to be working.

Input / Output

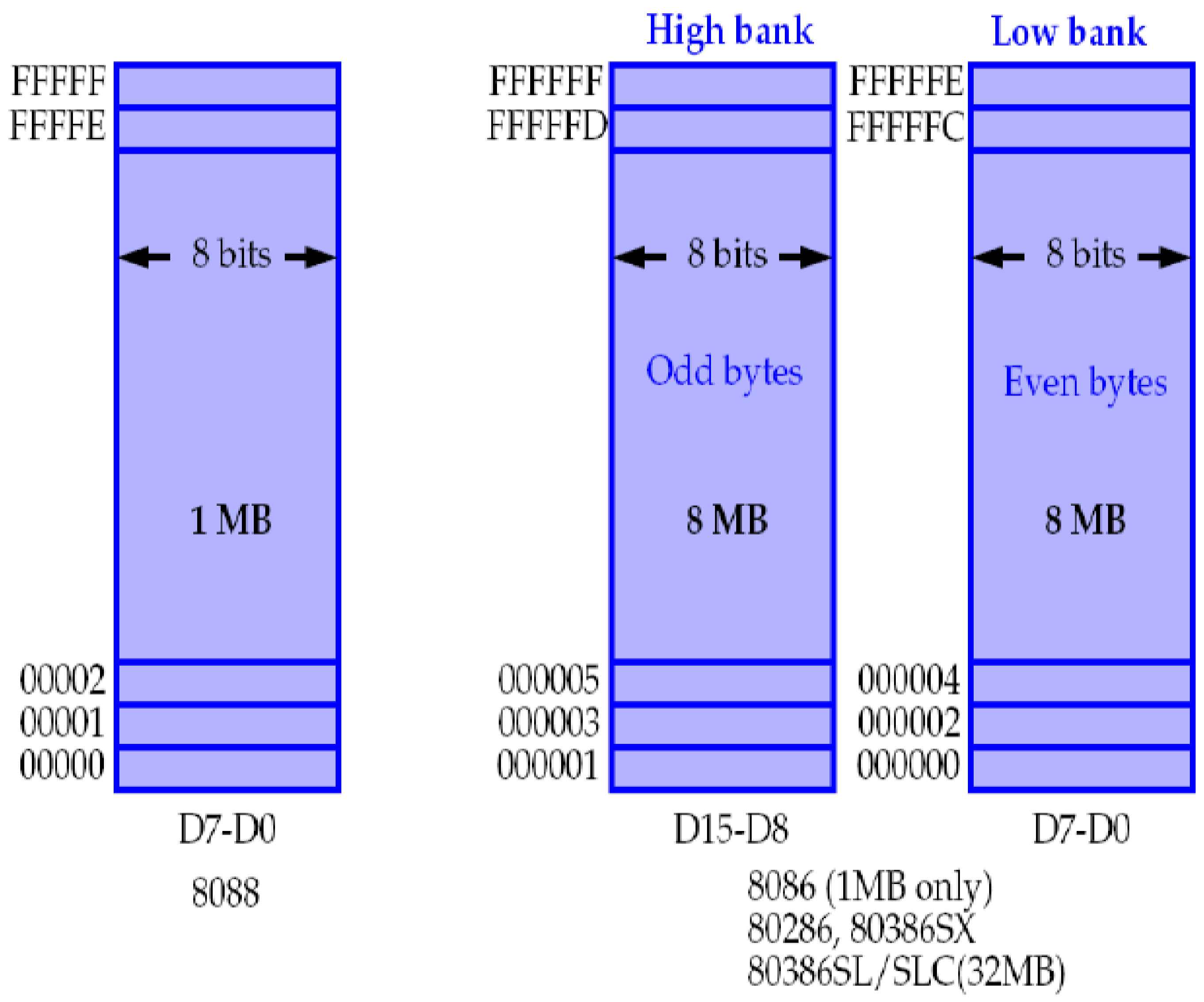
- The input/output or I/O Section allows the computer to take in data from the out side world or send data to the outside world.
- Peripherals such as keyboards, video display terminals, printers are connected to I/O Port.

Memory

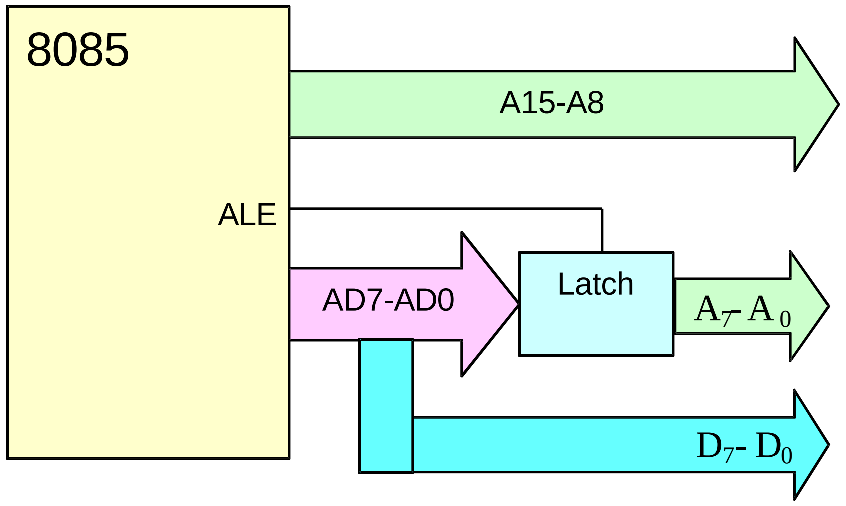

Binary Addresses
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
REG0
REG1
REG
REG3
REG4
REG5
REG6
REG7
REG8
REG9
REG10
REG11
REG12
REG13
REG14
REG15
A_3
A_2
A_1
A_0
D_0
Data Lines
D_7

Basic Memory Architecture

Bank layout



Demultiplexing AD7-AD0



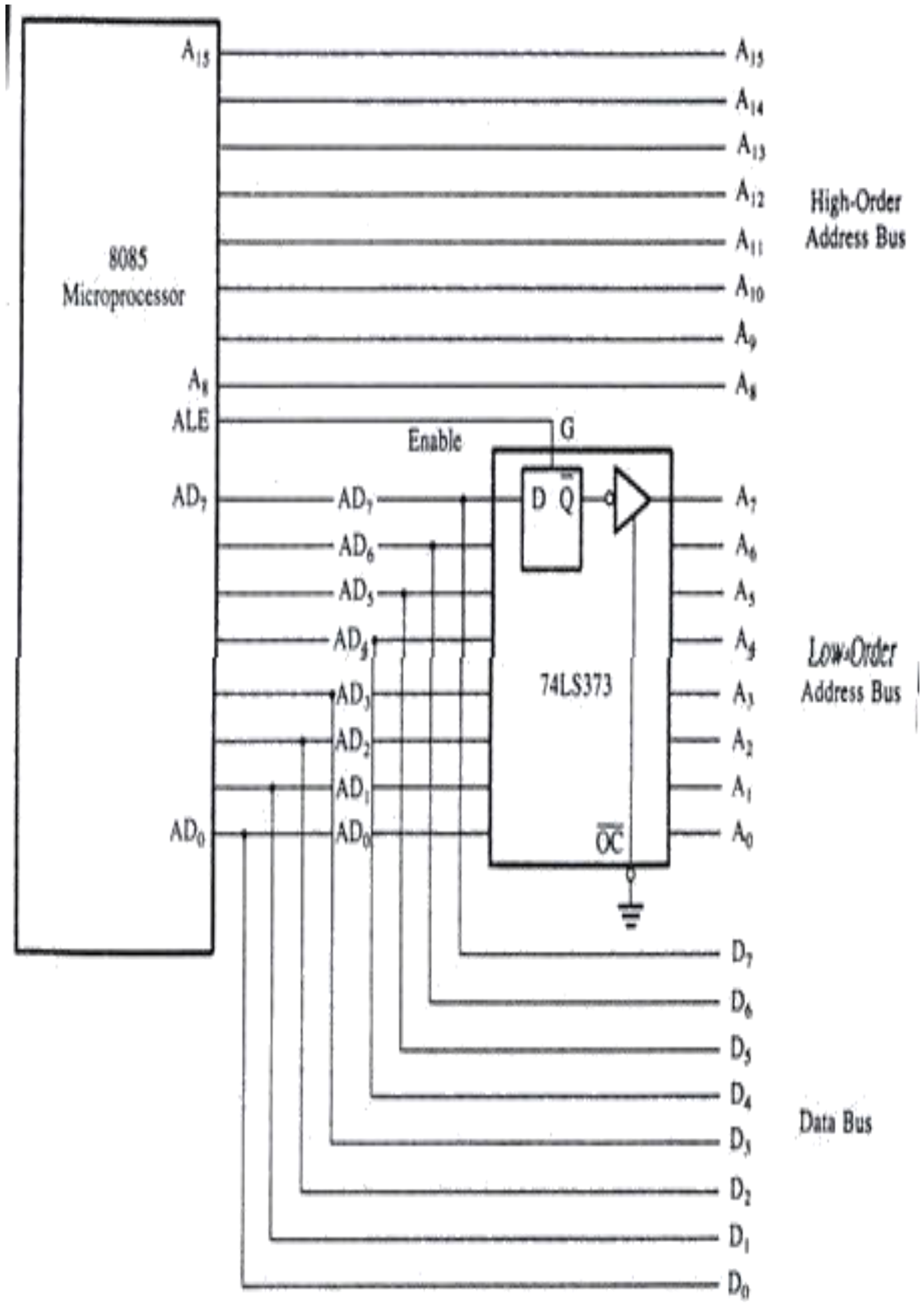
– Given that ALE operates as a pulse during T1, we will be able to latch the address. Then when ALE goes low, the address is saved and the AD7–AD0 lines can be used for their purpose as the bi-directional data lines.

Demultiplexing AD7-AD0

- From the above description, it becomes obvious that the AD7– AD0 lines are serving a dual purpose and that they need to be demultiplexed to get all the information.
- The high order bits of the address remain on the bus for three clock periods. However, the low order bits r ain for only on lo k p riod and t y would be lost if they are not saved externally. Also, notice that the low order bits of the address disappear when they are needed most.
- To make sure we have the entire address for the full three clock cycles, we will use an external latch to save the value of AD7– AD0 when it is carrying the address bits. We use the ALE signal to enable this latch.

Demultiplexing the Bus AD_7 – AD_0

- The high order address is placed on the address bus and hold for 3 clk periods,
- The low order address is lost after the first clk period, this address needs to be hold however we need to use latch
- The address AD7 – AD0 is connected as inputs to the latch 74LS373.
- The ALE signal is connected to the enable (G) pin of the latch and the OC – Output control – of the latch is grounded


8085
Microprocessor
A15
A8
ALE
AD7
AD0
Enable
G
AD7
AD6
AD5
AD4
AD3
AD2
AD1
AD0
D
Q
74LS373
OC
A15
A14
A13
A12
A11
A10
A9
A8
High-Order
Address Bus
A7
A6
A5
A4
A3
A2
A1
A0
Low-Order
Address Bus
D7
D6
D5
D4
D3
D2
D1
D0
Data Bus

How Does the Microprocessor Work?

- Assume that a program and data are already entered in the R/W memory.
- The microprocessor fetches the first instruction from its memory sheet, decodes it, and executes that instruction.
- The sequence of fetch, decode, and execute is continued until the microprocessor comes across an instruction to stop.
- During the entire process, the microprocessor uses the system bus to fetch the binary instructions and data from the memory.
- It uses registers from the register section to store data temporarily, and it performs the computing function in the ALU section.
- Finally, it sends out the result in binary, using the same bus lines, to memory or an output device.

How Does a Microprocessor Differentiate Between Data and Instruction Code?

The microprocessor interprets the first byte it fetches as an opcode. In the example, we tell the processor that our program begins at location 2000H. The first code it fetches is 3EH. When it decodes that code, it knows that it is a two-byte instruction. Therefore, it assumes that the second code, 32H, is a data byte. If we forget to enter 32H and enter the next code, 06H, instead, the 8085 will load 06H in the accumulator, interpret the next code, 48H, as an opcode, and continue the execution in sequence. As a consequence, we may encounter a totally unexpected result.

Mnemonics	Hex Code	Memory Address
MVI A, 32H	3E	2000H
	32	2001H
MVI B,48H	06	2002H
	48	2003H
ADD B	80	2004H

Week - 5

PROGRAM

Example: Write a program to perform 16 bit addition of 1234H & 4321H. Store answer at H & L registers.

```
MVI B, 21H                              B=21H
MVI A, 34H                              A=34H
MVI C, 43H                              C=43H
MVI D, 12H                              D=12H
ADD B                                   A=34+21H
MOV L, A                                L=55H
MOV A, C                                A=43H
ADC D                                   A=43+12H
MOV H, A                                H=55H
HLT                                     STOP.
```

PROGRAM

Example:Write a program to exchange contents of memory location D000H to D001H

LDA D000H	Load Acc with data from D000
MOV B, A	Move the data to B
LDA D001H	Load Acc with data from D001
STA D000H	Store Acc data at D000
MOV A,B	Move B’s data to A
STA D001H	Store Acc data at D001

PROGRAM

Write a program to reset last 4 bits of the number 32H

Store result at C200H.

MVI A, 32H	A=32H
ANI F0H	00110010 AND
	1111000
STA C200H	C200=30H
RST1	Stop

PROGRAM

Write a program to add 10 data bytes. Data is stored from locations C200H. Store result at C300H.

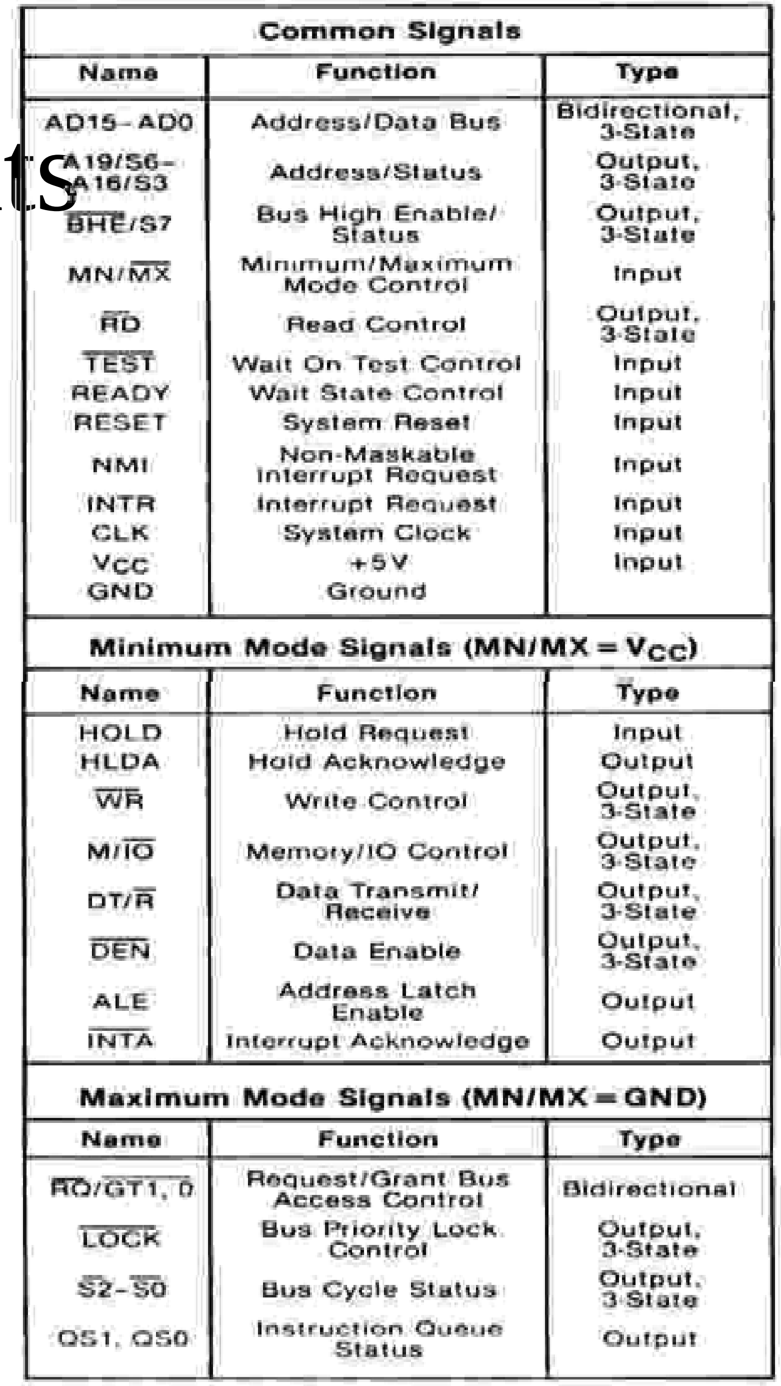
```
        LXI H,C200H
        MVI C, 0AH
        MVI A,00H
UP      MOV B,M
        ADD B
        INX H
        DCR C
        JNZ UP
        STA C300H
        RST1
```

8086 Features

- **16-bit Arithmetic Logic Unit**
- **16-bit data bus**
- **20-bit address bus - 2^{20} = 1,048,576 = 1 meg**

Week - 6

Pinouts



Common Signals

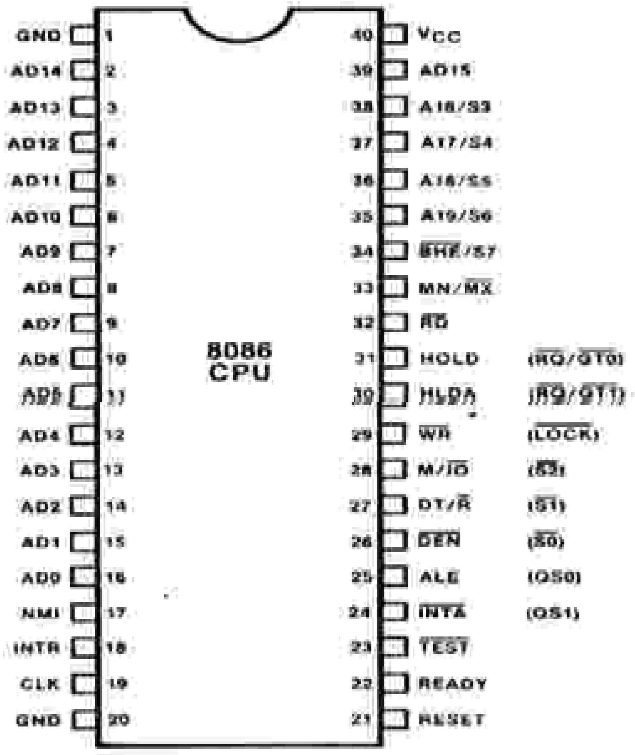
Name	Function	Type
AD15–AD0	Address/Data Bus	Bidirectional, 3-State
A19/S6–A16/S3	Address/Status	Output, 3-State
$\overline{BHE}$/S7	Bus High Enable/ Status	Output, 3-State
MN/$\overline{MX}$	Minimum/Maximum Mode Control	Input
$\overline{RD}$	Read Control	Output, 3-State
$\overline{TEST}$	Wait On Test Control	Input
READY	Wait State Control	Input
RESET	System Reset	Input
NMI	Non-Maskable Interrupt Request	Input
INTR	Interrupt Request	Input
CLK	System Clock	Input
V_{CC}	+5V	Input
GND	Ground	

Minimum Mode Signals (MN/MX = V_{CC})

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

Maximum Mode Signals (MN/MX = GND)

Name	Function	Type
$\overline{RQ}$/$\overline{GT1}$, 0	Request/Grant Bus Access Control	Bidirectional
$\overline{LOCK}$	Bus Priority Lock Control	Output, 3-State
$\overline{S2}$–$\overline{S0}$	Bus Cycle Status	Output, 3-State
QS1, QS0	Instruction Queue Status	Output



MAXIMUM MODE PIN FUNCTIONS (e.g. $\overline{LOCK}$) ARE SHOWN IN PARENTHESES

8086 Pins

The 8086 comes in a 40 pin package which means that some pins have more than one use or are multiplexed. The packaging technology of time limited the number of pin that could be used.

In particular, the address lines 0 - 15 are multiplexed with data lines 0-15, address lines 16-19 are multiplexed with status lines. These pins are

AD0 - AD15, A16/S3 - A19/S6

The 8086 has one other pin that is multiplexed and this is BHE'/S7. BHE stands for Bus High Enable. This is an active low signal that is asserted when there is data on the upper half of the data bus.

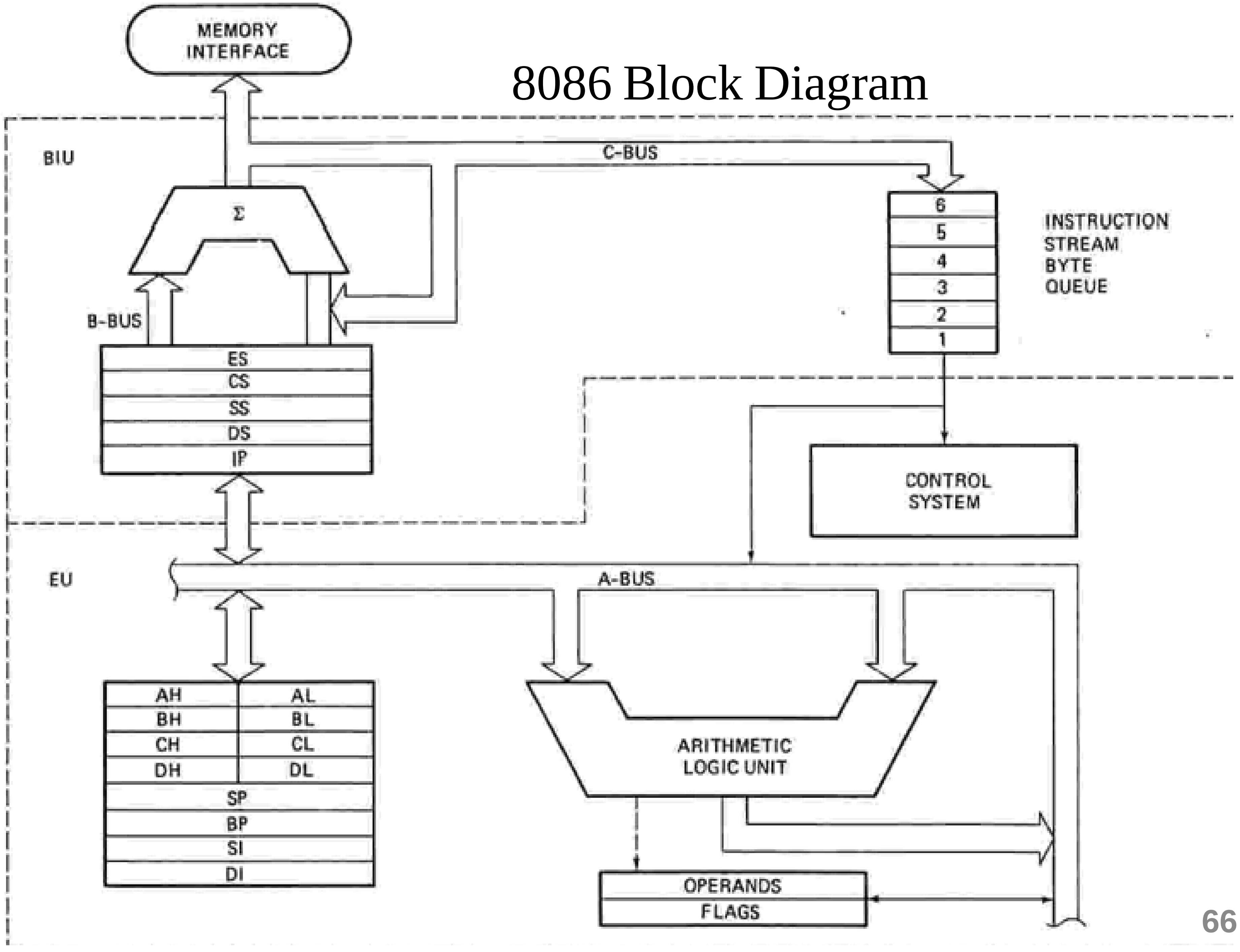
The 8086 has two modes of operation that changes the function of some pins. The SDK-86 uses the 8086 in the minimum mode with the MN/MX' pin tied to 5 volts. This is a simple single processor mode. To work in maximum mode, the MN/MX' pin is tied to ground. This is the mode required for a coprocessor like the 8087.

8086 Pins

In the <u>minimum mode</u> the following pins are available.

HOLD	**When this pin is high, another master is requesting control of the local bus, e.g., a DMA controller.**
HLDA	**HOLD Acknowledge: the 8086 signals that it is going to float the local bus.**
WR’	**Write: the processor is performing a write memory or I/O operation.**
M/IO’	**Memory or I/O operation.**
DT/R’	**Data Transmit or Receive.**
DEN’	**Data Enable: data is on the multiplexed address/data pins.**
ALE	**Address Latch Enable: the address is on the address/data pins. This signal is used to capture the address in latches to establish the address bus.**
INTA’	**Interrupt acknowledge: acknowledges external interrupt requests.**

8086 Block Diagram



8086 Architecture

- **The 8086 has two parts, the Bus Interface Unit (BIU) and the Execution Unit (EU).**
- **The BIU fetches instructions, reads and writes data, and computes the 20-bit address.**
- **The EU decodes and executes the instructions using the 16-bit ALU.**
- **The BIU contains the following registers:**

  IP - Instruction Pointer
  CS - Code Segment Register
  DS - Data Segment Register
  SS - Stack Segment Register
  ES - Extra Segment Register

The BIU fetches instructions using the CS and IP, written CS:IP, to contruct the 20-bit address. Data is fetched using a segment register (usually the DS) and an effective address (EA) computed by the EU depending on the addressing mode.

8086 Architecture

The EU contains the following 16-bit registers:

AX – Accumulator
BX - Base Register
CX - Count Register
DX - Data Register
SP - Stack Pointer \ defaults to stack segment
BP - Base Pointer /
SI - Source Index Register
DI - Destination Register

These are referred to as general-purpose registers, although, as seen by their names, they often have a special-purpose use for some instructions.

The AX, BX, CX, and DX registers can be considers as two 8-bit registers, a High byte and a Low byte. This allows byte operations and compatibility with the previous generation of 8-bit processors, the 8080 and 8085. 8085 source code could be translated in 8086 code and assembled. The 8-bit registers are:

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

I-11

Execution Unit (EU)

Flag Register

Auxiliary Carry Flag

This flag is set, when there is a carry out of low-nibble in case of addition or a borrow in case of subtraction

Carry Flag

This flag is set, when there is a carry out of MSB in case of addition or a borrow in case of subtraction.

Sign Flag

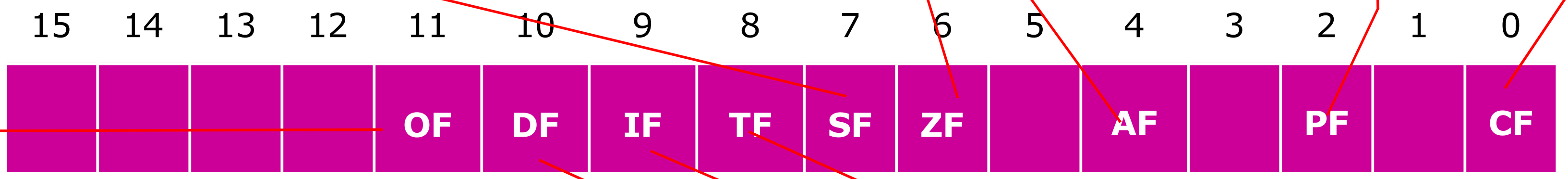
After any operation if the MSB is 1, then it indicates that the number is negative. And this flag is set to 1

Zero Flag

This flag is set, if the result of the computation or comparison performed by an instruction is zero

Parity Flag

This is even parity flag. When result has even number of 1, it will be set to 1, otherwise 0 for odd number of 1s



Trap Flag

If this flag is set, the processor enters the single step execution mode by generating internal interrupts after the execution of each instruction

Over flow Flag

The overflow flag is set to 1 when the result of a signed operation is too large to fit.

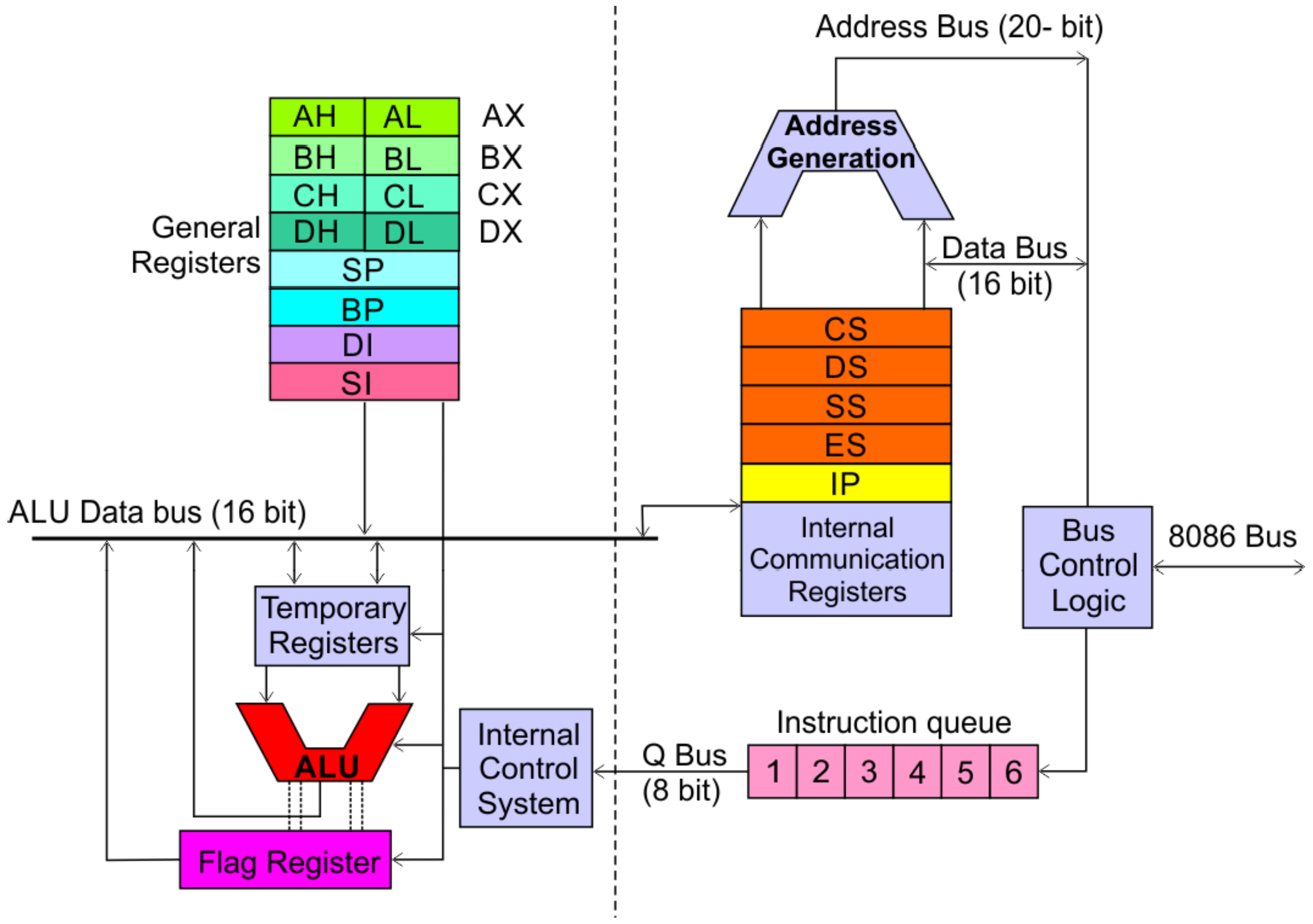
Direction Flag

This is used by string manipulation instructions. If this flag bit is '0', the string is processed beginning from the lowest address to the highest address, i.e., auto incrementing mode. Otherwise, the string is processed from the highest address towards the lowest address, i.e., auto decrementing mode.

Interrupt Flag

Causes the 8086 to recognize external mask interrupts; clearing IF disables these interrupts.

Architecture



Execution Unit (EU)

EU executes instructions that have already been fetched by the BIU.

BIU and EU functions separately.

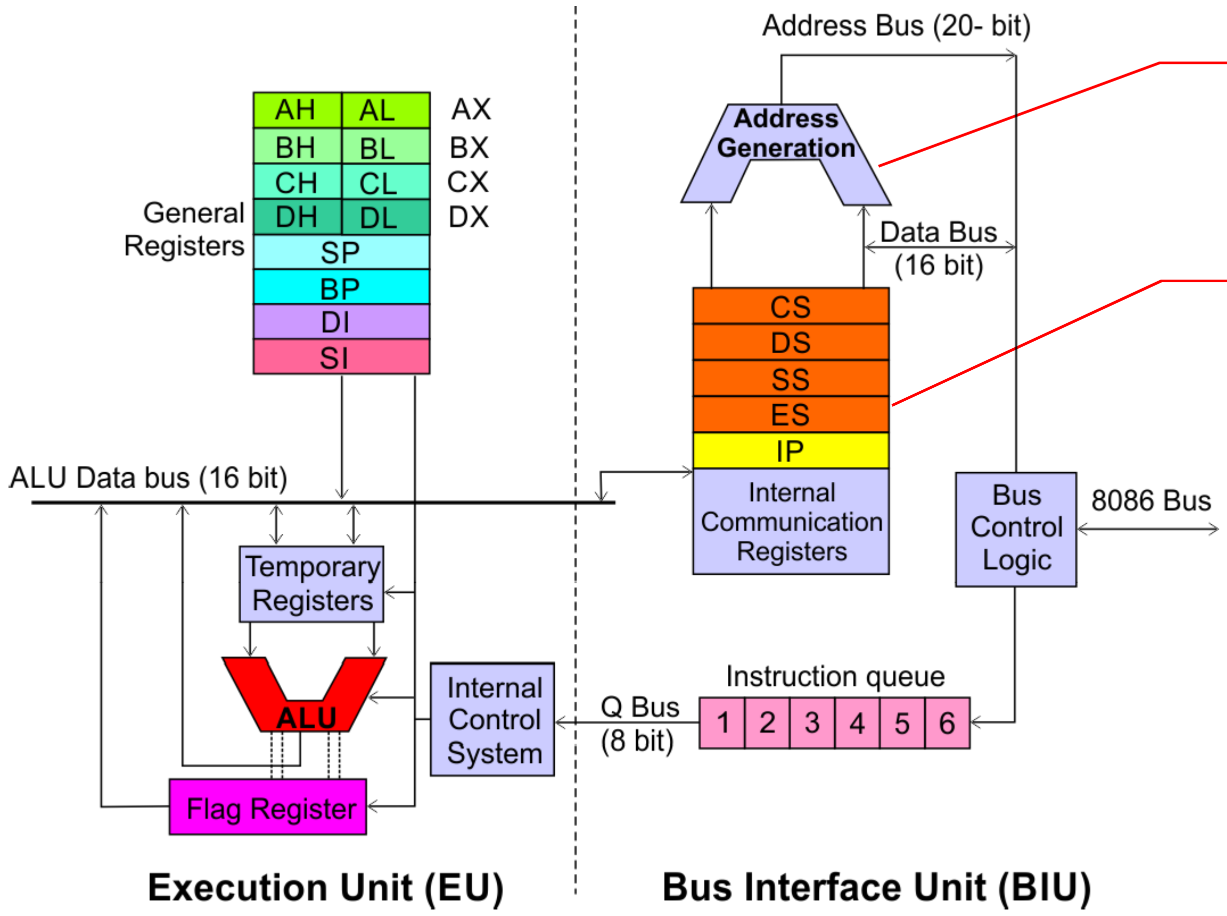
Bus Interface Unit (BIU)

BIU fetches instructions, reads data from memory and I/O ports, writes data to memory and I/ O ports.

8086 Microprocessor

Architecture

Bus Interface Unit (BIU)



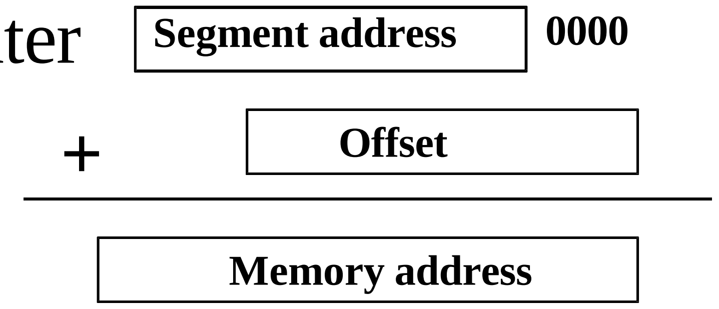
Dedicated Adder to generate 20 bit address

Four 16-bit segment registers

Code Segment (CS)
Data Segment (DS)
Stack Segment (SS)
Extra Segment (ES)

Memory Address Calculation

- Segment addresses must be stored in segment registers
- Offset is derived from the combination of pointer registers, the Instruction Pointer (IP), and immediate values



- **Examples**

CS	3	4	8	A	0
IP +		4	2	1	4
Instruction address	3	8	A	B	4

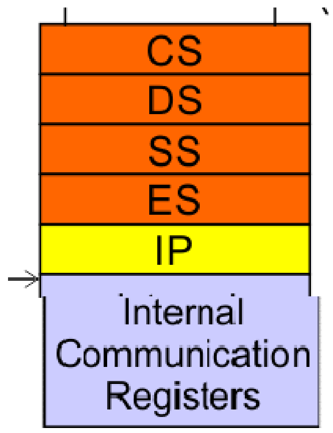
SS	5	0	0	0	0
SP +		F	F	E	0
Stack address	5	F	F	E	0

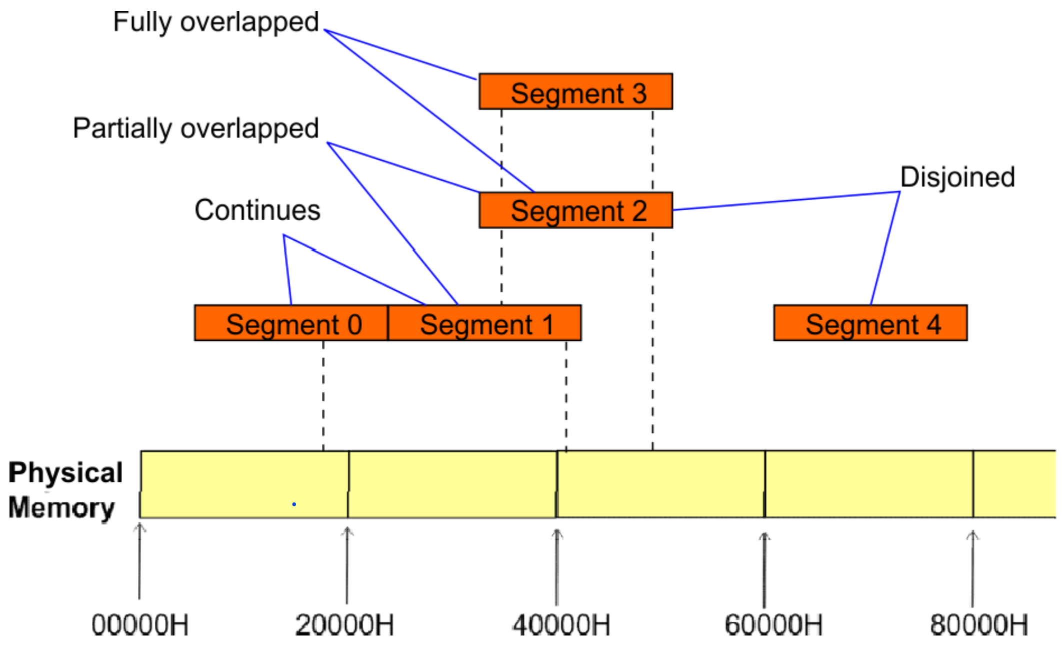
DS	1	2	3	4	0
DI +		0	0	2	2
Data address	1	2	3	6	2

Architecture

Bus Interface Unit (BIU)

Segment Registers





- 8086's 1-megabyte memory is divided into segments of up to 64K bytes each.
- The 8086 can directly address four segments (256 K bytes within the 1 M byte of memory) at a particular time.
- Programs obtain access to code and data in the segments by changing the segment register content to point to the desired segments.

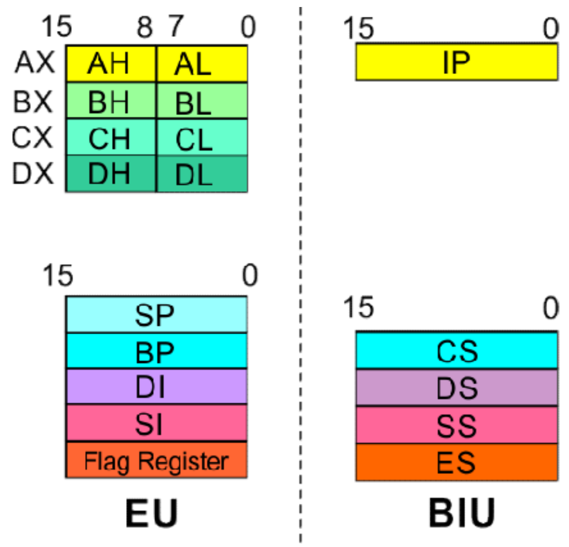
Architecture

Bus Interface Unit (BIU)

Segment Registers

Code Segment Register

- 16-bit
- CS contains the base or start of the current code segment; IP contains the distance or offset from this address to the next instruction byte to be fetched.
- BIU computes the 20-bit physical address by logically shifting the contents of CS 4-bits to the left and then adding the 16-bit contents of IP.
- That is, all instructions of a program are relative to the contents of the CS register multiplied by 16 and then offset is added provided by the IP.

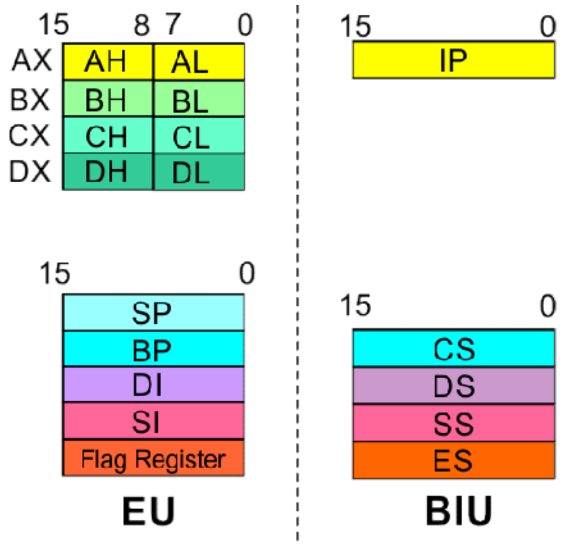


Architecture

Segment Registers

Data Segment Register

- 16-bit
- Points to the current data segment; operands for most instructions are fetched from this segment.
- The 16-bit contents of the Source Index (SI) or Destination Index (DI) or a 16-bit displacement are used as offset for computing the 20-bit physical address.



Week - 7

Architecture

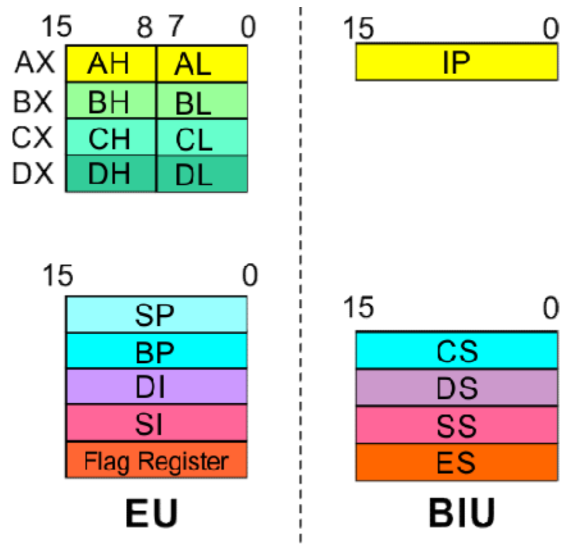
Bus Interface Unit (BIU)

Segment Registers

Stack Segment Register

- 16-bit
- Points to the current stack.
- The 20-bit physical stack address is ~~calculated from the Stack~~ Segment (SS) and the Stack Pointer (SP) for stack instructions such as PUSH and POP.

In based addressing mode, the 20-bit physical stack address is calculated from the Stack segment (SS) and the Base Pointer (BP).



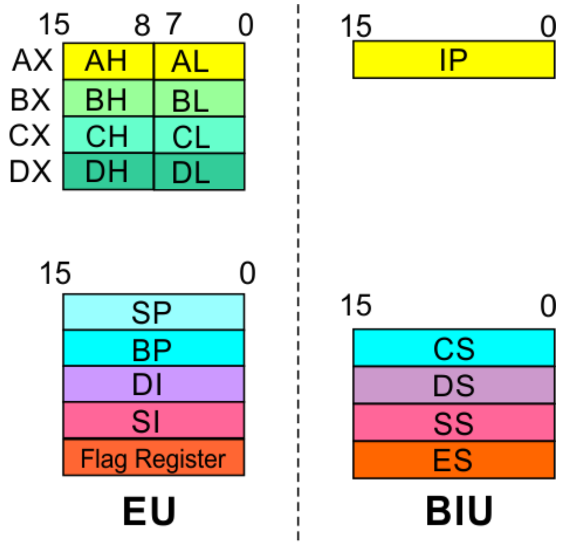
Architecture

Bus Interface Unit (BIU)

Segment Registers

Extra Segment Register

- **16-bit**
- **Points to the extra segment in which data (in excess of 64K pointed to by the DS) is stored.**
- **String instructions use the ES and DI to determine the 20-bit physical address for the destination.**



Architecture

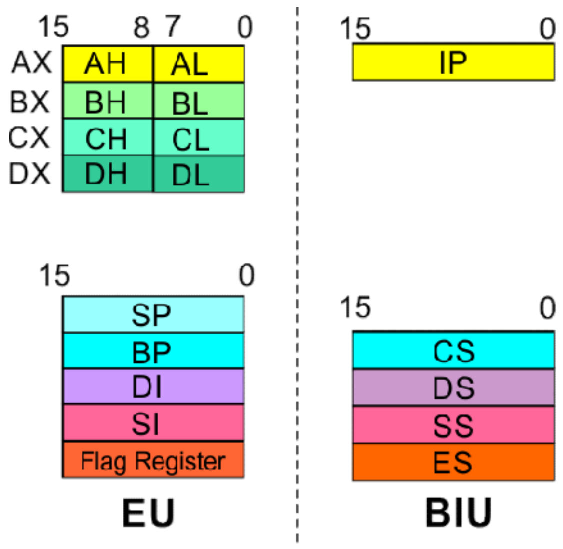
Bus Interface Unit (BIU)

Segment Registers

Instruction Pointer

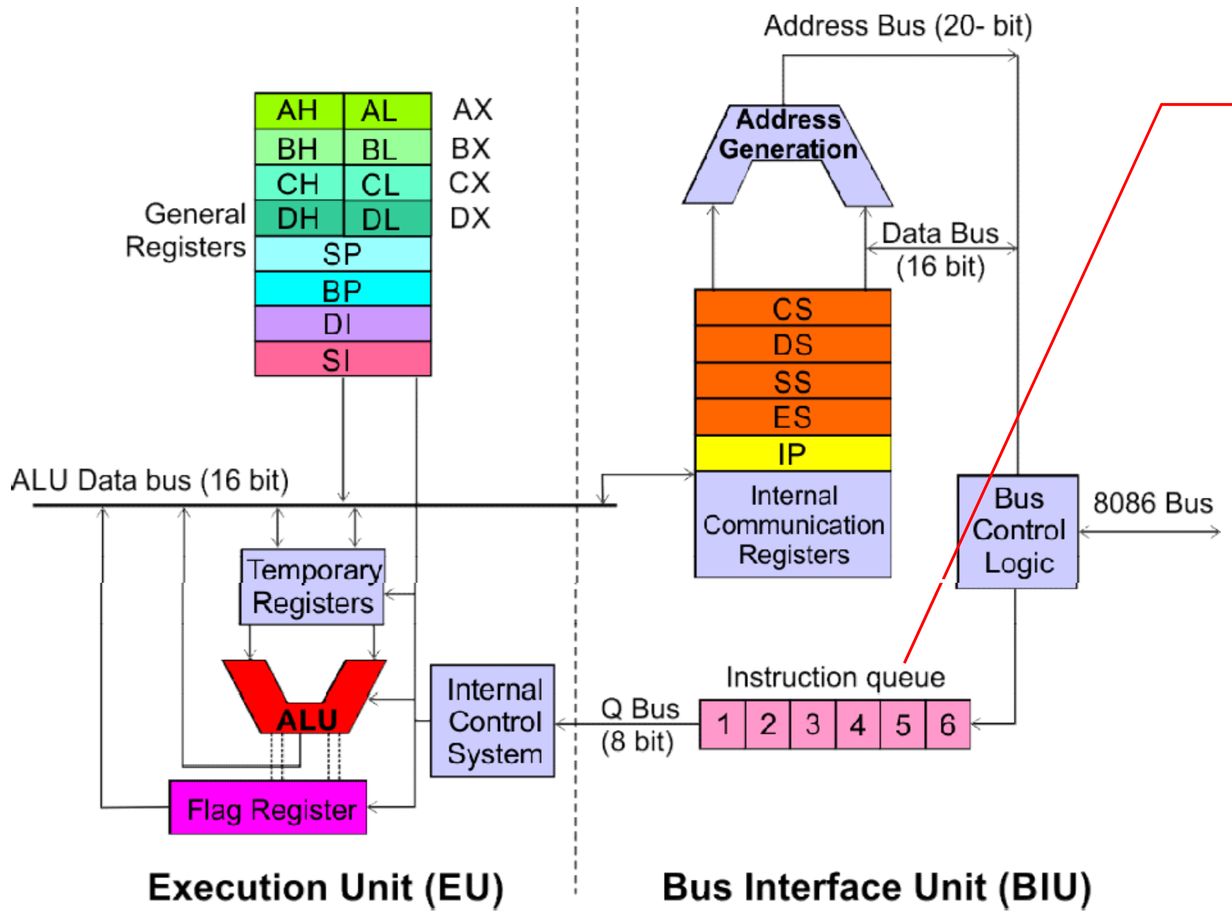
- 16-bit
- Always points to the next instruction to be executed within the currently executing code segment.
- So, this register contains the 16-bit offset address pointing to the next instruction code within the 64Kb of the code segment area.

Its content is automatically incremented as the execution of the next instruction takes place.



Architecture

Bus Interface Unit (BIU)



Instruction queue

- **A group of First-In-First-Out (FIFO) in which up to 6 bytes of instruction code are pre fetched from the memory ahead of time.**
- **This is done in order to speed up the execution by overlapping instruction fetch with execution.**
- **This mechanism is known as <u>pipelining</u>.**

Architecture

Execution Unit (EU)

EU decodes and executes instructions.

A decoder in the EU control system translates instructions.

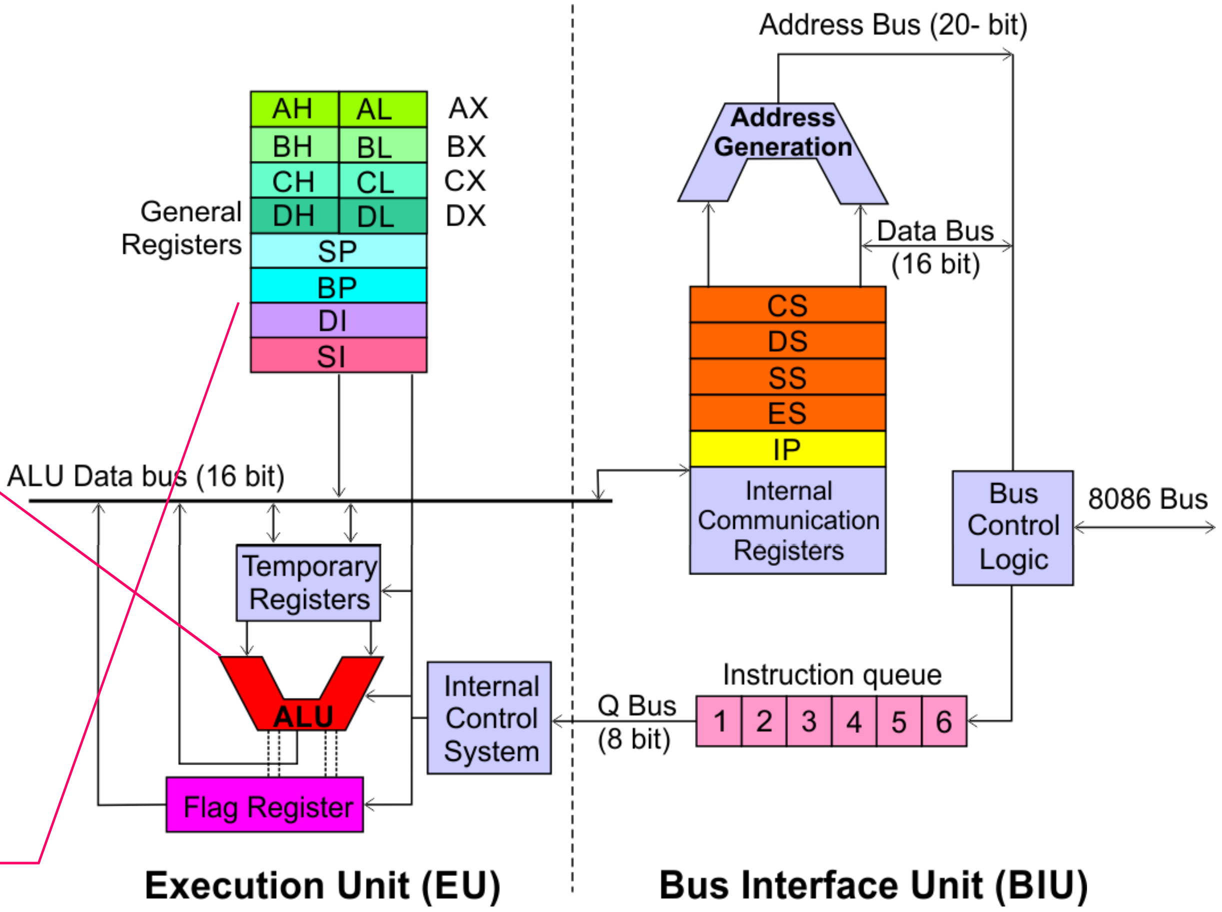
16-bit ALU for performing arithmetic and logic operation

Four general purpose registers(AX, BX, CX, DX);

Pointer registers (Stack Pointer, Base Pointer);

and

Index registers (Source Index, Destination Index) each of 16-bits



Some of the 16 bit registers can be used as two 8 bit registers as :

AX can be used as AH and AL
BX can be used as BH and BL
CX can be used as CH and CL
DX can be used as DH and DL

Architecture

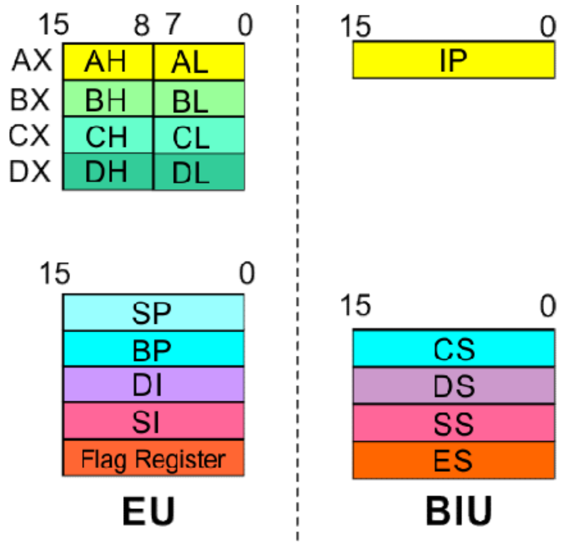
Execution Unit (EU)

EU Registers

Accumulator Register (AX)

- 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.



The I/O instructions use the AX or AL for inputting / outputting 16 or 8 bit data to or from an I/O port.

Multiplication and Division instructions also use the AX or AL.

Architecture

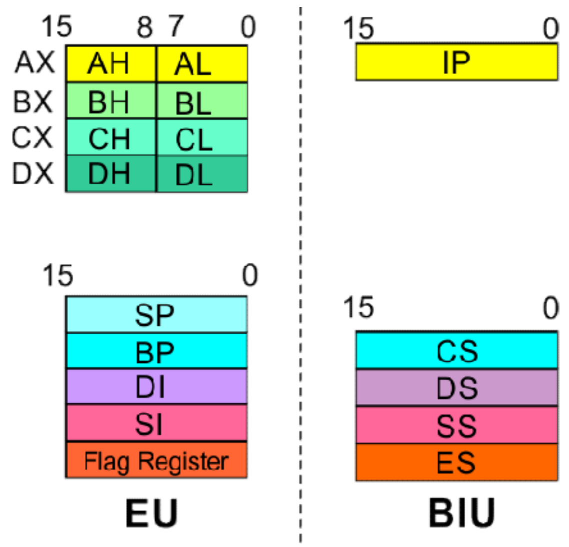
Execution Unit (EU)

EU Registers

Base Register (BX)

- 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.



This is the only general purpose register whose contents can be used for addressing the 8086 memory.

All memory references utilizing this register content for addressing use DS as the default segment register

Week - 8

Addressing Modes

- Every instruction of a program has to operate on a data.
- The different ways in which a source operand is denoted in an instruction are known as addressing modes.

1. Register Addressing 2. Immediate Addressing	Group I : Addressing modes for register and immediate data
3. Direct Addressing 4. Register Indirect Addressing 5. Based Addressing 6. Indexed Addressing 7. Based Index Addressing 8. String Addressing	Group II : Addressing modes for memory data
9. Direct I/O port Addressing 10. Indirect I/O port Addressing	Group III : Addressing modes for I/O ports
11. Relative Addressing	Group IV : Relative Addressing mode
12. Implied Addressing	Group V : Implied Addressing mode

Addressing Modes

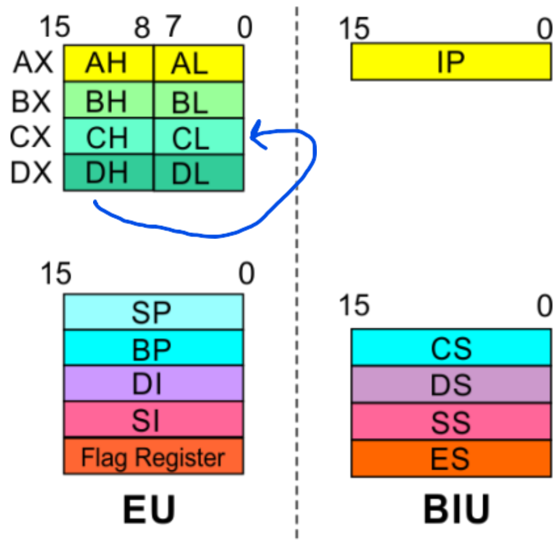
The instruction will specify the name of the register which holds the data to be operated by the instruction.

Example:

MOV CL, DH

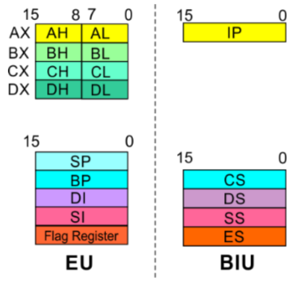
The content of 8-bit register DH is moved to another 8-bit register CL

(CL) → (DH)



Addressing Modes

1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing



In immediate addressing mode, an 8-bit or 16-bit data is specified as part of the instruction

Example:

MOV DL, 08H

The 8-bit data (08_H) given in the instruction is moved to DL

$(DL) \rightarrow 08_H$

MOV AX, 0A9FH

The 16-bit data ($0A9F_H$) given in the instruction is moved to AX register

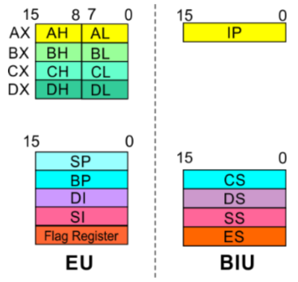
$(AX) \rightarrow 0A9F_H$

8086 Microprocessor

Addressing Modes

Group II : Addressing modes for memory data

1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Index Addressing
8. String Addressing
9. Direct I/O port Addressing
10. Indirect I/O port Addressing
11. Relative Addressing
12. Implied Addressing



Here, the effective address of the memory location at which the data operand is stored is given in the instruction.

The effective address is just a 16-bit number written directly in the instruction.

Example:

```
MOV   BX, [1354H]
MOV   BL,  [0400H]
```

Use Bx only

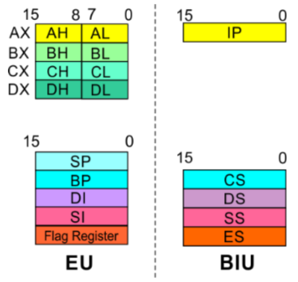
The square brackets around the 1354_H denotes the contents of the memory location. When executed, this instruction will copy the contents of the memory location into BX register.

This addressing mode is called direct because the displacement of the operand from the segment base is specified directly in the instruction.

8086 Microprocessor

Addressing Modes

Group II : Addressing modes for memory data



In Register indirect addressing, name of the register which holds the effective address (EA) will be specified in the instruction.

Registers used to hold EA are any of the following registers:

BX, BP, DI and SI.

Content of the DS register is used for base address calculation.

Example:

MOV CX, [BX]

Note : Register/ memory enclosed in brackets refer to content of register/ memory

Operations:

EA = (BX)

BA = (DS) x 16_{10}

MA = BA + EA

Used as offset of the DS

(CX) → (MA) or,

(CL) → (MA)

(CH) → (MA +1)

Week - 9

Instruction Set

8086 supports 6 types of instructions.

1. **Data Transfer Instructions**
2. **Arithmetic Instructions**
3. **Logical Instructions**
4. **String Manipulation Instructions**
5. **Process Control Instructions**
6. **Control Transfer Instructions**

Instruction Set

1. Data Transfer Instructions

Mnemonics: MOV, XCHG, PUSH, POP, IN, OUT …

MOV reg2/ mem, reg1/ mem MOV reg2, reg1 MOV mem, reg1 MOV reg2, mem	 (reg2) → (reg1) (mem) → (reg1) (reg2) → (mem)
MOV reg, data MOV mem, data	(reg) → data (mem) → data

XCHG reg2/ mem, reg1 XCHG reg2, reg1 XCHG mem, reg1	 (reg2) x (reg1) (mem) x (reg1)

8086 Microprocessor

Instruction Set

1. Data Transfer Instructions

Mnemonics: **MOV, XCHG, PUSH, POP, IN, OUT …**

IN A, [DX]	
IN AL, [DX]	**PORT$_{addr}$ = (DX)** **(AL) → (PORT)**
IN AX, [DX]	**PORT$_{addr}$ = (DX)** **(AX) → (PORT)**

OUT [DX], A	
OUT [DX], AL	**PORT$_{addr}$ = (DX)** **(PORT) → (AL)**
OUT [DX], AX	**PORT$_{addr}$ = (DX)** **(PORT) → (AX)**

IN A, addr8	
IN AL, addr8	**(AL) → (addr8)**
IN AX, addr8	**(AX) → (addr8)**

OUT addr8, A	
OUT addr8, AL	**(addr8) → (AL)**
OUT addr8, AX	**(addr8) → (AX)**

Instruction Set

2. Arithmetic Instructions

Mnemonics: **ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP…**

ADD reg2/ mem, reg1/mem ADD reg2, reg1 ADD reg2, mem ADD mem, reg1	(reg2) → (reg1) + (reg2) (reg2) → (reg2) + (mem) (mem) → (mem)+(reg1)
ADD reg, data ADD mem, data	(reg) → (reg)+ data (mem) → (mem)+data
ADD A, data ADD AL, data8 ADD AX, data16	(AL) → (AL) + data8 (AX) → (AX) +data16

Instruction Set

2. Arithmetic Instructions

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP…

ADC reg2/ mem, reg1/mem **ADC reg2, reg1** **ADC reg2, mem** **ADC mem, reg1**	 **(reg2) → (reg1) + (reg2)+CF** **(reg2) → (reg2) + (mem)+CF** **(mem) → (mem)+(reg1)+CF**
ADC reg/mem, data **ADC reg, data** **ADC mem, data**	 **(reg) → (reg)+ data+CF** **(mem) → (mem)+data+CF**
ADDC A, data **ADD AL, data8** **ADD AX, data16**	 **(AL) → (AL) + data8+CF** **(AX) → (AX) +data16+CF**

Instruction Set

2. Arithmetic Instructions

Mnemonics: **ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP…**

SUB reg2/ mem, reg1/mem **SUB reg2, reg1** **SUB reg2, mem** **SUB mem, reg1**	**(reg2) → (reg1) - (reg2)** **(reg2) → (reg2) - (mem)** **(mem) → (mem) - (reg1)**
SUB reg, data **SUB mem, data**	**(reg) → (reg) - data** **(mem) → (mem) - data**
SUB A, data **SUB AL, data8** **SUB AX, data16**	**(AL) → (AL) - data8** **(AX) → (AX) - data16**

Instruction Set

2. Arithmetic Instructions

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP…

SBB reg2/ mem, reg1/mem **SBB reg2, reg1** **SBB reg2, mem** **SBB mem, reg1**	**(reg2) → (reg1) - (reg2) - CF** **(reg2) → (reg2) - (mem)- CF** **(mem) → (mem) - (reg1) –CF**
SBB reg/mem, data **SBB reg, data** **SBB mem, data**	**(reg) → (reg) – data - CF** **(mem) → (mem) - data - CF**
SBB A, data **SBB AL, data8** **SBB AX, data16**	**(AL) → (AL) - data8 - CF** **(AX) → (AX) - data16 - CF**

Instruction Set

2. Arithmetic Instructions

Mnemonics: **ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP…**

INC reg/ mem	
INC reg8	**(reg8) → (reg8) + 1**
INC reg16	**(reg16) → (reg16) + 1**
INC mem	**(mem) → (mem) + 1**

DEC reg/ mem	
DEC reg8	**(reg8) → (reg8) - 1**
DEC reg16	**(reg16) → (reg16) - 1**
DEC mem	**(mem) → (mem) - 1**

8086 Microprocessor

Instruction Set

2. Arithmetic Instructions

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP…

Doesn't change the value of the register. result is not stored anywhere

CMP reg2/mem, reg1/ mem

CMP reg2, reg1 MOV AL, 5 MOV BL, 5 CMP AL, BL ; AL = 5, ZF = 1 (so equal!)	**Modify flags → (reg2) – (reg1)** **If (reg2) > (reg1) then CF=0, ZF=0, SF=0** **If (reg2) < (reg1) then CF=1, ZF=0, SF=1** **If (reg2) = (reg1) then CF=0, ZF=1, SF=0**
CMP reg2, mem	**Modify flags → (reg2) – (mem)** **If (reg2) > (mem) then CF=0, ZF=0, SF=0** **If (reg2) < (mem) then CF=1, ZF=0, SF=1** **If (reg2) = (mem) then CF=0, ZF=1, SF=0**
CMP mem, reg1	**Modify flags → (mem) – (reg1)** **If (mem) > (reg1) then CF=0, ZF=0, SF=0** **If (mem) < (reg1) then CF=1, ZF=0, SF=1** **If (mem) = (reg1) then CF=0, ZF=1, SF=0**

8086 Microprocessor

Instruction Set

2. Arithmetic Instructions

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP…

CMP reg/mem, data	
CMP reg, data	**Modify flags → (reg) – (data)** **If (reg) > data then CF=0, ZF=0, SF=0** **If (reg) < data then CF=1, ZF=0, SF=1** **If (reg) = data then CF=0, ZF=1, SF=0**
CMP mem, data	**Modify flags → (mem) – ~~(mem)~~** **If (mem) > data then CF=0, ZF=0, SF=0** **If (mem) < data then CF=1, ZF=0, SF=1** **If (mem) = data then CF=0, ZF=1, SF=0**

Instruction Set

2. Arithmetic Instructions

Mnemonics: ADD, ADC, SUB, SBB, INC, DEC, MUL, DIV, CMP…

CMP A, data	
CMP AL, data8	Modify flags → (AL) − data8 If (AL) > data8 then CF=0, ZF=0, SF=0 If (AL) < data8 then CF=1, ZF=0, SF=1 If (AL) = data8 then CF=0, ZF=1, SF=0
CMP AX, data16	Modify flags → (AX) − data16 If (AX) > data16 then CF=0, ZF=0, SF=0 If (mem) < data16 then CF=1, ZF=0, SF=1 If (mem) = data16 then CF=0, ZF=1, SF=0

Week - 10

Instruction Set

3. Logical Instructions

Mnemonics: AND, OR, XOR, TEST, SHR, SHL, RCR, ROL …

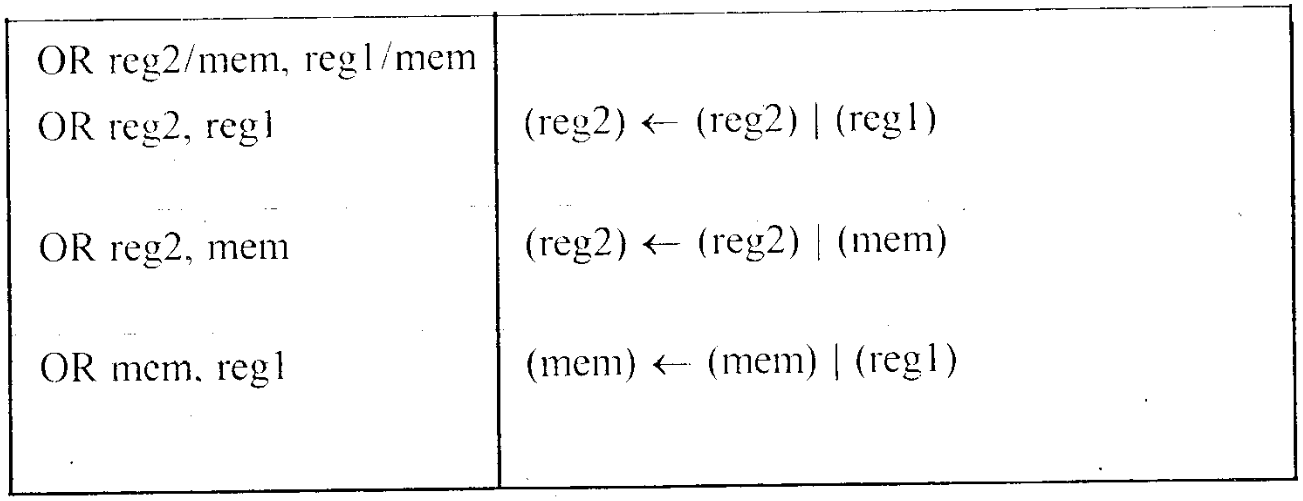
AND A, data AND AL, data8	(AL) ← (AL) & data8
AND AX, data16	(AX) ← (AX) & data16

AND reg/mem, data AND reg, data	(reg) ← (reg) & data
AND mem, data	(mem) ← (mem) & data

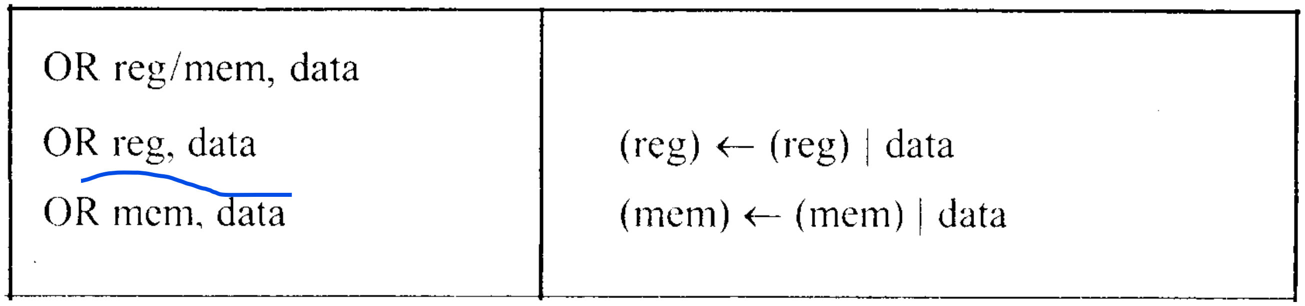
Instruction Set

3. Logical Instructions

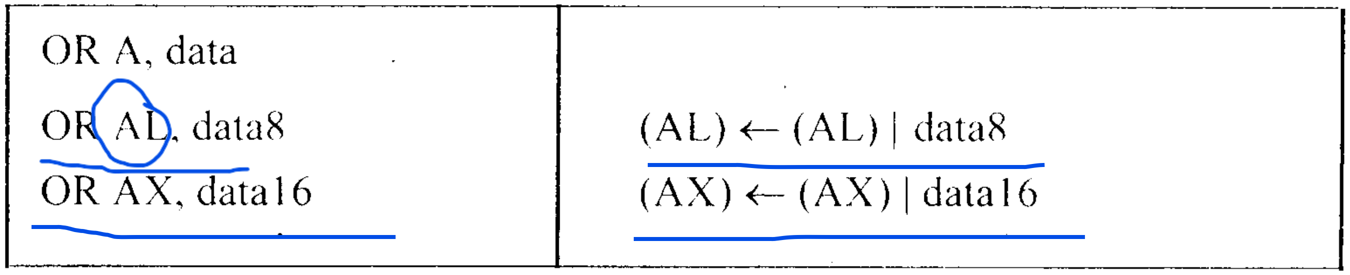
Mnemonics: **AND, OR, XOR, TEST, SHR, SHL, RCR, ROL …**



OR reg2/mem, reg1/mem	
OR reg2, reg1	(reg2) ← (reg2) | (reg1)
OR reg2, mem	(reg2) ← (reg2) | (mem)
OR mem, reg1	(mem) ← (mem) | (reg1)



OR reg/mem, data	
OR reg, data	(reg) ← (reg) | data
OR mem, data	(mem) ← (mem) | data

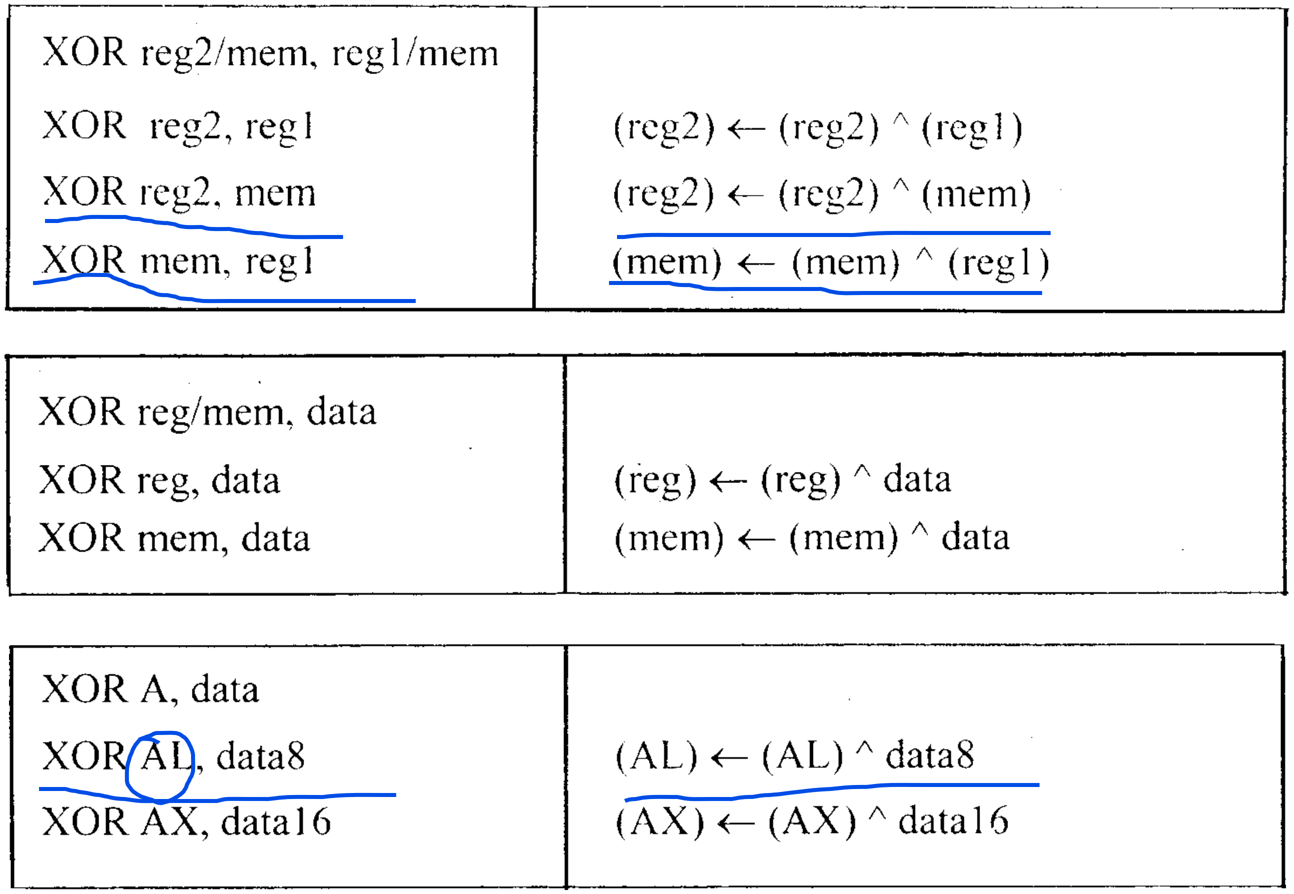


OR A, data	
OR AL, data8	(AL) ← (AL) | data8
OR AX, data16	(AX) ← (AX) | data16

Instruction Set

3. Logical Instructions

Mnemonics: **AND, OR, XOR, TEST, SHR, SHL, RCR, ROL …**



XOR reg2/mem, reg1/mem	
XOR reg2, reg1	(reg2) ← (reg2) ^ (reg1)
XOR reg2, mem	(reg2) ← (reg2) ^ (mem)
XOR mem, reg1	(mem) ← (mem) ^ (reg1)

XOR reg/mem, data	
XOR reg, data	(reg) ← (reg) ^ data
XOR mem, data	(mem) ← (mem) ^ data

XOR A, data	
XOR AL, data8	(AL) ← (AL) ^ data8
XOR AX, data16	(AX) ← (AX) ^ data16

Basic Interfacing Concept

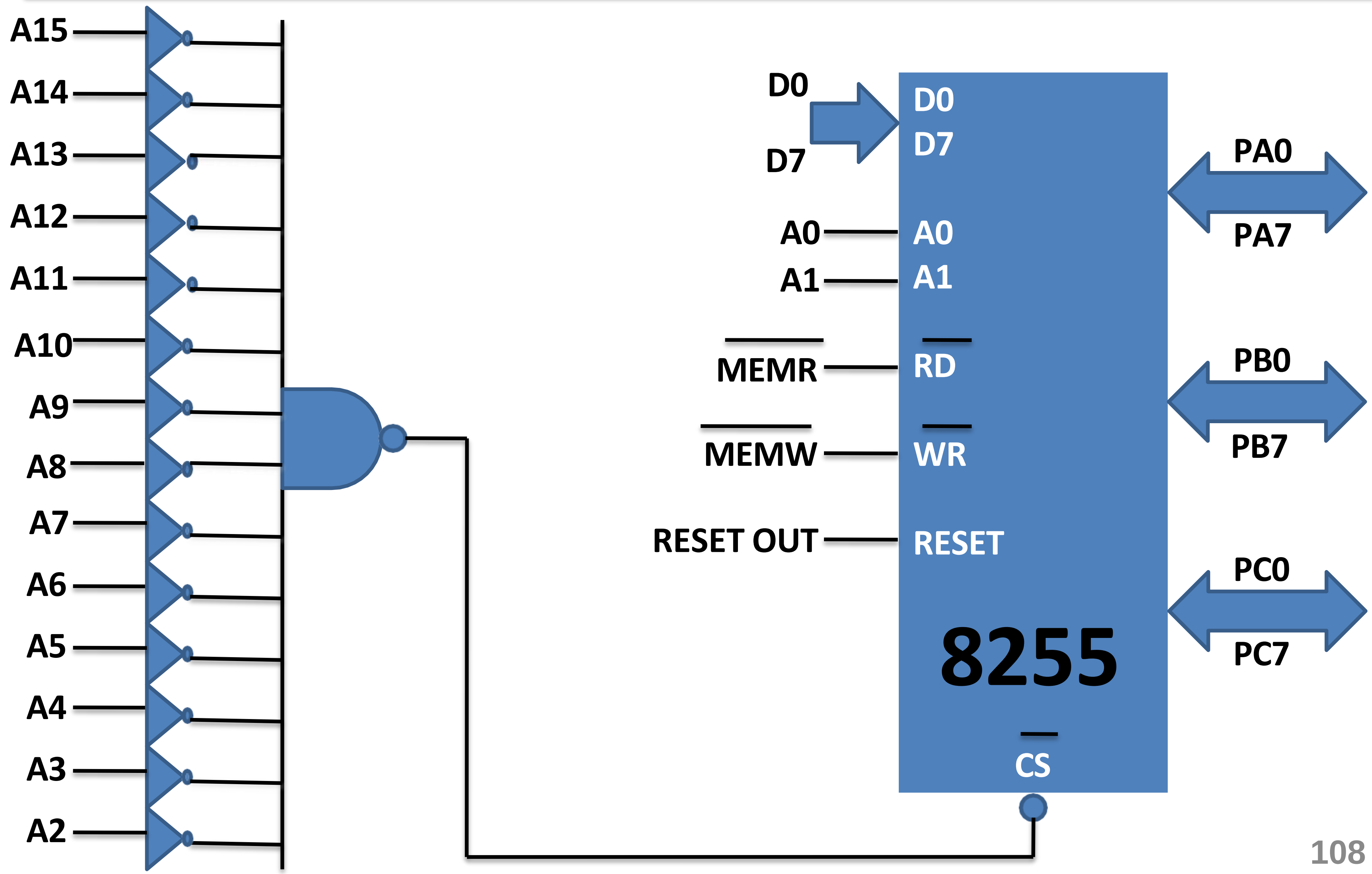
- **Any application of Microprocessor Based system Requires the transfer of data between external circuitry to the Microprocessor and Microprocessor to the External circuitry. User can give information (i.e. input) to the Microprocessor using keyboard and user can see the result or output information from the Microprocessor with the help of display.**

- **Hence interfacing is used to exchange information between two different applications/devices.**

Memory Mapped I/O

- Device address is of 16 Bit. means A_0 to A_{15} lines are used to generate device address.
- $\overline{\text{MEMR}}$ and $\overline{\text{MEMW}}$ control signals are used to control read and write I/O operations.
- Data transfer is between Any register and I/O device.
- Maximum number of I/O devices are 65536.
- Decoding 16 bit address may requires more hardware.
- For e.g. MOV R M, ADD M,CMP M etc.

INTERFACING IN MEMORY MAPPED I/O

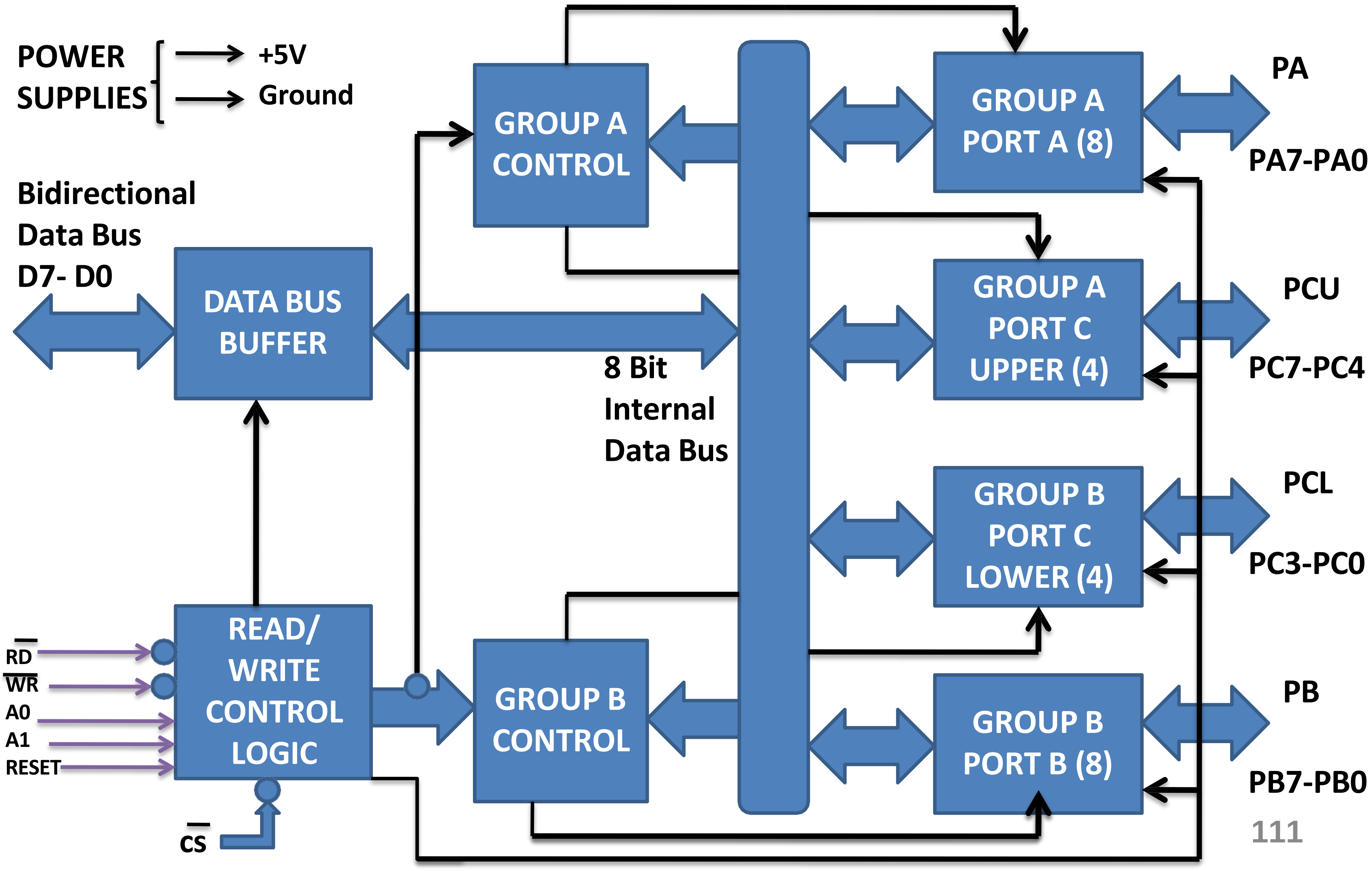


8255 PPI

- **The INTEL 8255 is a 40 pin IC having total 24 I/O pins. consisting of 3 numbers of 8 –bit parallel I/O ports (i.e. PORT A, PORT B,PORT C). The ports can be programmed to function either as a input port or as a output port in different operating modes. It requires 4 internal addresses and has one logic LOW chip select pin. Its main functions are to interface peripheral devices to the microprocessor. Basically used for parallel data transfer. operates in mainly two modes.**
- **(1) Bit Set Reset Mode (BSR Mode).**
- **(2) I/O Mode.**

Week - 11

Block Diagram of 8255 PPI

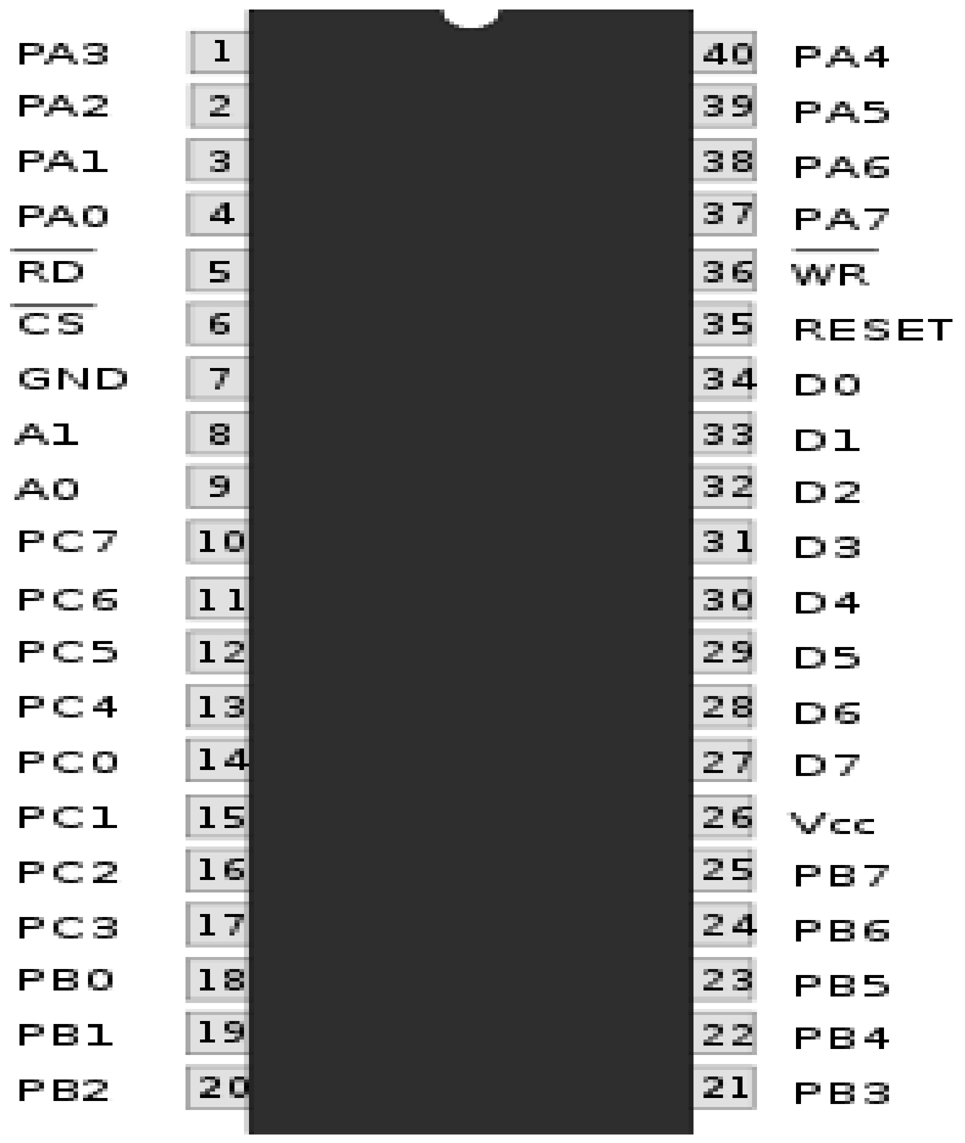


Function of Blocks

BLOCK	FUNCTION OF BLOCK
Data Bus Buffer	It is used to interface the internal data bus of 8255 to the system data bus by reading and writing operations.
Read/write Control logic	It accepts the input from the address bus and issues commands to the individual group blocks. also issues appropriate enabling signals to access the required data/control words/status words.
Port A	It can be programmed in three modes Mode0, Mode1 and Mode2.
Port B	It can be programmed in three modes Mode0 and Mode1.
Port C	It can be programmed for Bit Set/reset operation.

Pin Diagram of 8255 PPI

8255 Pin Diagram



Function of Pins

PIN	FUNCTION OF PIN
D0-D7 (Data Bus)	These are bidirectional, tri-state data bus lines are connected to the system data bus. They are used to transfer data and control word from microprocessor (8085) to 8255 or receive data or status word from 8255 to the 8085.
PA0-PA7 (Port A)	These are 8 Bit bidirectional I/O pins used to send data to output device and to receive data from input device. It functions as an 8 Bit data output latch/buffer when used in output mode and as an 8 Bit data input latch/buffer when used in input mode.
PB0-PB7 (Port B)	These are 8 Bit bidirectional I/O pins used to send data to output device and to receive data from input device. It functions as an 8 Bit data output latch/buffer when used in output mode and as an 8 Bit data input latch/buffer when used in input mode.

Function of Pins

PIN	FUNCTION OF PIN
PC0-PC7 (Port C)	These are 8 bit bidirectional I/O pins divided into two groups PCL (PC3-PC) and PCU (PC7-PC4).these groups can individually transfer data in or out when programmed for simple I/O, and used as handshake signals when programmed for handshake or bidirectional modes.
$\overline{\text{RD}}$	When this pin is low, the CPU can read data in the ports or the status word through the data bus buffer.
$\overline{\text{WR}}$	When this pin is low, the CPU can write data on the ports or in the control register through the data bus buffer.
$\overline{\text{CS}}$	This pin can be enabled for data transfer operation between the CPU and 8255.
RESET	This pin is used to reset 8255.i.e control register gets cleared and all the ports are set to the input mode.

Operating Modes Of 8255

- **There are two main operational modes of 8255:**

(1) Input/output mode,

(2) Bit set/reset mode (BSR Mode).

I/O mode again classified into three types

- **(1) Mode 0,**
- **(2) Mode 1,**
- **(3) Mode 2.**

MODE 0

- In this mode, the ports can be used for simple input/output operations without handshaking.
- If both port A and B are initialized in mode 0, the two halves of port C can be either used together as an additional 8-bit port, or they can be used as individual 4-bit ports.
- Since the two halves of port C are independent, they may be used such that one-half is initialized as an input port while the other half is initialized as an output port.

The mode 0 has following features:

- O/p are latched.
- I/p are buffered not latched.
- Port do not have handshake or interrupt capability.

MODE 1

- **When we wish to use port A or port B for handshake (strobed) input or output operation, we initialize that port in mode 1.**
- **For port B in this mode (irrespective of whether is acting as an input port or output port), PC0, PC1 and PC2 pins function as handshake lines.**

The mode 1 has following features:

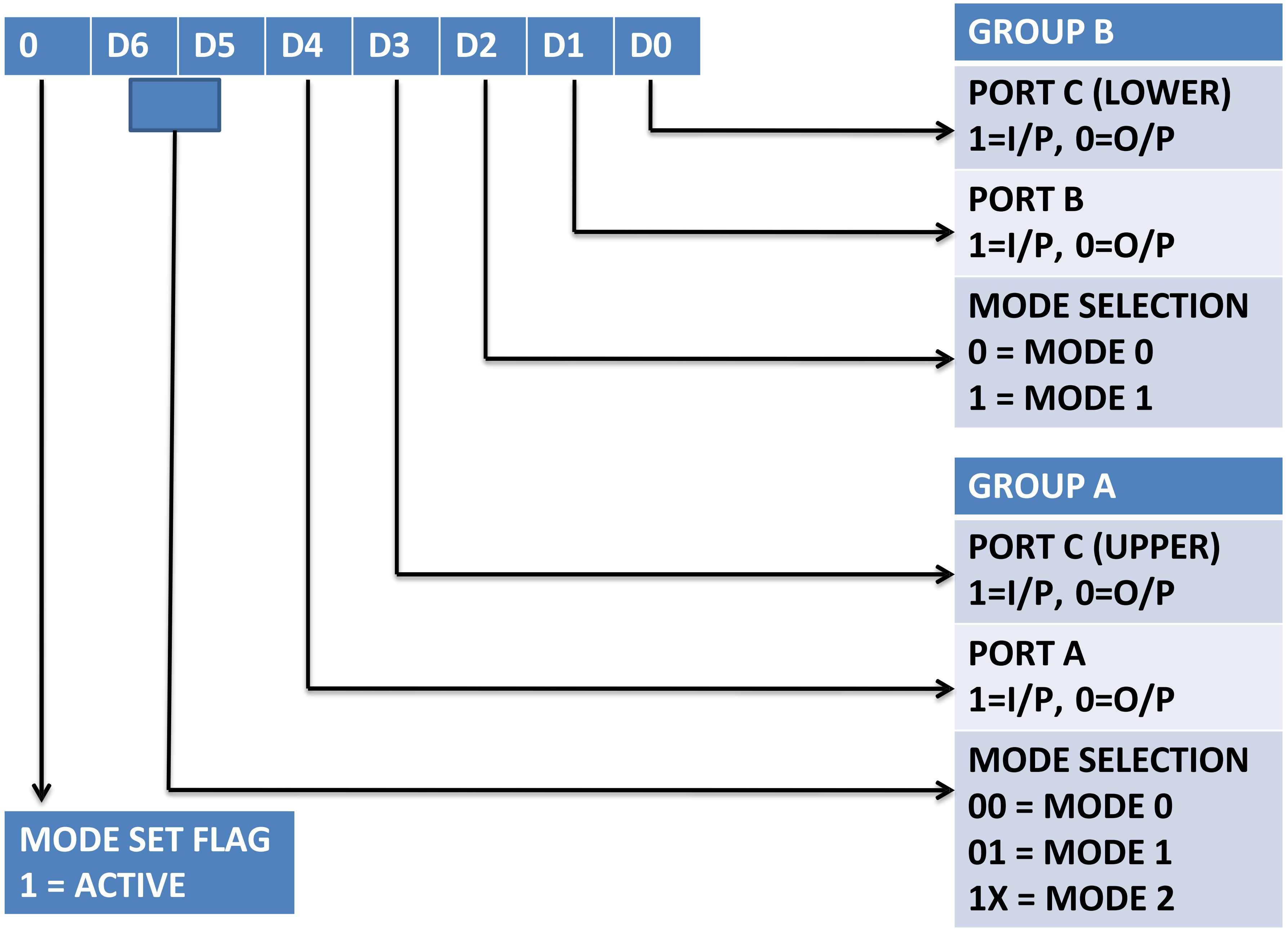
- **Two ports i.e. port A and B can be use as 8-bit i/o port.**
- **Each port uses three lines of port c as handshake signal and remaining two signals can be function as i/o port.**
- **Interrupt logic is supported.**
- **Input and Output data are latched.**

MODE 2

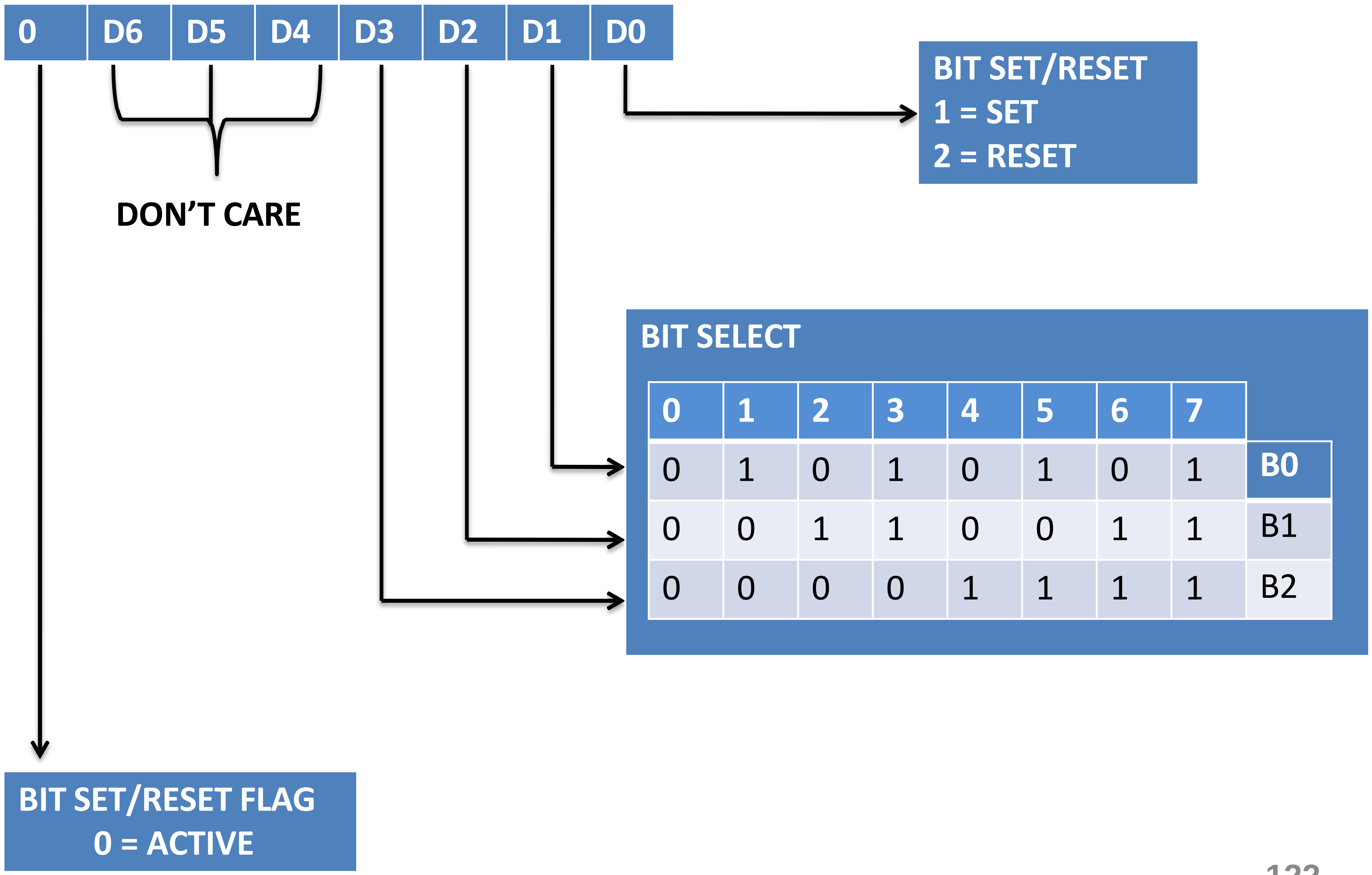
- Only group A can be initialized in this mode.
- Port A can be used for *bidirectional handshake* data transfer. This means that data can be input or output on the same eight lines (PA0 - PA7).
- Pins PC3 - PC7 are used as handshake lines for port A.
- The remaining pins of port C (PC0 - PC2) can be used as input/output lines if group B is initialized in mode 0.
- In this mode, the 8255 may be used to extend the system bus to a slave microprocessor.

Week - 12

Control Word Format in I/O Mode



Control Word Format in BSR Mode



BIT SELECT

	0	1	2	3	4	5	6	7
B0	0	1	0	1	0	1	0	1
B1	0	0	1	1	0	0	1	1
B2	0	0	0	0	1	1	1	1

Write a program to initialize 8255 in the configuration below.(assume address of the CW register as 83H).

(1) Port A: simple input (2) Port B: simple output
(3) Port CL: output (4)Port CU: input

- **Solution:**

1	0	0	1	1	0	0	0

= 98H

Program:

```
MVI A,98H  ; LOAD CONTROL WORD

OUT 83H    ; SEND CONTROL WORD
```

Write a program to initialize 8255 in the configuration below.(assume address of the CW register as 23H).

(1) Port A: output with handshake

(2) Port B: input with handshake

(3) Port CL: output (4)Port CU: input

- **Solution:**

1	0	1	0	1	1	1	0

= AEH

Program:

```
MVI A,AEH   ; LOAD CONTROL WORD

OUT 23H     ; SEND CONTROL WORD
```

Find the control word for the register arrangement of the ports of intel 8255 for mode 0 operation.

- Port A: Output, Port B: Output,
- Port CU: Output, Port CL: Output

Solution:

1	0	0	0	0	0	0	0

= 80H

The control word register for the above ports of intel 8255 is 80H.

Find the control word for the register arrangement of the ports of intel 8255 for mode 0 operation.

- Port A: Input, Port B: Input,
- Port CU: Input, Port CL: Input

<u>Solution</u>:

1	0	0	1	1	0	1	1

= 9BH

The control word register for the above ports of intel 8255 is 9BH.

Week - 13

8259 Programmable Interrupt Controller (PIC)

8259 Programmable Interrupt Controller (PIC)

1. This IC is designed to simplify the implementation of the interrupt interface in the 8088 and 8086 based microcomputer systems.
2. This device is known as a ‘Programmable Interrupt Controller’ or PIC.
3. It is manufactured using the NMOS technology and It is available in 28-pin DIP.
4. The operation of the PIC is programmable under software control (Programmable)and it can be configured for a wide variety of applications.
5. 8259A is treated as peripheral in a microcomputer system.
6. 8259A PIC adds eight vectored priority encoded interrupts to the microprocessor.
7. This controller can be expanded without additional hardware to accept up to 64 interrupt request inputs. This expansion required a master 8259A and eight 8259A slaves.
8. Some of its programmable features are:

    · The ability to accept level-triggered or edge-triggered inputs.

    · The ability to be easily cascaded to expand from 8 to 64 interrupt-inputs.

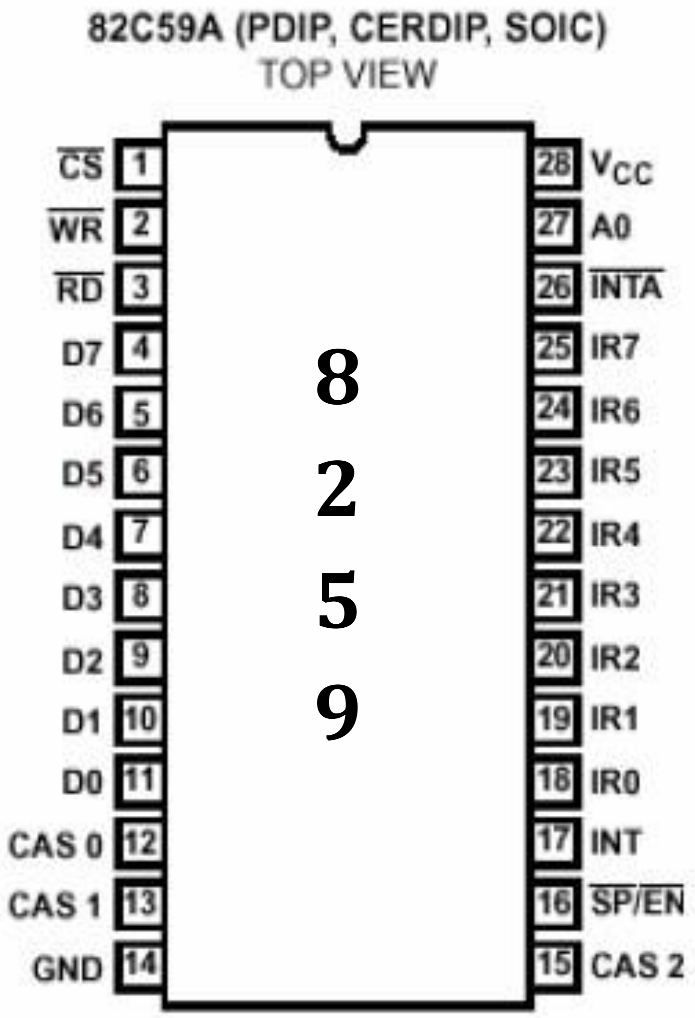
    · Its ability to be configured to implement a wide variety of priority schemes.

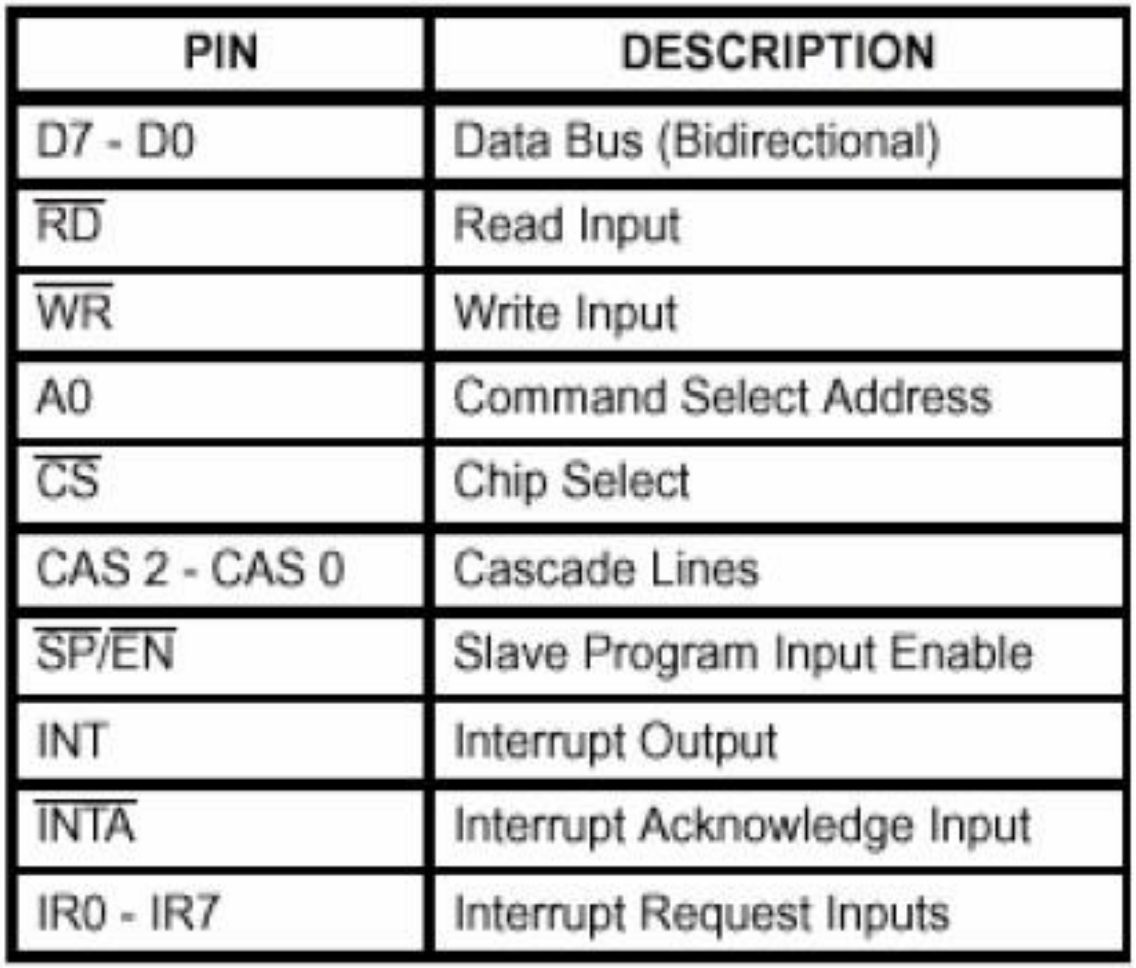
ASSINGMENT OF SIGNALS FOR 8259:

1. **D7- D0** is connected to microprocessor data bus D7-D0 (AD7-AD0).
2. **IR7- IR0**, Interrupt Request inputs are used to request an interrupt and to connect to a slave in a system with multiple 8259As.
3. $\overline{\textbf{WR}}$ **-** the write input connects to write strobe signal of microprocessor.
4. $\overline{\textbf{RD}}$ **-** the read input connects to the IORC signal.
5. **INT -** the interrupt output connects to the INTR pin on the microprocessor from the master, and is connected to a master IR pin on a slave.
6. **I**$\overline{\textbf{NTA}}$ **-** the interrupt acknowledge is an input that connects to the INTA signal on the system. In a system with a master and slaves, only the master INTA signal is connected.
7. **A0** - this address input selects different command words within the 8259A.
8. $\overline{\textbf{CS}}$ - chip select enables the 8259A for programming and control.
9. $\overline{\textbf{SP}}$**/**$\overline{\textbf{EN}}$ **-** Slave Program/Enable Buffer is a dual-function pin.

- ❖ When the 8259A is in buffered mode, this pin is an output that controls the data bus transceivers in a large microprocessor-based system.
- ❖ When the 8259A is not in buffered mode, this pin programs the device as a master (1) or a slave (0).
- ❖ CAS2-CAS0, the cascade lines are used as outputs from the master to the slaves for cascading multiple 8259As in a system.

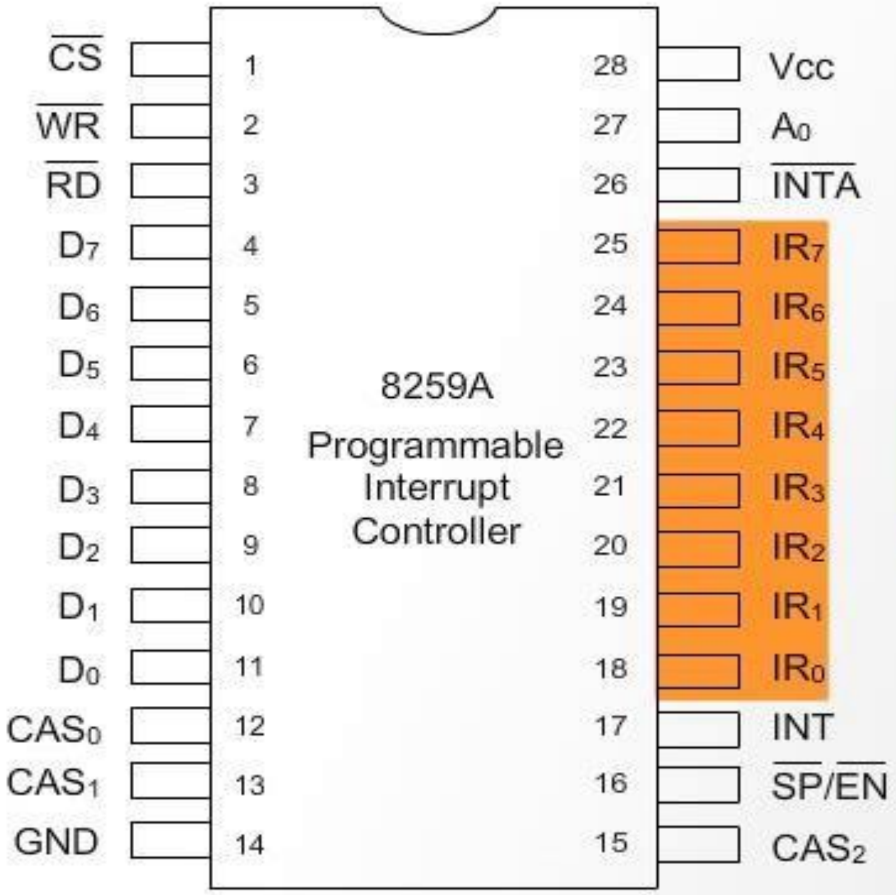
8259A PIC- PIN DIGRAM





PIN	DESCRIPTION
D7 - D0	Data Bus (Bidirectional)
$\overline{RD}$	Read Input
$\overline{WR}$	Write Input
A0	Command Select Address
$\overline{CS}$	Chip Select
CAS 2 - CAS 0	Cascade Lines
$\overline{SP}/\overline{EN}$	Slave Program Input Enable
INT	Interrupt Output
$\overline{INTA}$	Interrupt Acknowledge Input
IR0 - IR7	Interrupt Request Inputs

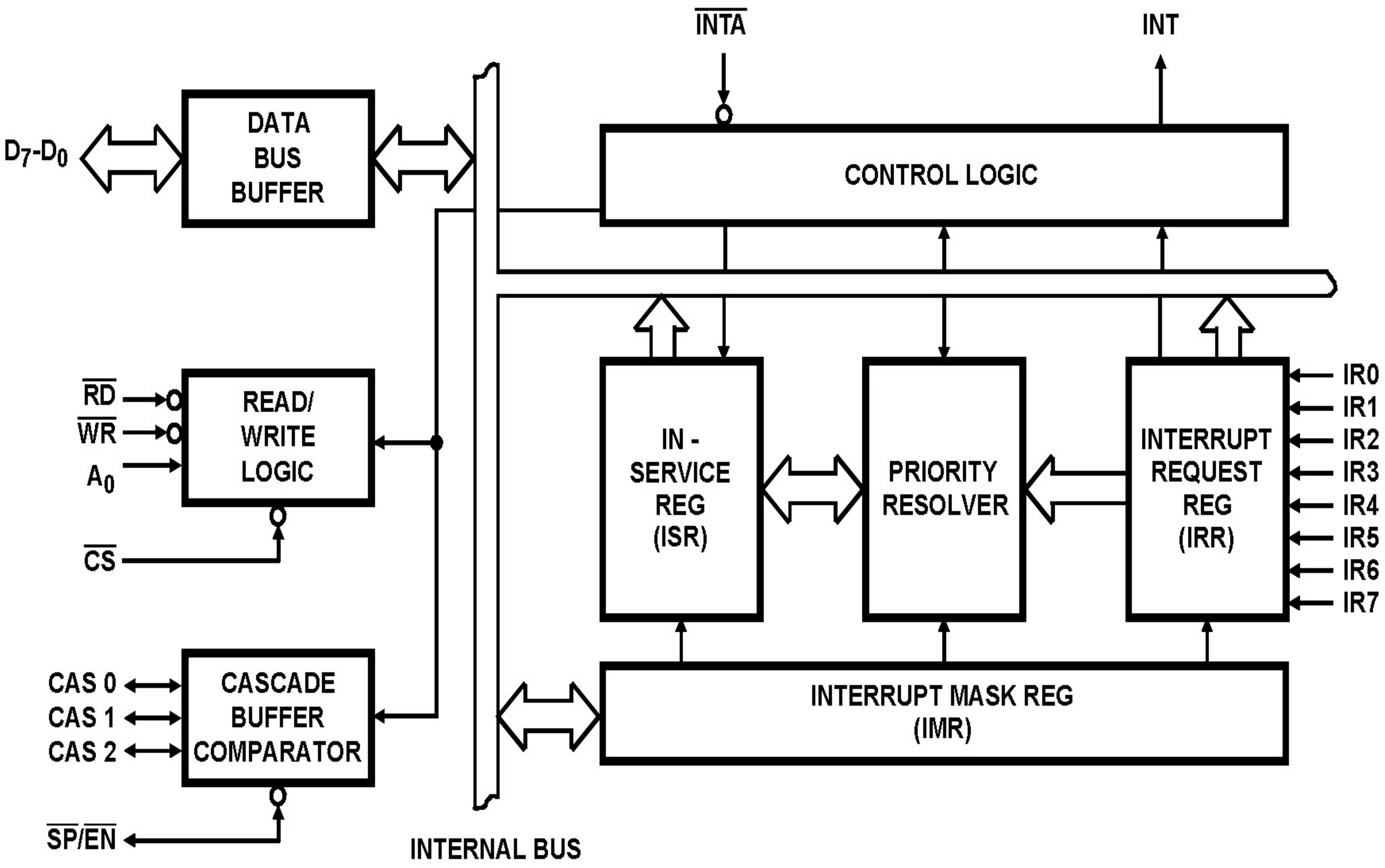
8259 PIC



Upto eight Hardware Interrupting devices are supported.

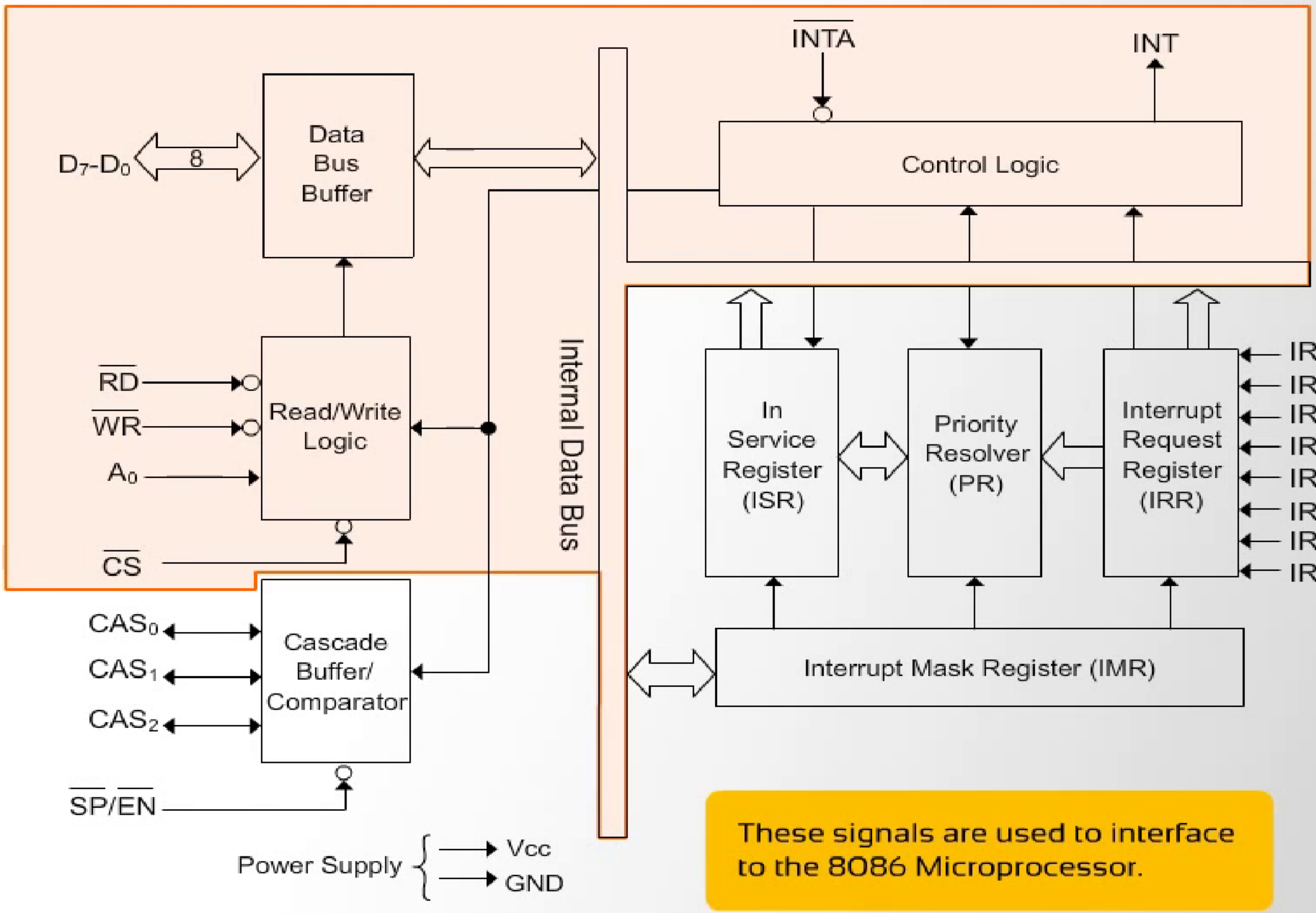
The processor is interrupted whenever the Interrupting device delivers a signal to the 8259.

8259A PIC- BLOCK DIAGRAM



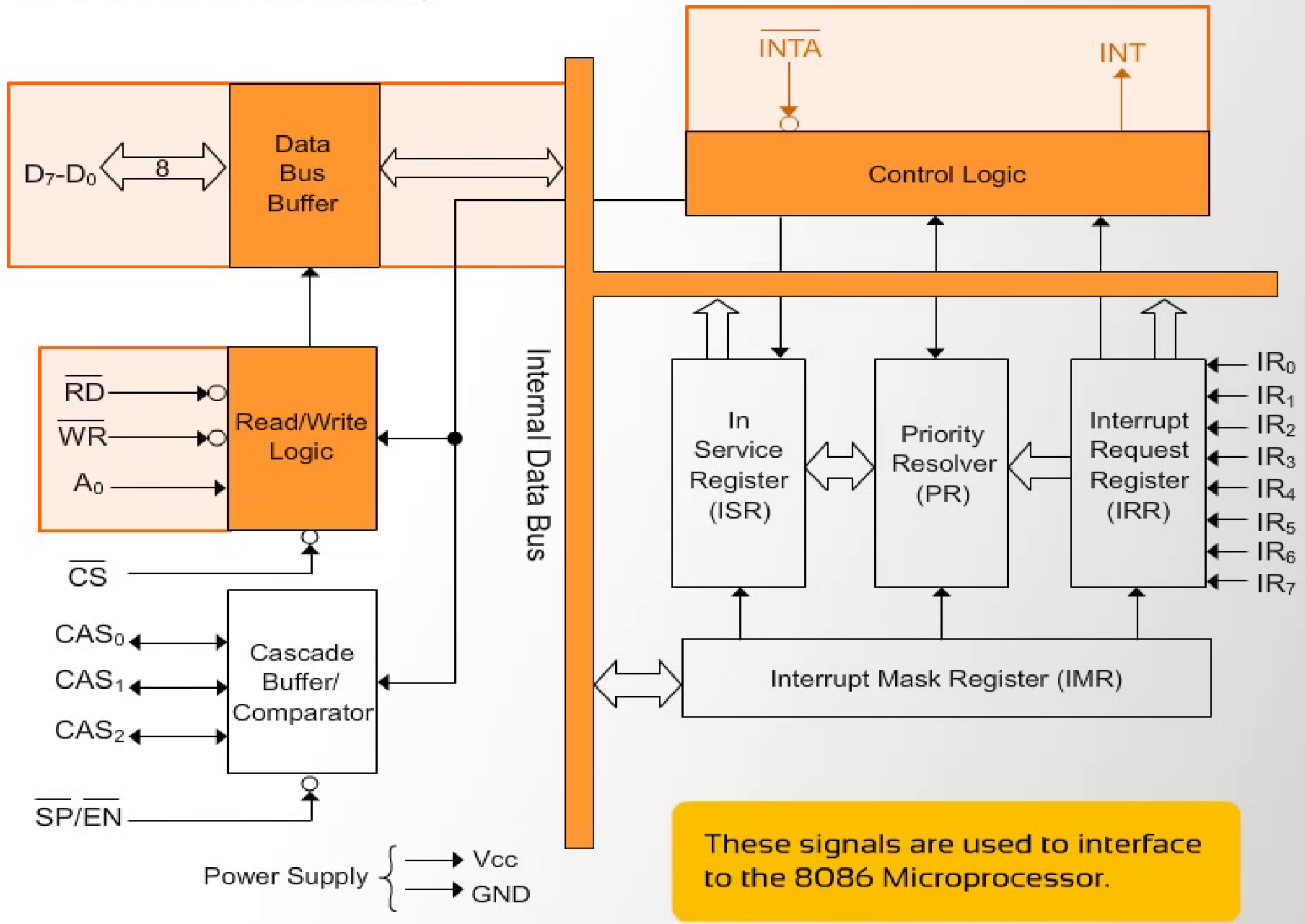
Week - 14

The Internals of 8259



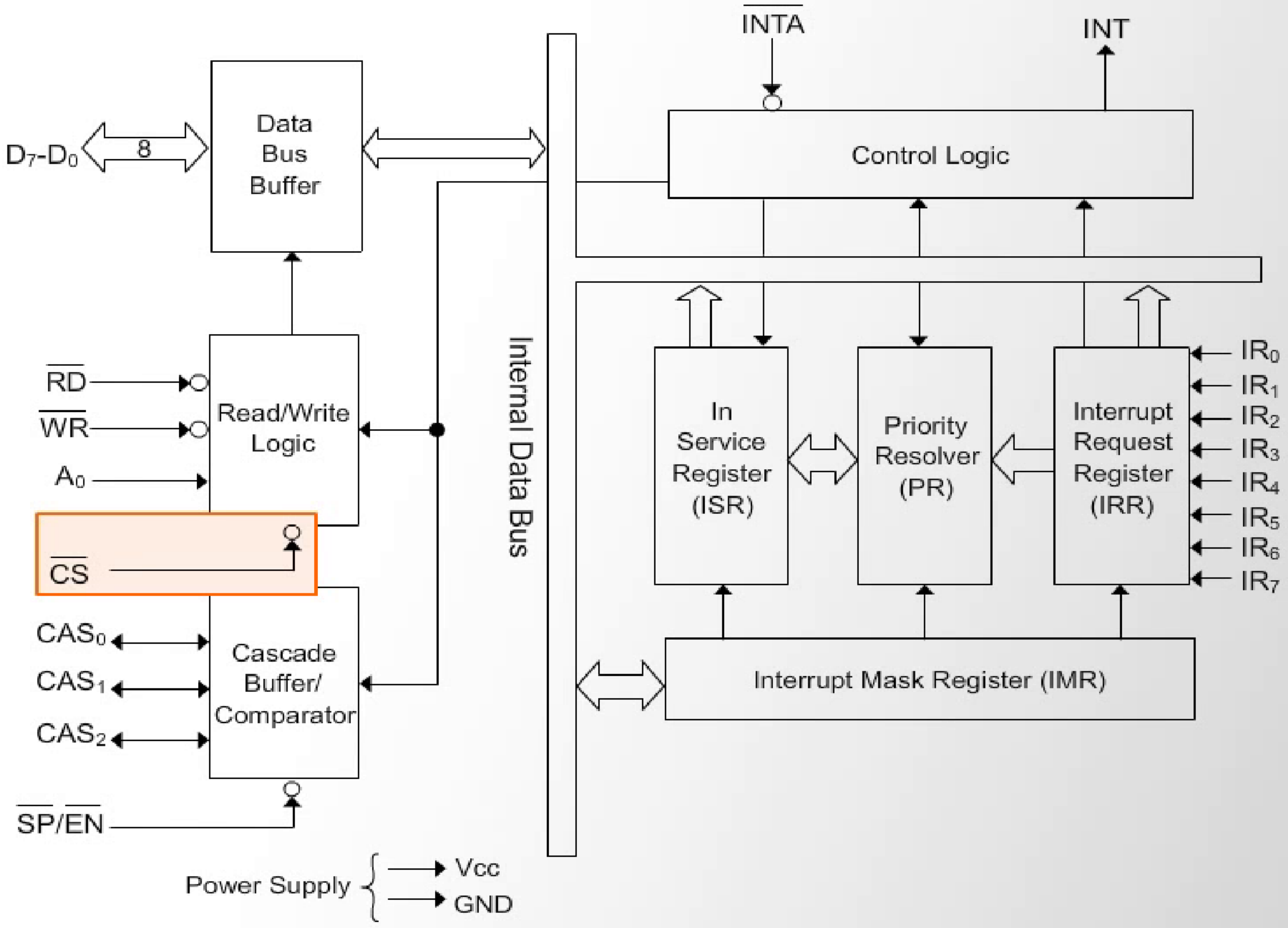
These signals are used to interface to the 8086 Microprocessor.

The Internals of 8259

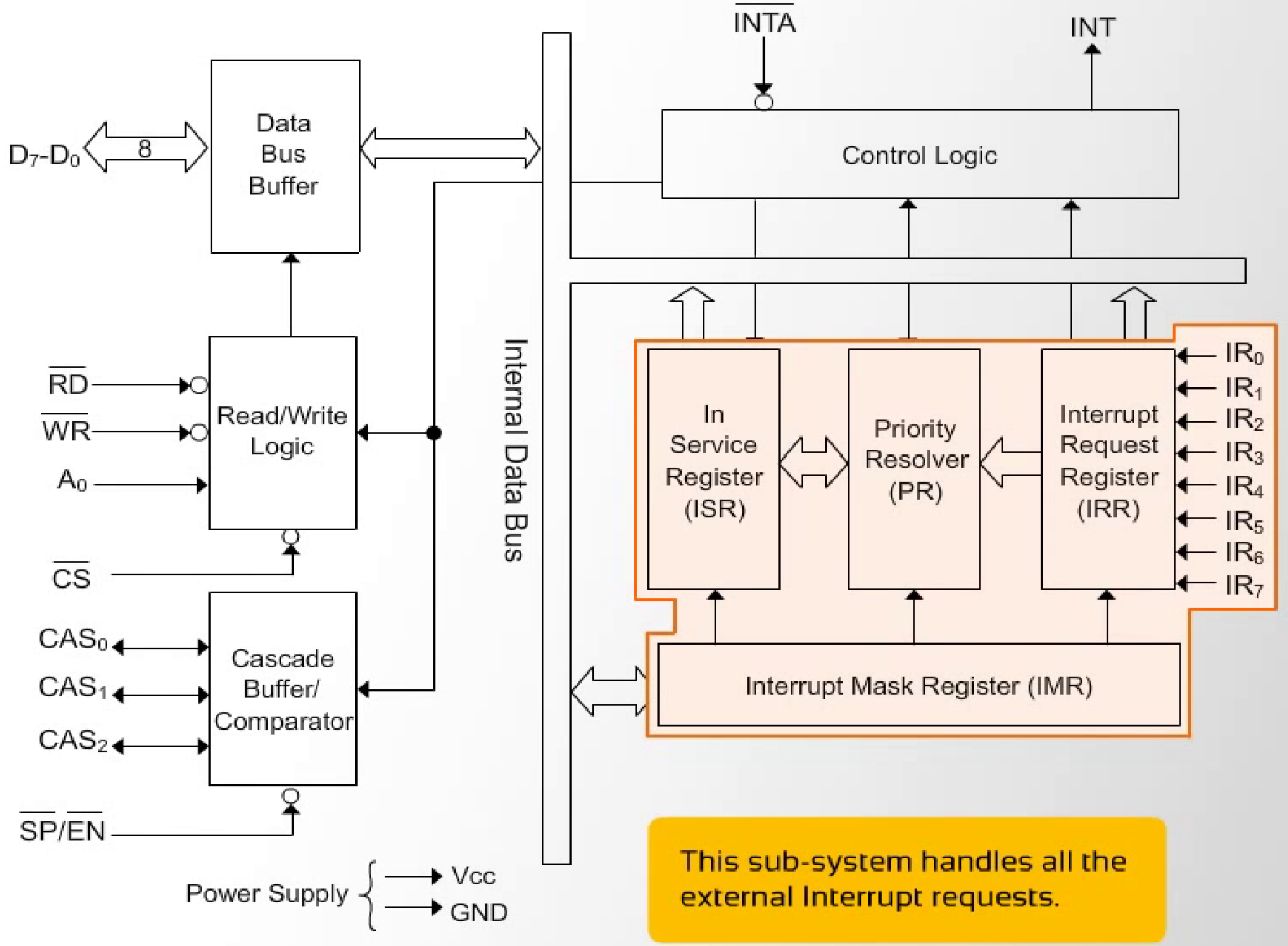


These signals are used to interface to the 8086 Microprocessor.

The Internals of 8259

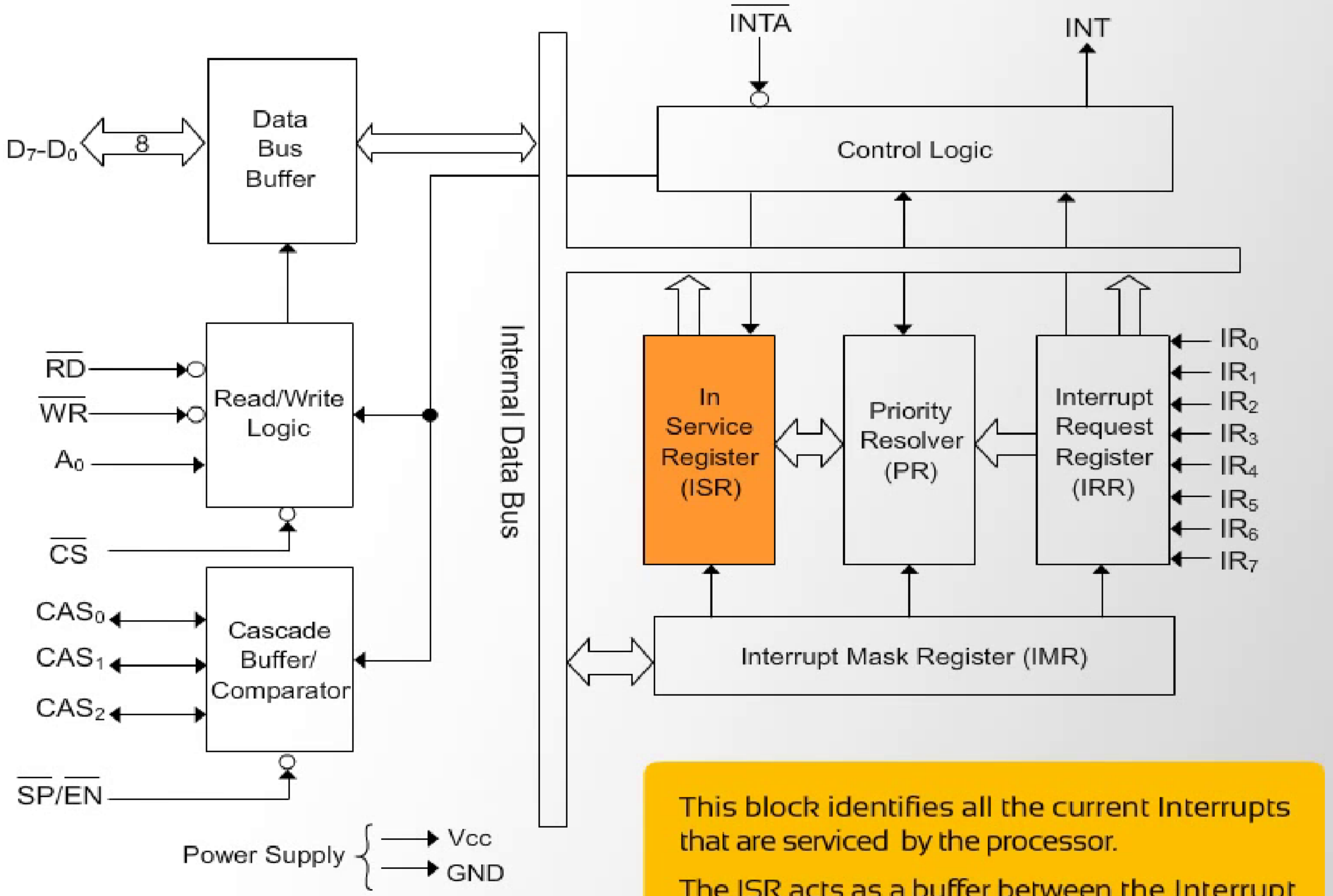


The Internals of 8259



This sub-system handles all the external Interrupt requests.

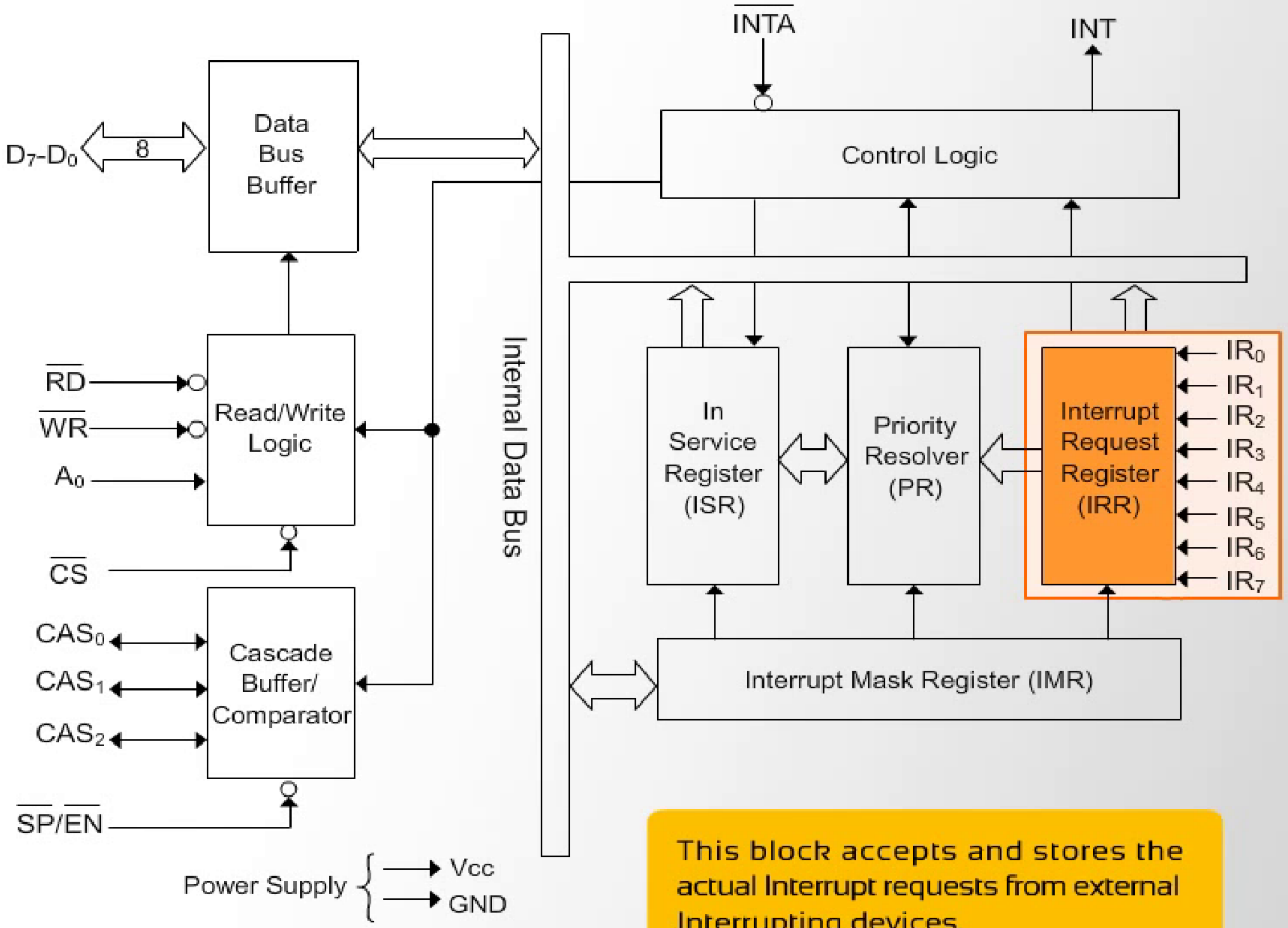
The Internals of 8259



This block identifies all the current Interrupts that are serviced by the processor.

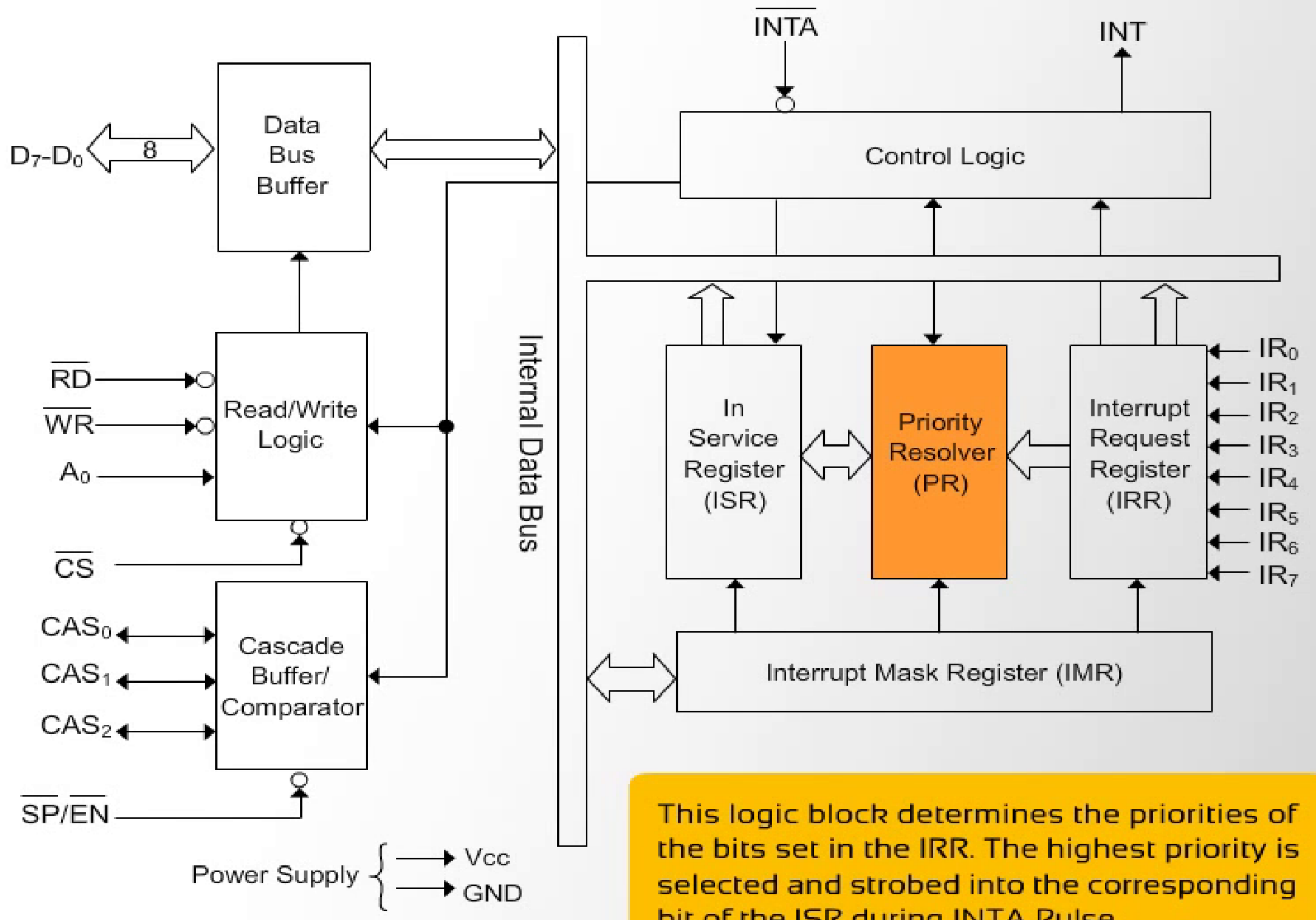
The ISR acts as a buffer between the Interrupt Request Register and the 8086.

The Internals of 8259



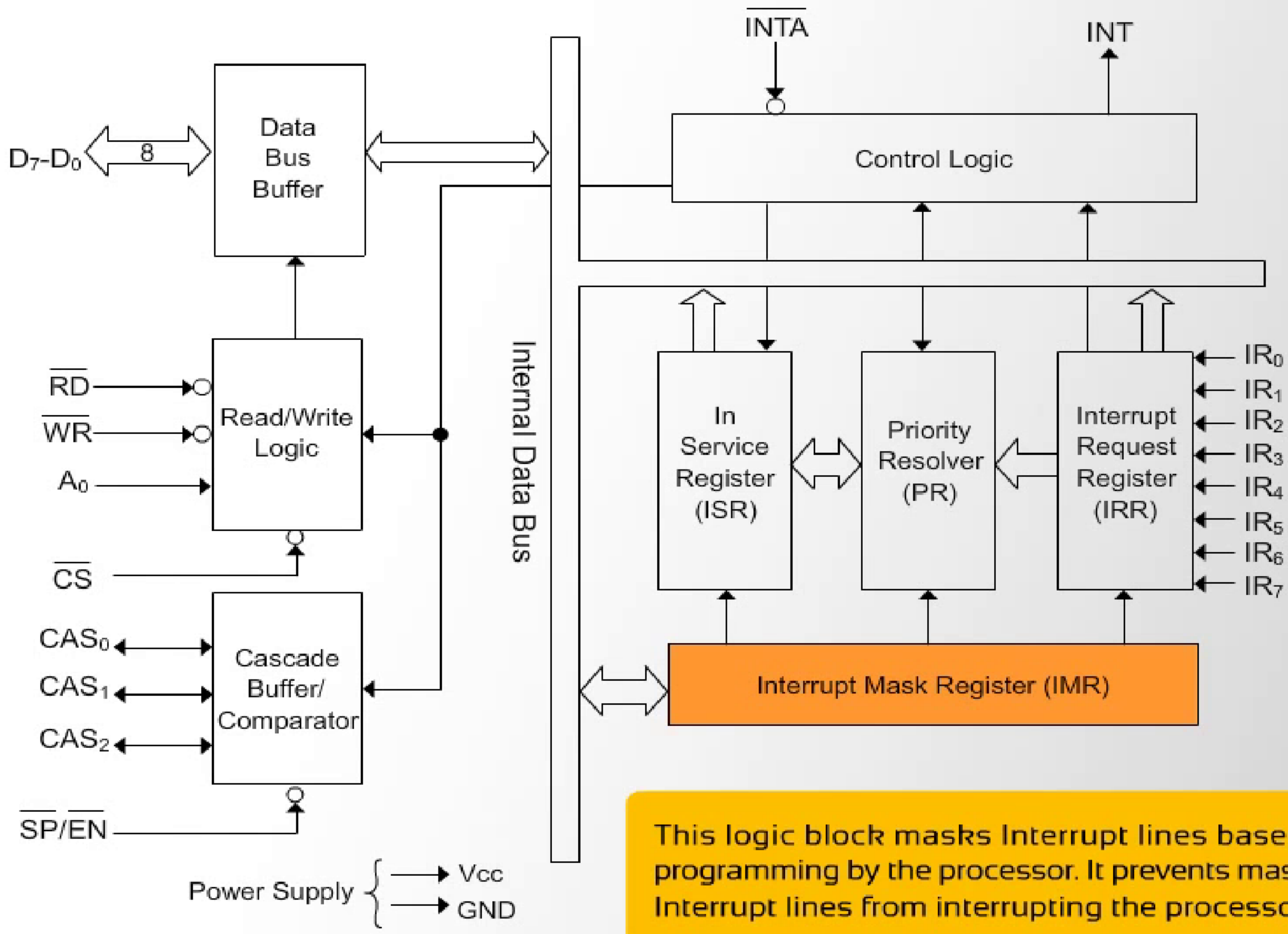
This block accepts and stores the actual Interrupt requests from external Interrupting devices.

The Internals of 8259



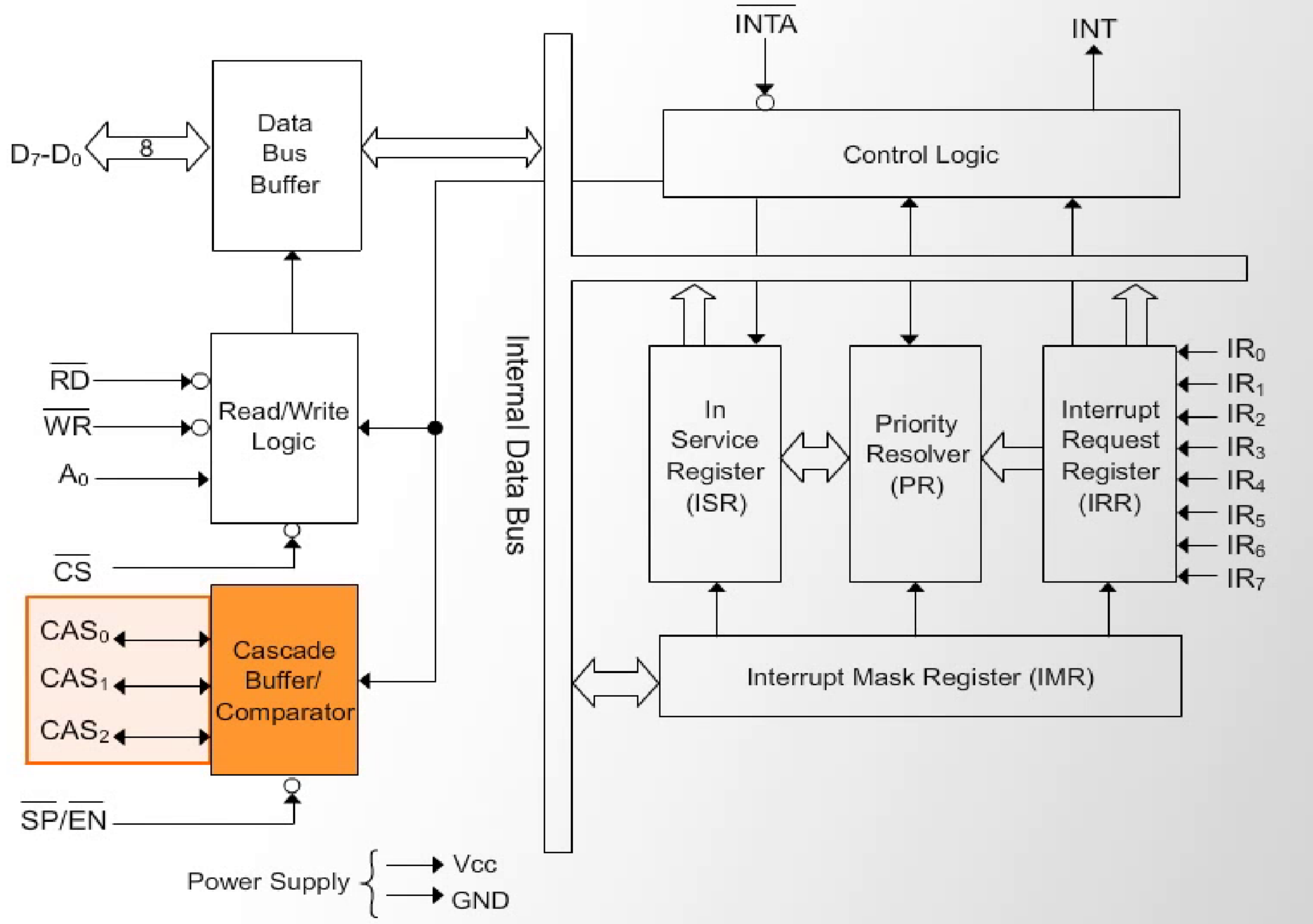
This logic block determines the priorities of the bits set in the IRR. The highest priority is selected and strobed into the corresponding bit of the ISR during INTA Pulse.

The Internals of 8259

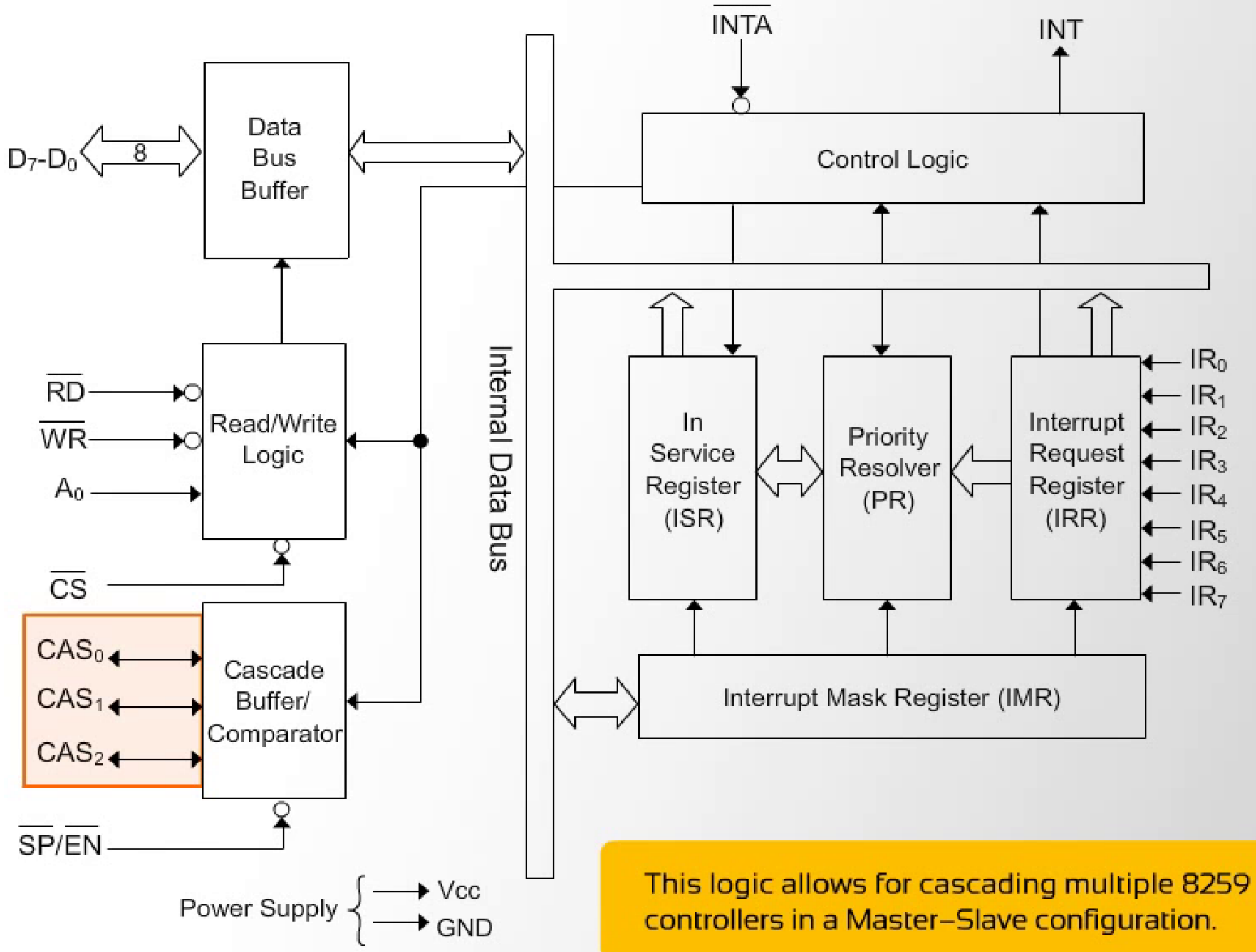


This logic block masks Interrupt lines based on programming by the processor. It prevents masked Interrupt lines from interrupting the processor.

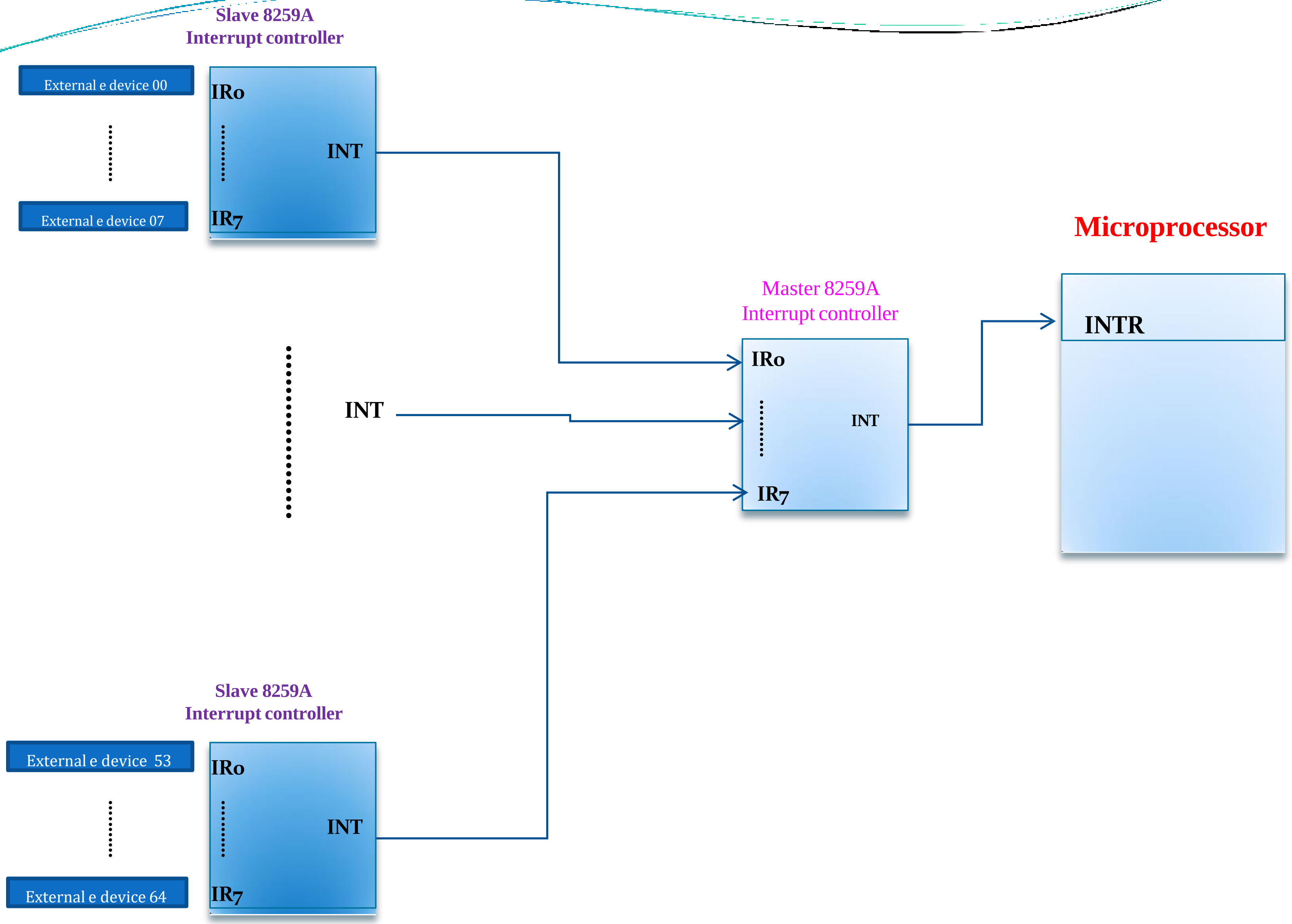
The Internals of 8259



The Internals of 8259

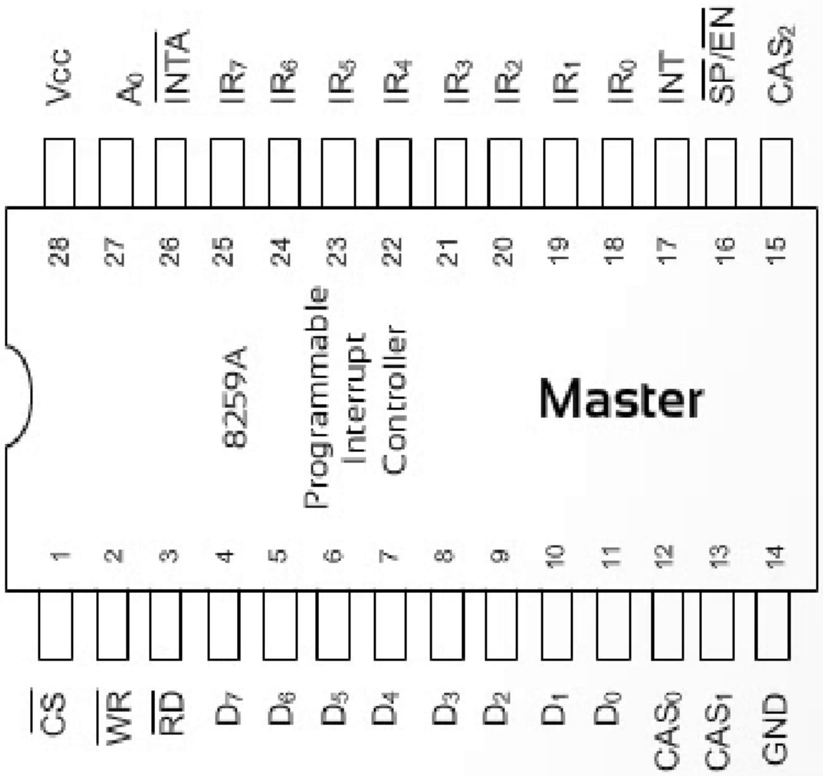


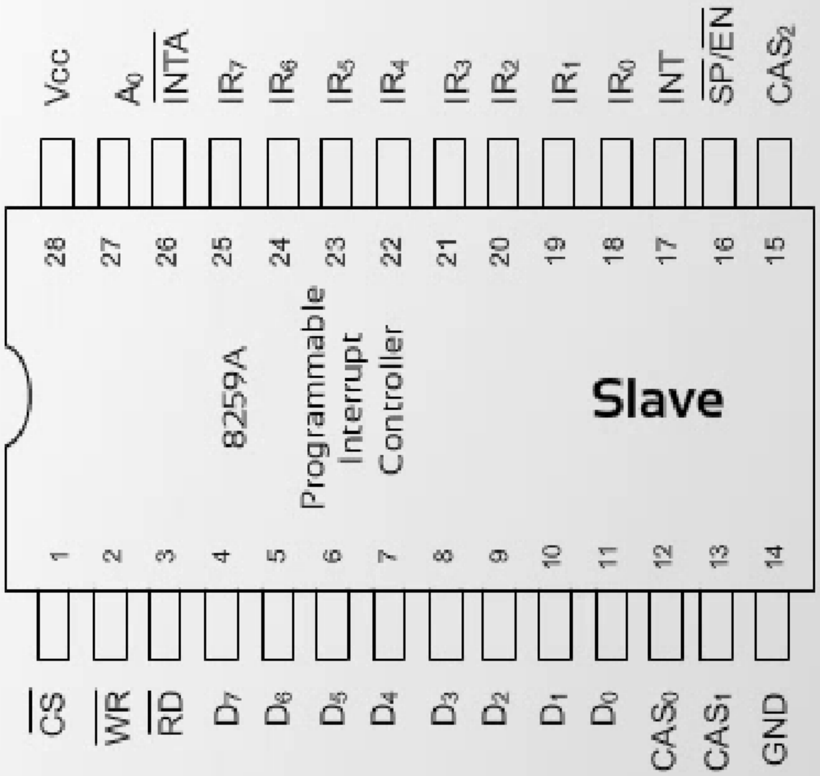
This logic allows for cascading multiple 8259 controllers in a Master–Slave configuration.



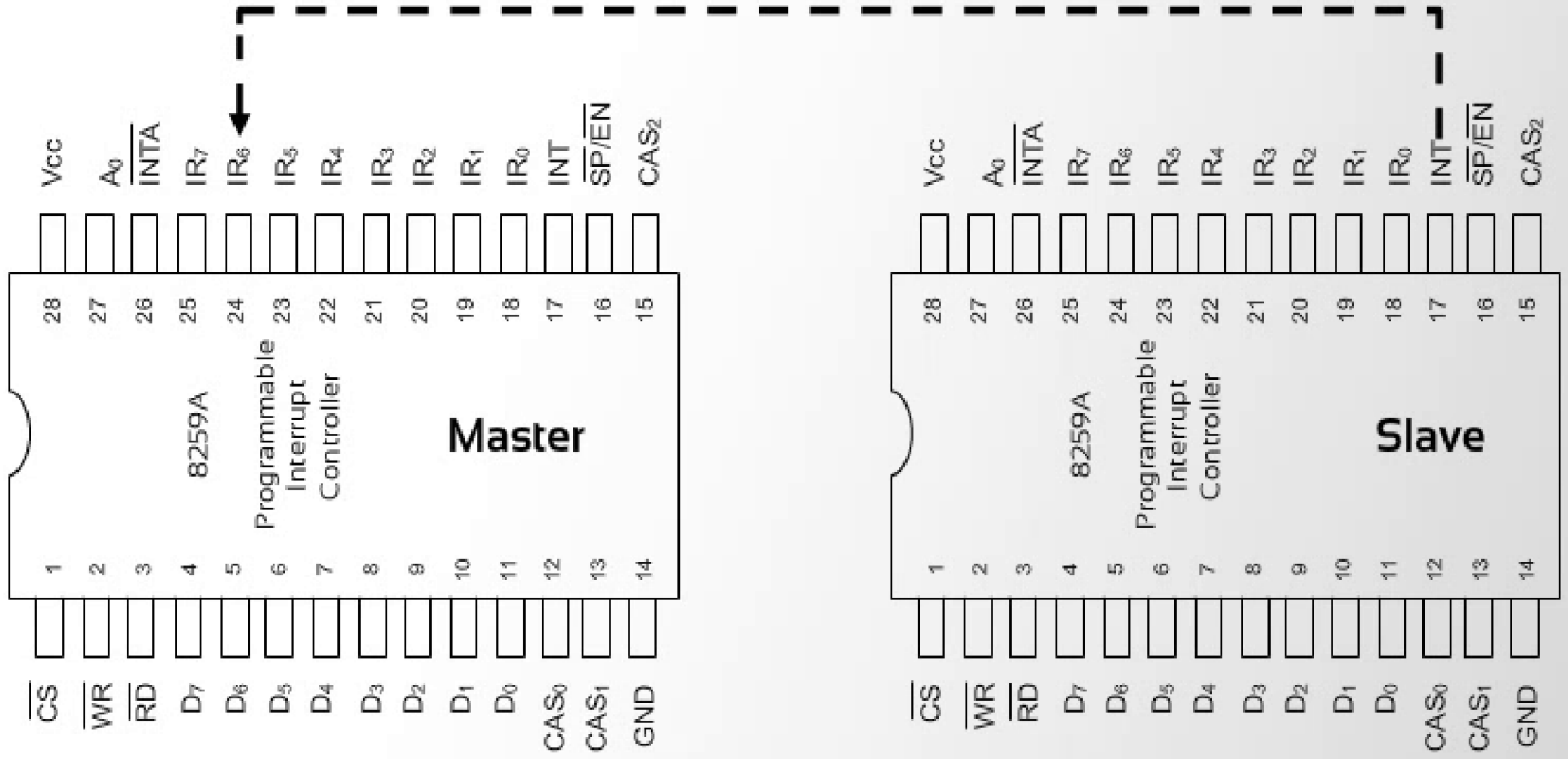
CONNECTING MULTIPLE (64) INTERRUPTED I/O DEVICES TO PROCESSOR

Master–Slave Concept in 8259





Master–Slave Concept in 8259



Week - 15

Programming the 8259A: -

The 82C59A accepts two types of command words generated by the CPU:

1. **Initialization Command Words (ICWs):**

   Before normal operation can begin, each 82C59A in the system must be brought to a starting point - by a sequence of 2 to 4 bytes timed by WR pulses.

2. **Operational Command Words (OCWs):**

   These are the command words which command the 82C59A to operate in various interrupt modes. Among these modes are:

   a. Fully nested mode.

   b. Rotating priority mode.

   c. Special mask mode.

   d. Polled mode.

The OCWs can be written into the 82C59A anytime after initialization.

Initialization Command Words:

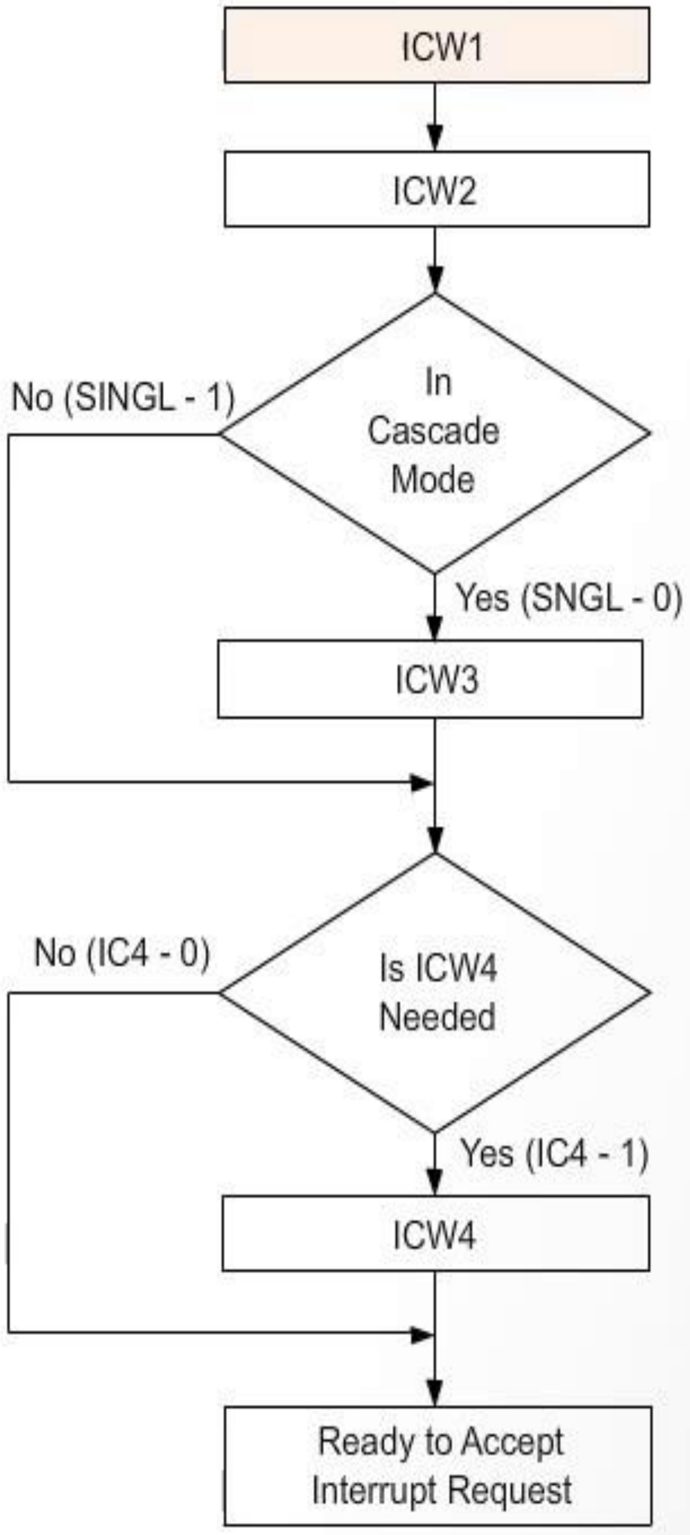
There are four Initialization Command Words for the 8259A that are selected with the help of logic level of A0 pin.

When the 8259A is first powered up, it must be sent ICW1, ICW2 and ICW4.

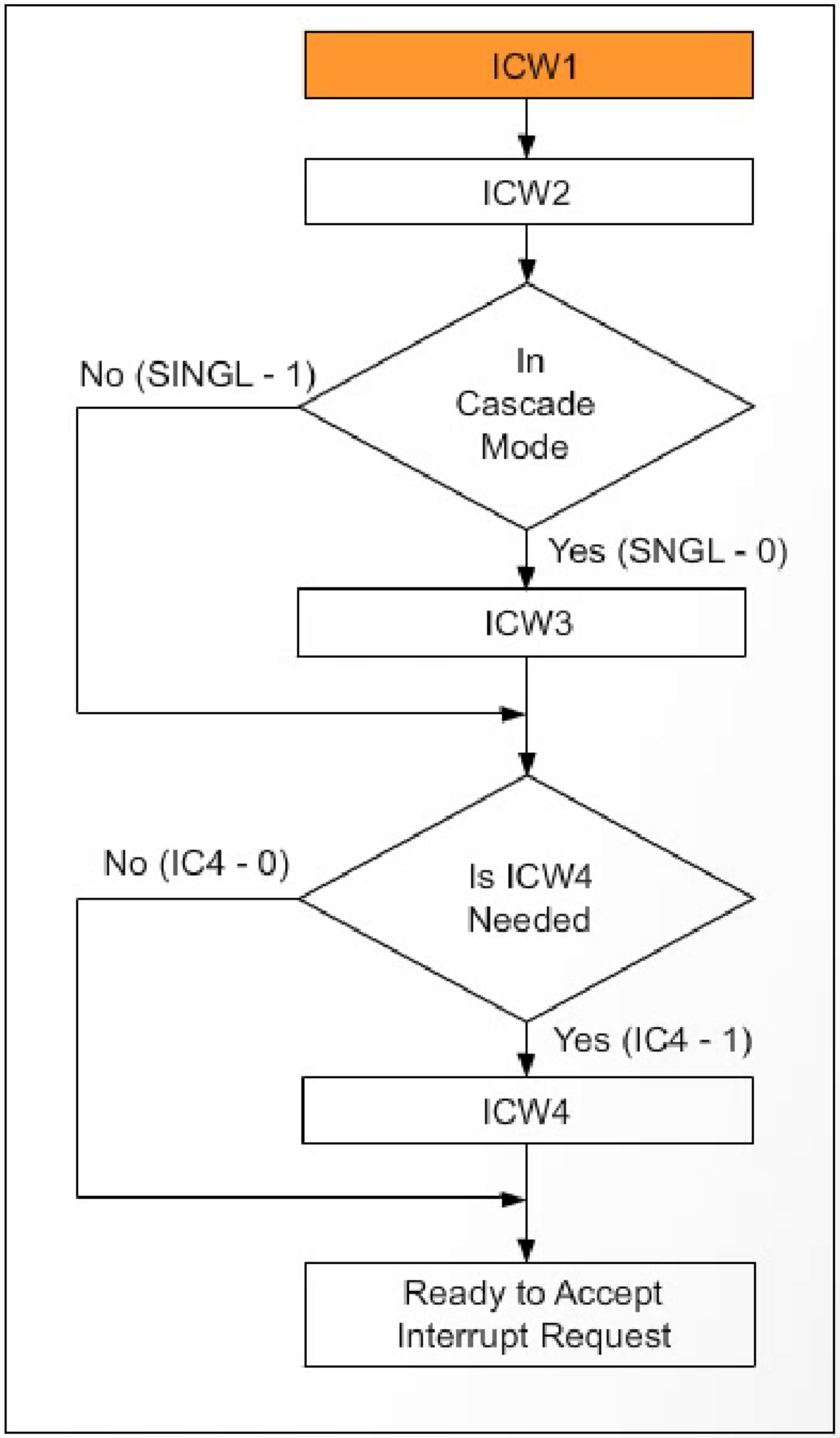
If the 8259A is programmed in cascade mode by ICW1, then we also must program ICW3.

So, if a single 8259A is used in a system ICW1, ICW2 and ICW4 must be programmed.

If cascade mode is used in a system, then all four ICWs must be programmed.



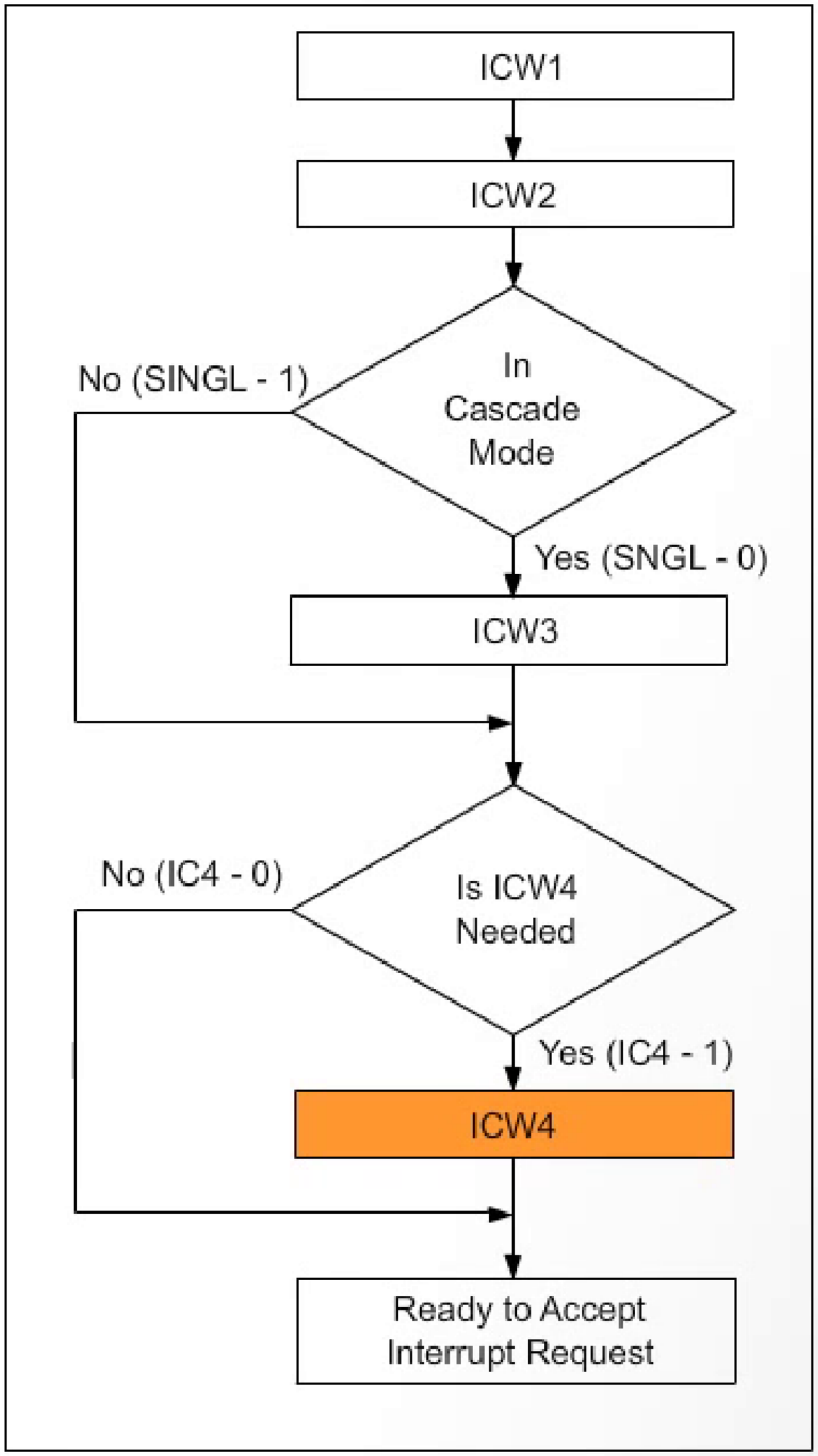
Initialisation Sequence of 8259 A



ICW1 Format

A7	A6	A5	1	LTIM	ADI	SNGL	IC4

Initialisation Sequence of 8259 A



ICW4 Format

0	0	0	SFNM	BUF	M/S	AEOI	MODE

Basics of serial communication & USART

AGENDA

- Data Transmission Types
- Parallel VS Serial
- Synchronous VS Asynchronous
- USART

Why do we need fast interfaces?

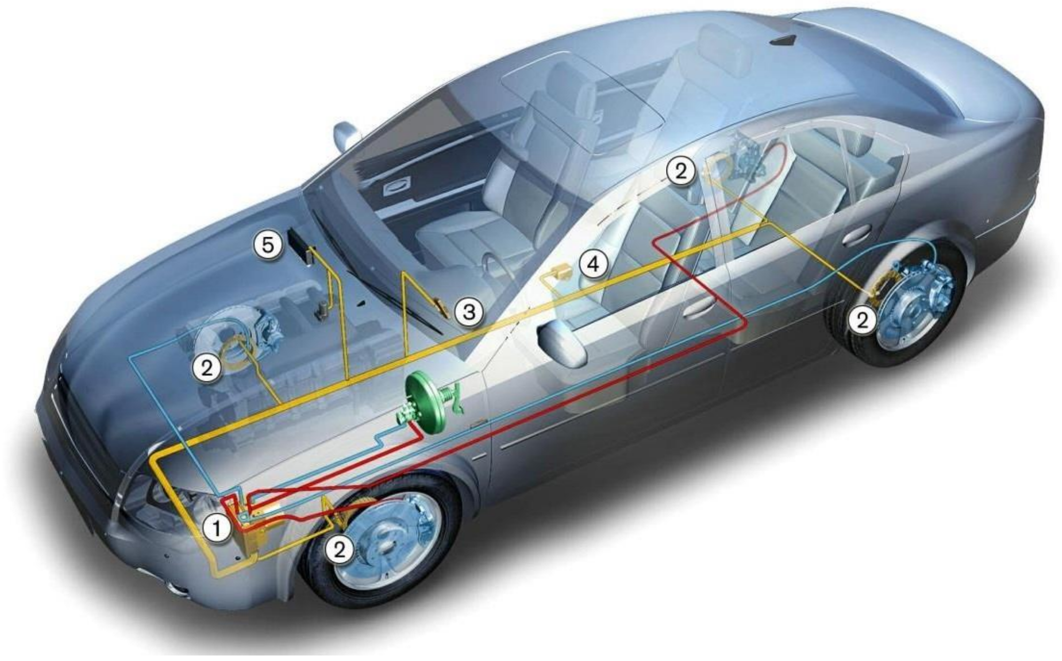
- Microcontrollers need fast ways of communication to the outside world for:

1 Communicating with other microcontrollers, DSPs or even FPGAs. (ex. SRIO, PCIe, I2C)

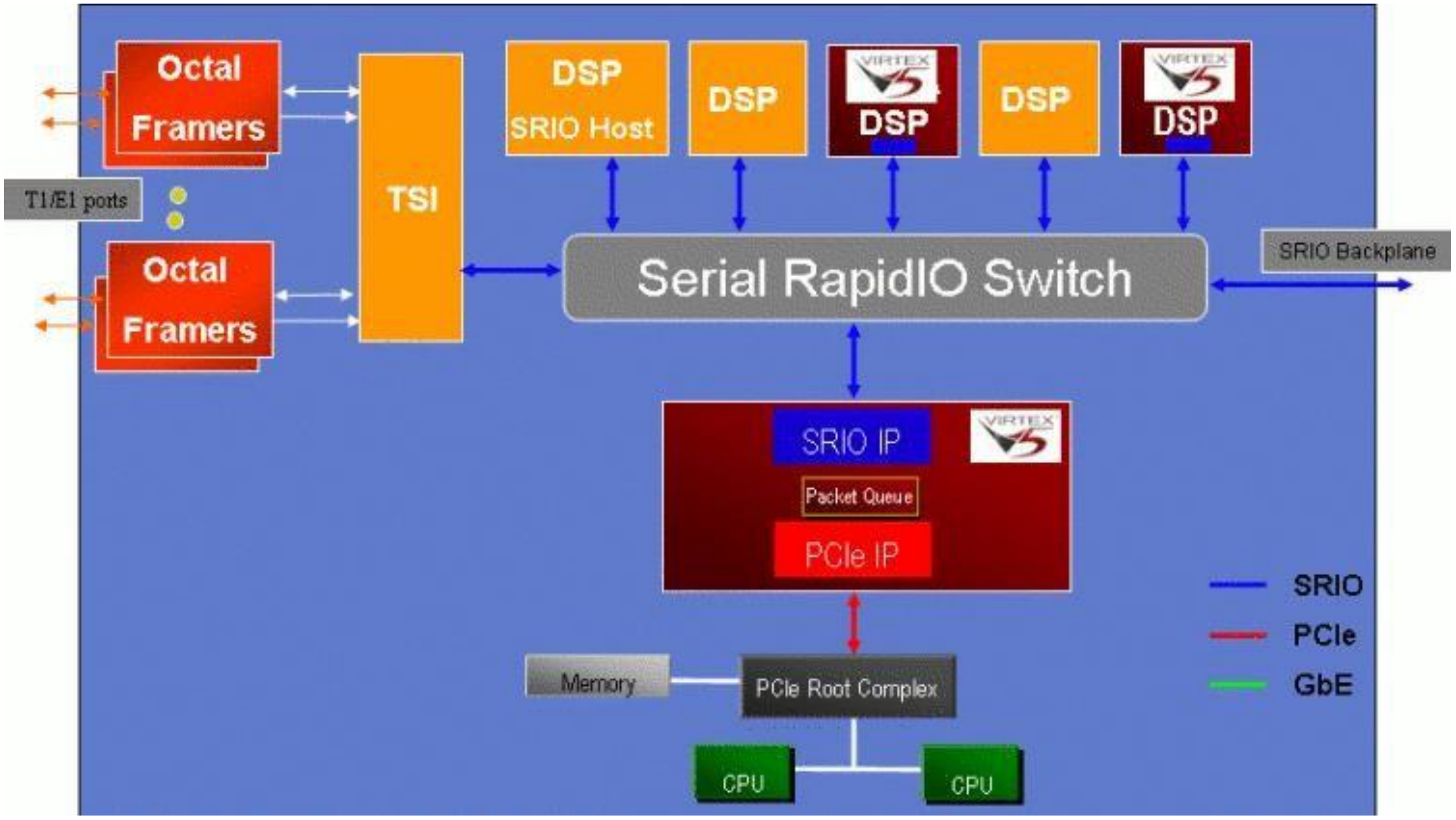
2 Capturing input from user and displaying outputs.

3 Communicating with other microcontrollers on different boards for applications with network of microcontrollers. (ex. CAN and LIN)

- Communicating with other microcontrollers on different boards for applications with network of microcontrollers.



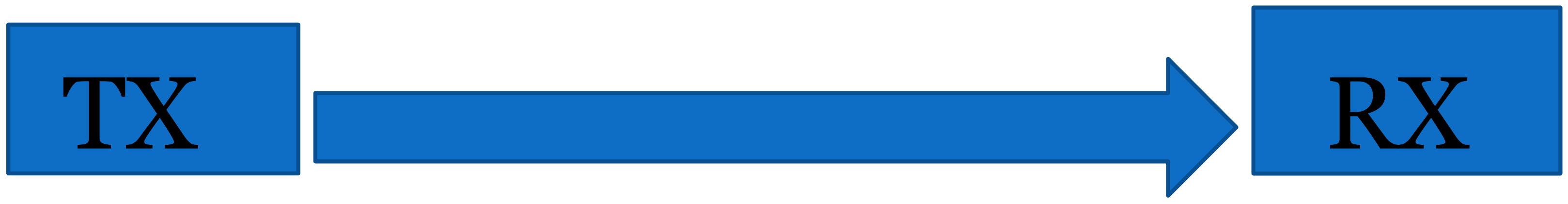
- Communicating with other microcontrollers, DSPs or even FPGAs. (ex. SRIO)



Data Transmission Types

1. **Simplex:**

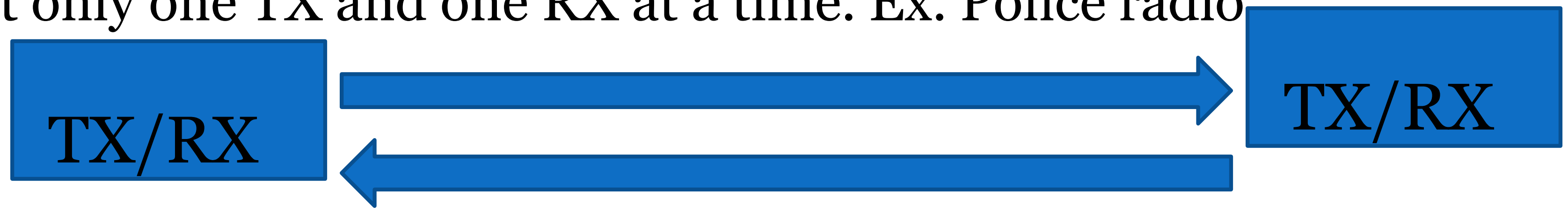
- Communication is possible in one direction only. Ex.TV



2. **Half duplex:**

- Communication is possible in both directions,

but only one TX and one RX at a time. Ex. Police radio



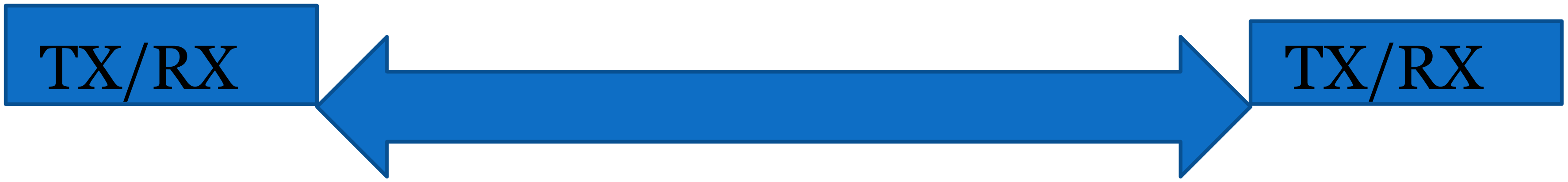
Week - 16

Data Transmission Types

3. **<u>Full duplex:</u>**

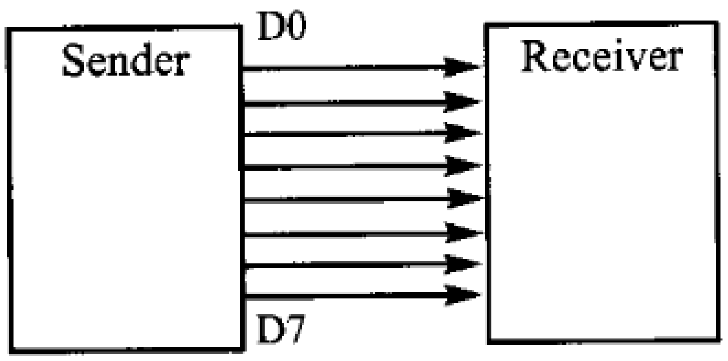
- Communication is possible in both directions,

both sides can transmit and receive in the same time.



Parallel communication

- The process of sending several bits as a whole, on a link with several parallel channels.



- It requires a separate channel for each bit to be transmitted
- A parallel link use simpler hardware as there is no need for a serializer/deserializer.
- Usually used for very short distances.

PARALLEL COMMUNICATION

❖ **Problems of parallel communication:**

1. **Clock skew:**

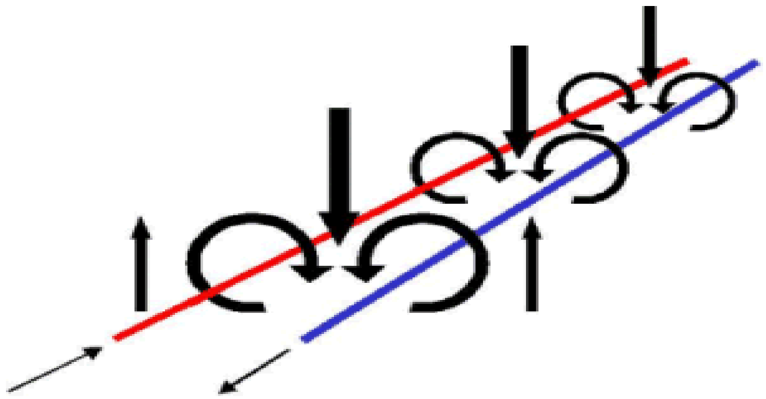
-The clock signal (sent from the clock circuit) arrives at different components (at different times).

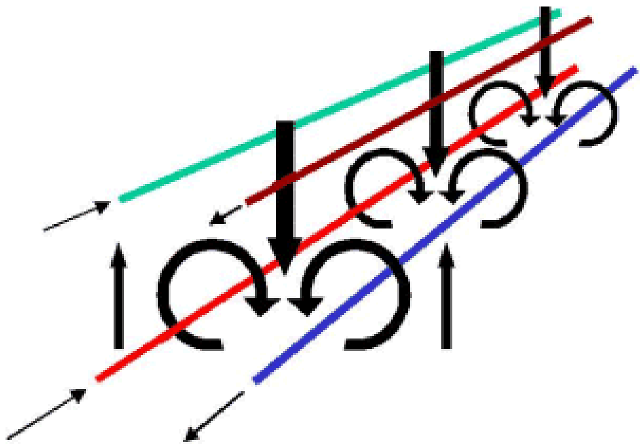
-This happens because of wire-interconnect length, capacitive coupling, material imperfections …etc. It reduces the speed of every link to the slowest of all of the links.

Parallel communication

2. **Crosstalk:**

- Crosstalk happens If the transmitted signal is badly affected by another nearby signal, when electromagnetic energy from one cable leaves an imprint on adjacent cables.





PARALLEL COMMUNICATION

- In parallel communication,
  it can refer to electromagnetic interference from one unshielded pair to another pair, normally running in parallel.

- This issue places an upper limit on the length of a parallel data connection that is usually shorter than a serial connection.

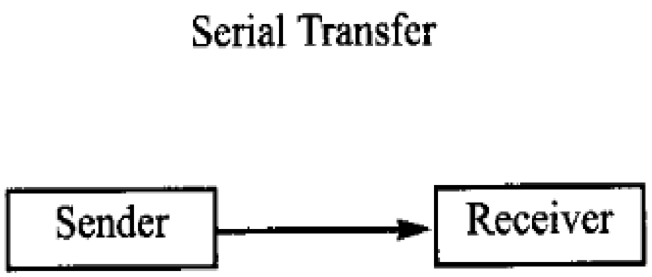
Parallel communication

- **Features of parallel communication:**

1. Very simple.
2. Crosstalk places an upper limit on the length of a parallel data connection (usually shorter than a serial connection).
3. Clock skew between different channels.
4. Low data rate compared to a serial connection for long distances. (Due to the last two reasons).
5. Has a cable cost higher than a serial connection.

SERIAL COMMUNICATION

- The process of sending data bit by bit sequentially, over a single channel between sender and receiver.
- For correct data transmission, there has to be some form of synchronization between transmitter and receiver.
- Cost of cable and synchronization difficulties make parallel communication impractical.



SERIAL COMMUNICATION

4. Serial links can be clocked considerably faster than parallel links, and achieve a higher data rate.

5. Used for all long-haul communication and most computer networks

Comparison

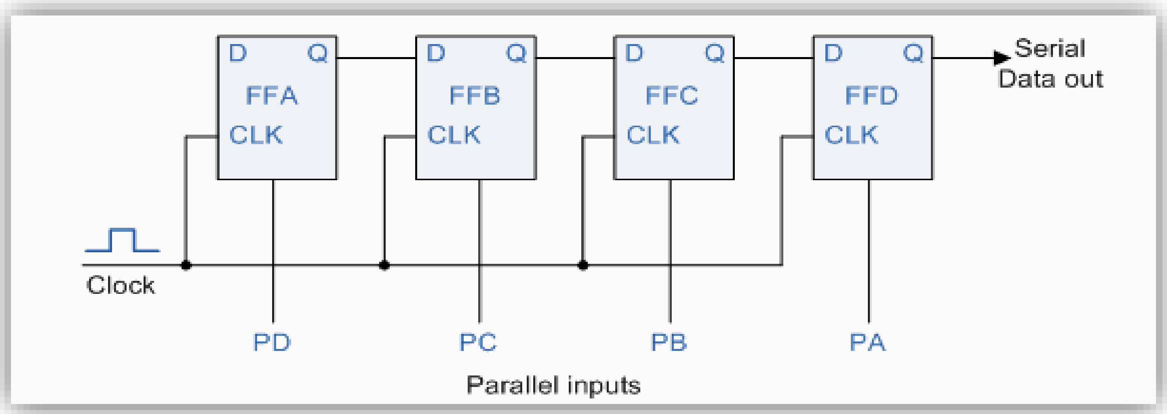
	Parallel	Serial
Crosstalk	Limits the cable length	Not a problem because of using fewer conductors.
Clock skew	Slowing up the data rate	Not an issue for asynchronous communication
Length of the used cable	Short	Long
Cost	High	Low
Simplicity	Simple	Needs a serializer/ deserializer

BASICS OF SERIAL COMMUNICATION

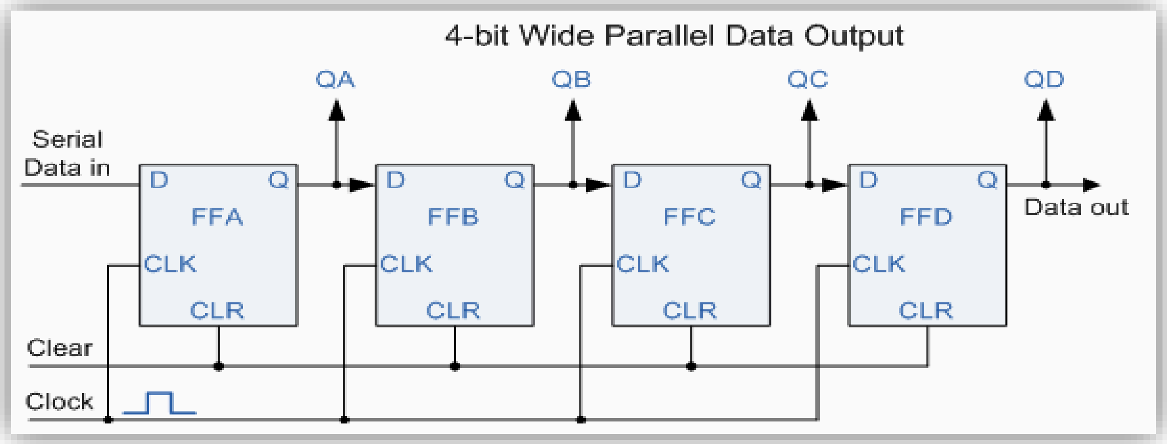
❖ **The basic idea of serial communication:**

- To convert parallel data from a computer bus to serial bits, or to receive serial data we need to use two kinds of shift registers:

1. **Transmitter:**
- A parallel-in, serial-out shift register



2. **Receiver:**
- A serial-in, parallel-out shift register.



- For each clock pulse, the data is shifted in or out.

Basics of serial communication

- The sender and receiver must agree on a set of rules **(Protocol)** on :

1. When data transmission begins and ends.
2. The used bit rate and data packaging format.

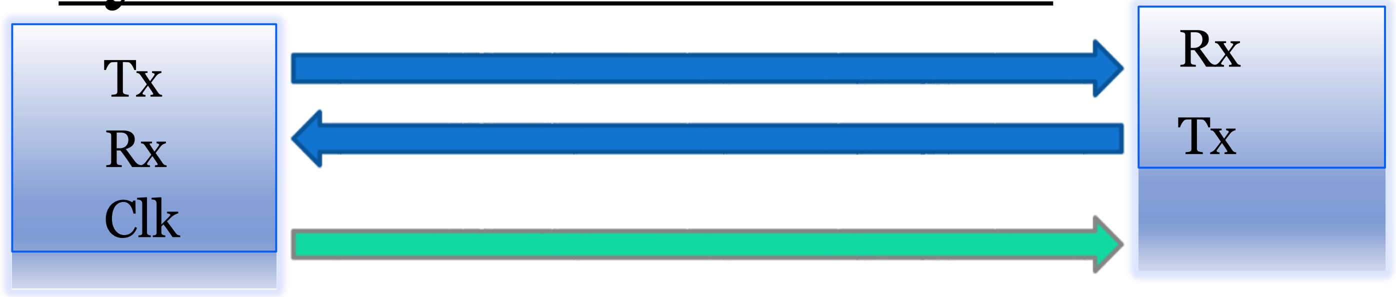
- If there is no reference for the receiver to know when transmission begins or the used bit rate,

  → it'll sample the wrong values and data will be lost.

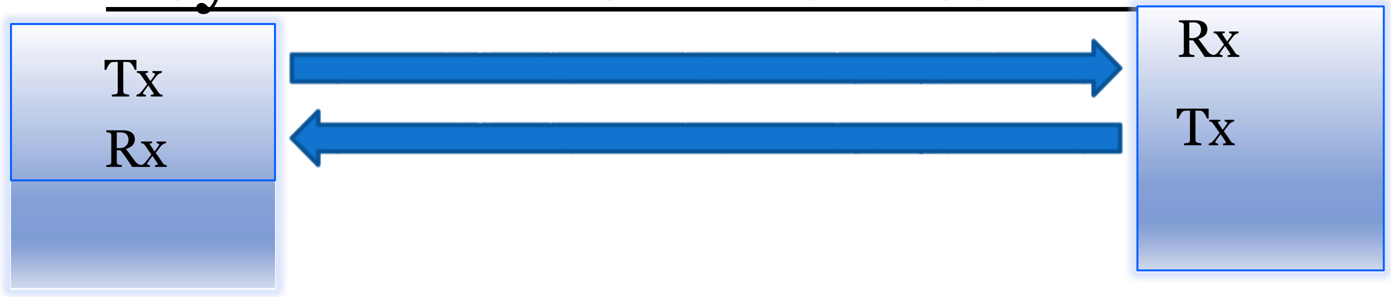
Week - 17

Synchronous VS Asynchronous

1. **Synchronous transmission:**



1. **Asynchronous transmission:**



SYNCHRONOUS VS ASYNCHRONOUS

- In synchronous transmission, a separate link is dedicated for the clock from one terminal (Master) to another (Slave).

- In asynchronous transmission, no link for the clock. Synchronization is done → using a fixed baud rate and using start and stop bits.

HOW SYNCHRONIZATION IS DONE?

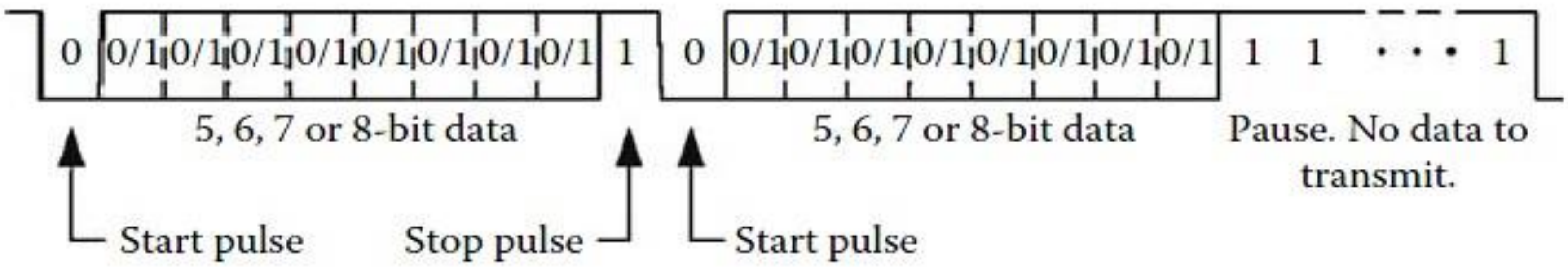
- **For asynchronous transmission:**

- Synchronization is done every word

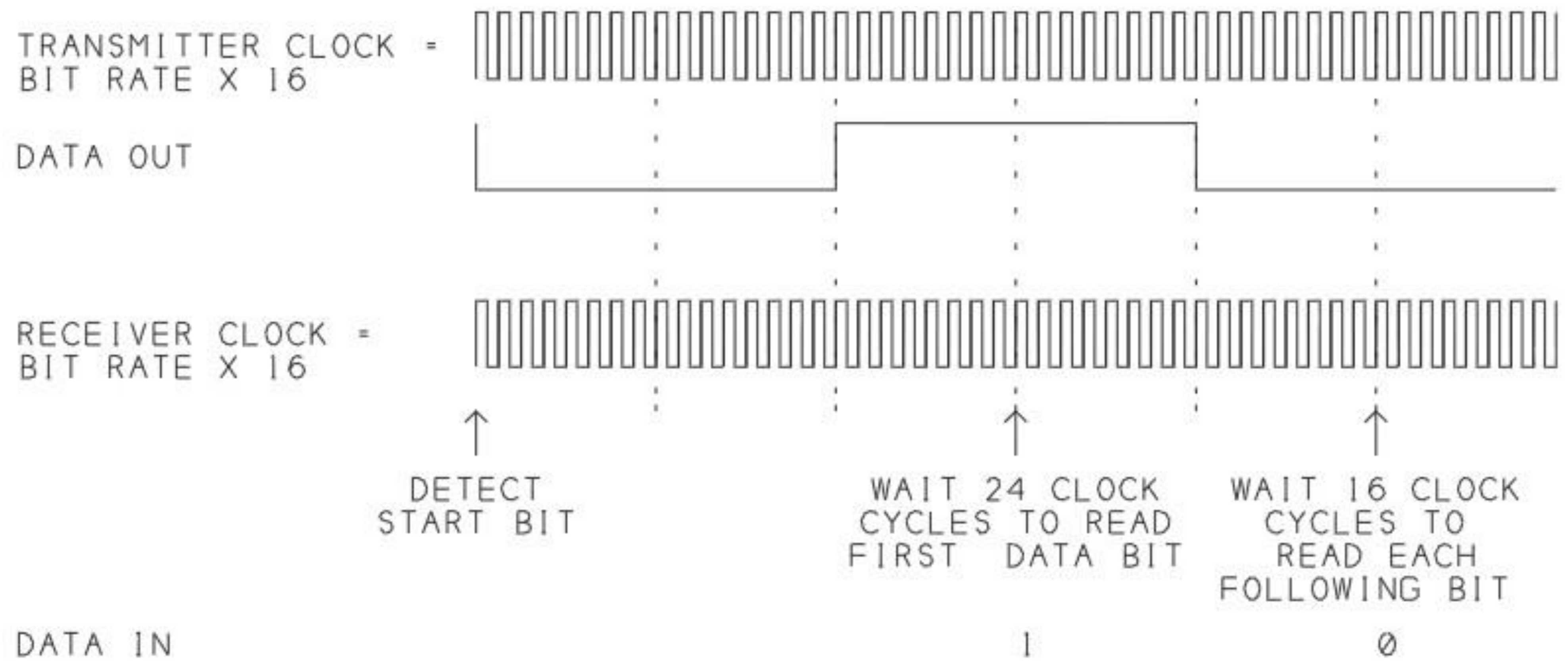
  → A *start bit* with the value 0 indicate the beginning of each word, then eight data bits are sent bit by bit, and finally a *stop bit* with the value 1 to indicate the end of the word.

- Both the transmitter and receiver use the same baud rate.

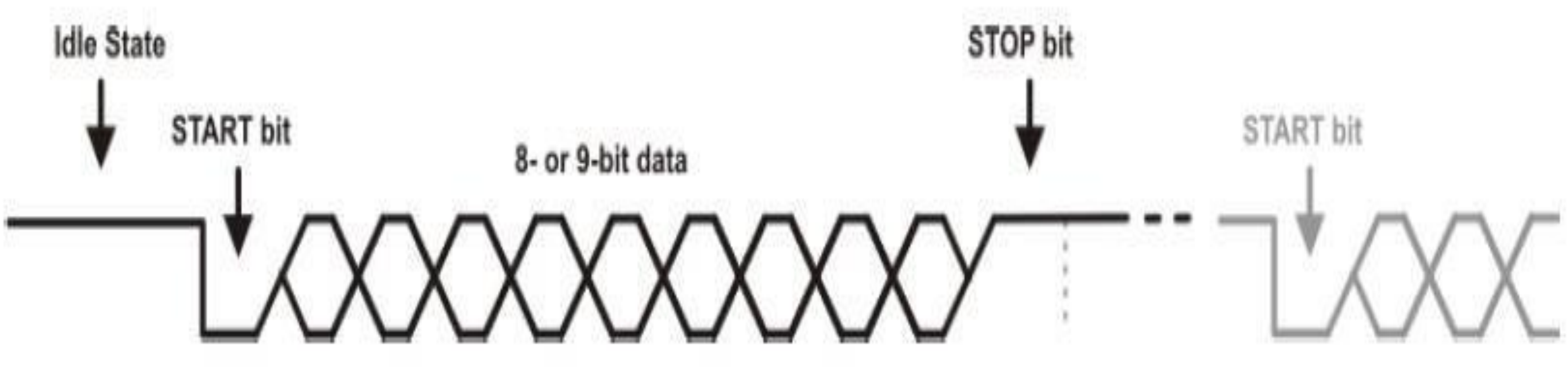
- When the transmitter pauses because it does not have data to transmit (*idle state*), it keeps a sequence of stop bits (logic high) in its output.



- **Over speed clock (x16 baud rate)**


TRANSMITTER CLOCK = BIT RATE X 16
DATA OUT
RECEIVER CLOCK = BIT RATE X 16
DETECT START BIT
WAIT 24 CLOCK CYCLES TO READ FIRST DATA BIT
WAIT 16 CLOCK CYCLES TO READ EACH FOLLOWING BIT
DATA IN
1
Ø

Universal Synchronous Asynchronous Receiver Transmitter (USART)



USART

- The USART module is a full duplex, serial I/O communication peripheral.

- It contains all shift registers, clock generators and data buffers needed for serial communication.

- It can work in synchronous mode, or in asynchronous mode.

- The USART uses two I/O pins to transmit and receive serial data. Both transmission and reception can occur at the same time i.e. ‘full duplex’ operation.

USART

- To send a byte, the application writes the byte to the transmit buffer.
- The UART then sends the data, bit by bit in the requested format, adding Stop, Start, and parity bits as needed.

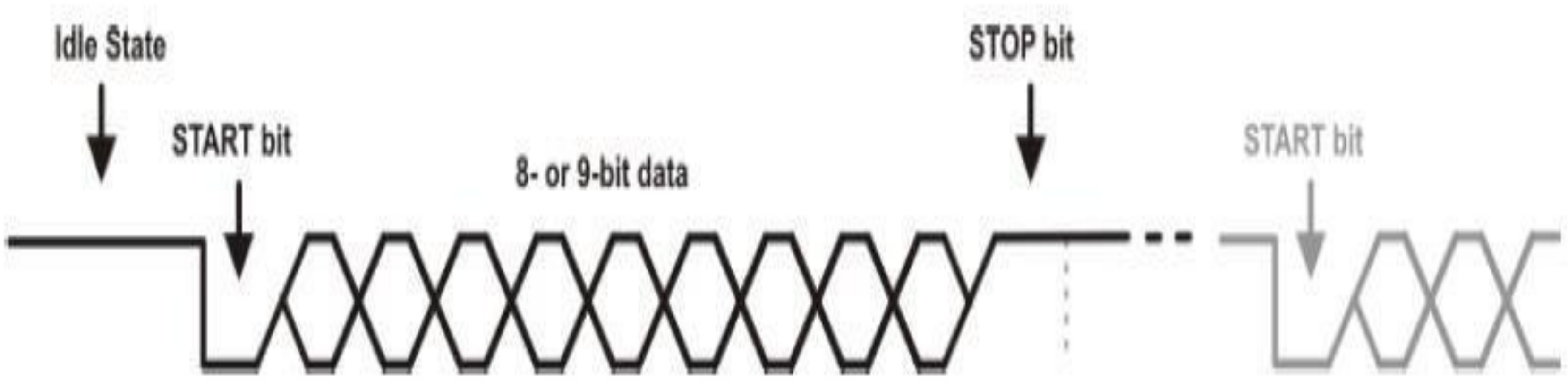
- In a similar way, the UART stores received bytes in a buffer.
- Then the UART can generate an interrupt to notify the application or software can poll the port to find out if data has arrived.

USART

❖ <u>**Asynchronous Mode:**</u>

Data transfer happens in the following way:

1. In idle state, data line has logic high (1).
2. Data transfer starts with a start bit, which is always a zero.
3. Data word is transferred (8 or 9 bit), LSB is sent first.
4. Each word ends with a stop bit, which is always high (1).
5. Another byte can be sent directly after, and will start also with a start bit befor data.



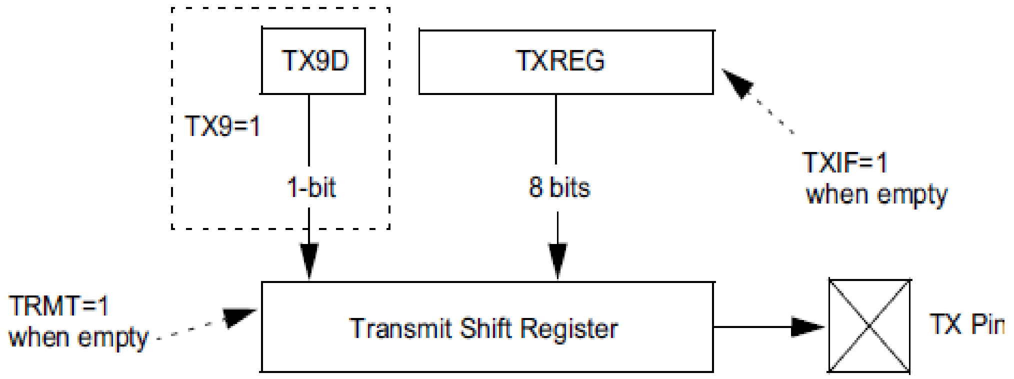
TRANSMITTER

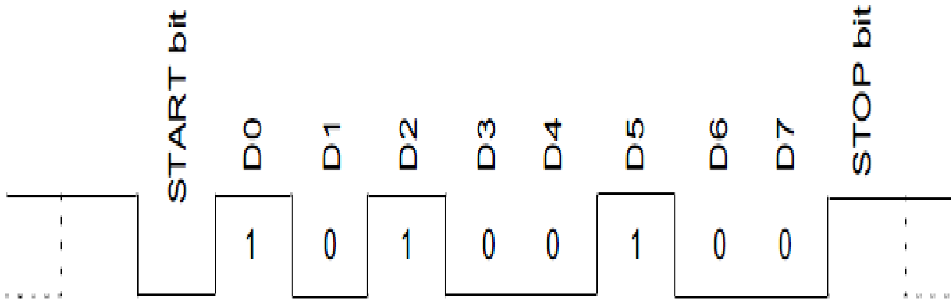
1. The module is enabled by setting the TXEN bit.

2. Data to be sent should be written into the TXREG register. When using 9-bit, TX9D must be written before writing TXREG.

3. Byte will be immediately transferred to the shift register TSR after the STOP bit from the pervious load is sent.

4. From there, data will be clocked out onto the TX pin preceded by a START bit and followed by a STOP bit.



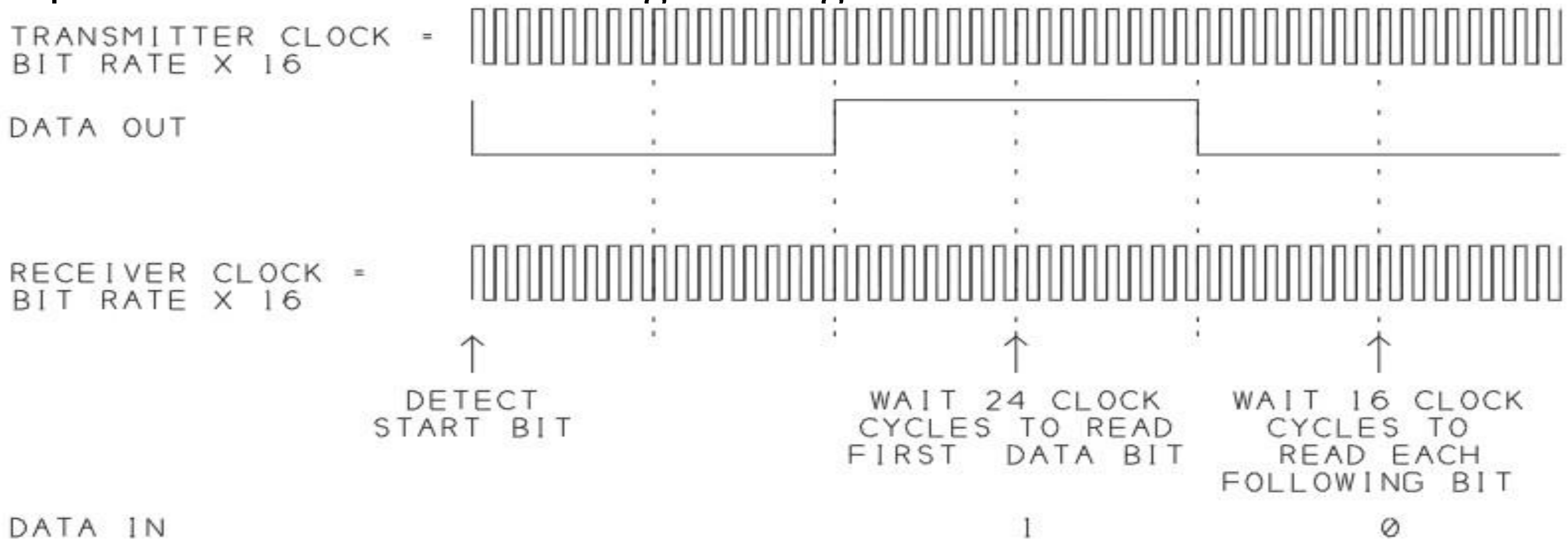


RECEIVER

1. The clock of the receiver is a multiple of the bit rate, in PIC 16f877A,it's x16 or x64.

So, each bit is transmitted/received in 16 clock cycle.

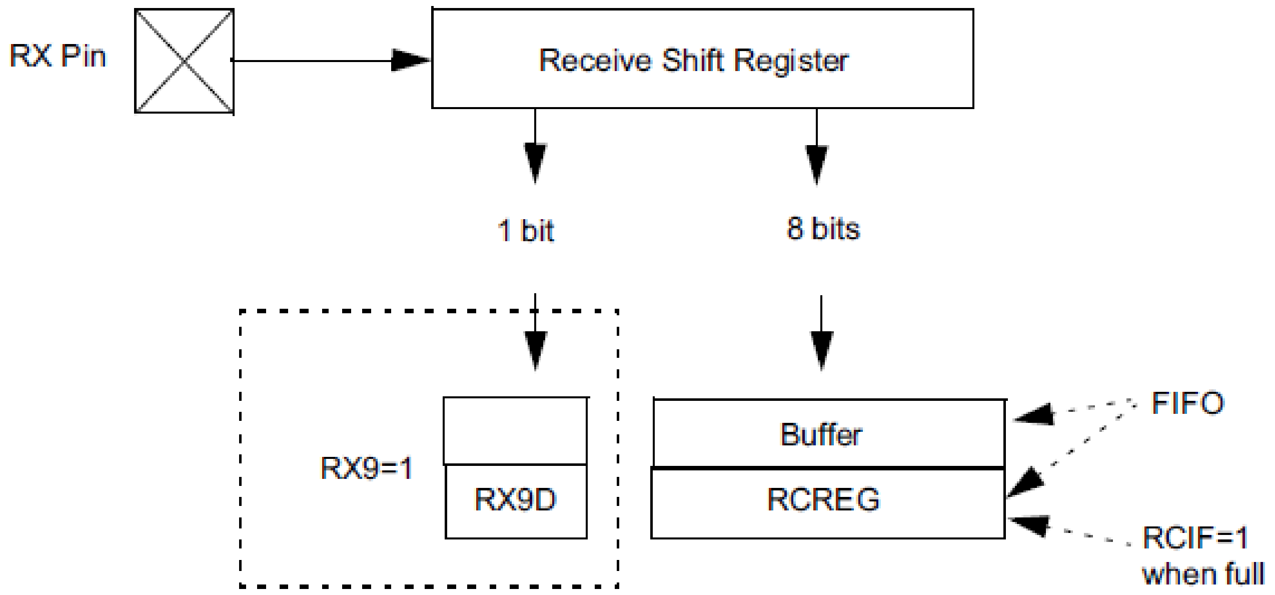
2. If the receiver detects a start bit for a period= bit period (16 clock cycles), then it waits for the period of half bit, and then sample the value on the RX pin and shift it in the receiving shift register.



TRANSMITTER CLOCK = BIT RATE X 16
DATA OUT
RECEIVER CLOCK = BIT RATE X 16
DETECT START BIT
WAIT 24 CLOCK CYCLES TO READ FIRST DATA BIT
WAIT 16 CLOCK CYCLES TO READ EACH FOLLOWING BIT
DATA IN
1
0

RECEIVER

3. Every received bit is sampled at the middle of the bit's time period.
4. The USART can be configured to receive eight or nine bits by the RX9 bit in the RCSTA register.
5. After the detection of a START bit, eight or nine bits of serial data are shifted from the RX pin into the Receive Shift Register, one bit at a time.
6. After the last bit has been shifted in, the STOP bit is checked and the data is moved into the FIFO buffer.


RX Pin
Receive Shift Register
1 bit
8 bits
RX9=1
RX9D
Buffer
RCREG
FIFO
RCIF=1
when full