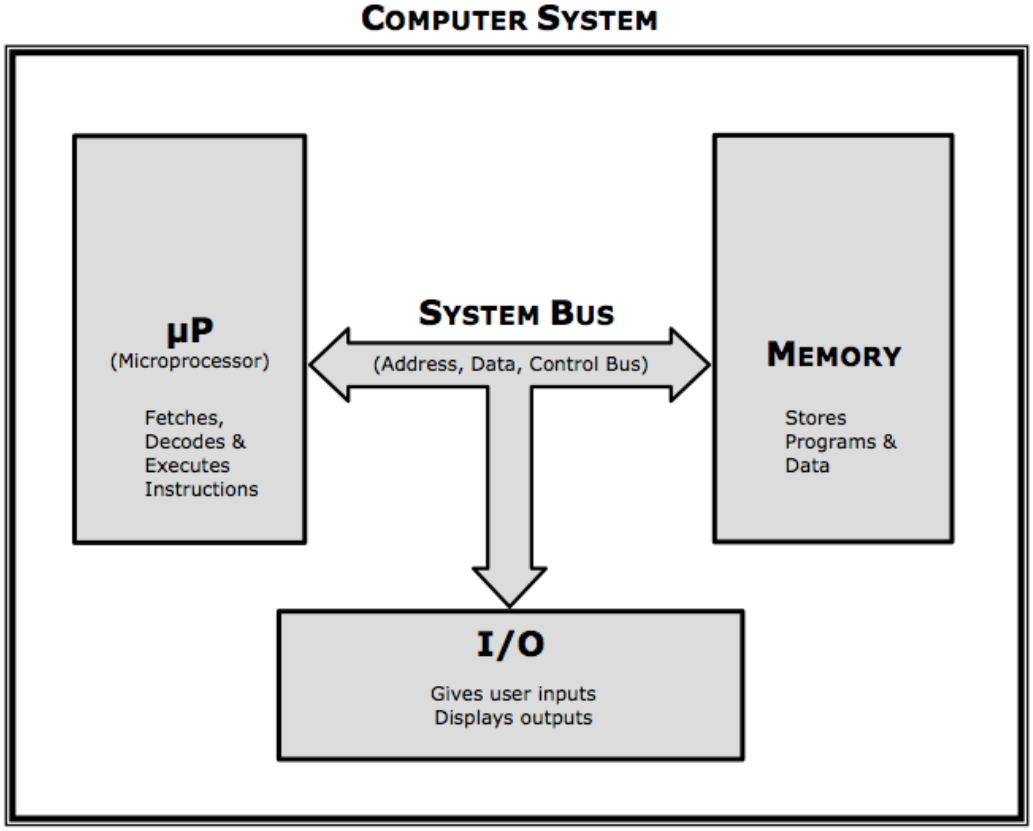
INTRODUCTION TO MICROPROCESSORS

WWW.BHARATACHARYAEDUCATION.COM

INTRODUCTION | BASIC ORGANIZATION OF A COMPUTER

A computer system, as we know it, consists of various components.
They can be broadly classified into three sections:
The Processor, Memory and I/O.



THE PROCESSOR – "µP"

The heart of the computer is its µP (Microprocessor).
Current generation computers use processors like Intel Core i3, i5 or i7 and so on. They have come a long way from the initial processors that you are about to learn E.g.: 8085, 8086 etc.

Back in the day (1940s), when micro-electronics was not invented, processors looked very different and were certainly not "micro" in appearance. They were created using huge arrays of physical switches which were operated manually and often occupied large rooms.

In the following decades, with the invention of micro-electronics, scientists managed to embed thousands of microscopic switches (transistors) inside a small chip, and called it a "**Micro-processor**".

Over the years, microprocessors grew in strength.
From housing a few thousand transistors (8085) to containing more than a billion transistors (Core i7), the computational power has been increasing exponentially. Having said that, some of the basics still remain the same.

To put it simply, **the main function of a µP is to Fetch, Decode and Execute instructions.**

Instructions are a part of programs. Programs are stored in the memory.
Firstly, µP fetches an instruction from the memory.
It then decodes the instruction. This means, it "understands" the binary pattern of the instruction, also called its opcode. Every instruction when stored in the memory is in its unique binary form, which indicates the operation to be performed. This is called its opcode.
Upon decoding the opcode, µP understands the operation to be performed and hence "executes" the instruction. This entire process is called an **"Instruction cycle"**.
Now the process is repeated for the next instruction.
Like this, one by one, all instructions of a program are executed.
Of course by advanced concepts like **pipelining**, **multitasking**, **multiprocessing** etc., this procedure has become very advanced and efficient today. You will get to learn all of them, in the due course of this ever intriguing subject.

We begin learning with basic processors like **8085** or **8086**, but make no mistake, non of this is "outdated". Yes, your mobile phone or your computer today uses the most advanced cutting edge processors (**A11 Bionic** et.al), but to run a **traffic light** or **TV remote** control you don't need a core i7 now, do you? And these are used by the millions across the world. They simply use processors of the same grade as an 8085 or an 8086, with different product numbers as they are made by various manufacturers.

MEMORY

Memory is used to store information.
It stores two kinds of information… programs and data.

For example:
MS Word is a program, and the word documents are its data.
Video player is a program, and the videos are its data.
WhatsApp is a program, and the messages are its data, and so on.
All programs and data are stored in the memory, in digitized form, where every information is represented in 1s and 0s called binary digits or simply bits.

There are various forms of memory devices.
The main memory also called primary memory consists of Ram and ROM.
Other memory devices like Hard disk, Floppy, CD/ DVD etc. are secondary storage devices.
Additionally there is also a high speed memory called Cache composed of SRAM.
For the majority portion of this book, you are dealing with the initial processors like 8086.
It will be in your best interest to think of Primary Memory only, whenever we speak of memory. That is because, secondary memory and high speed memories were implemented much later in the evolution of processors as the demand for mass storage and high speed performance started increasing. So, from now on in this book, unless specified otherwise, **the word memory refers to primary memory that is RAM and ROM.**

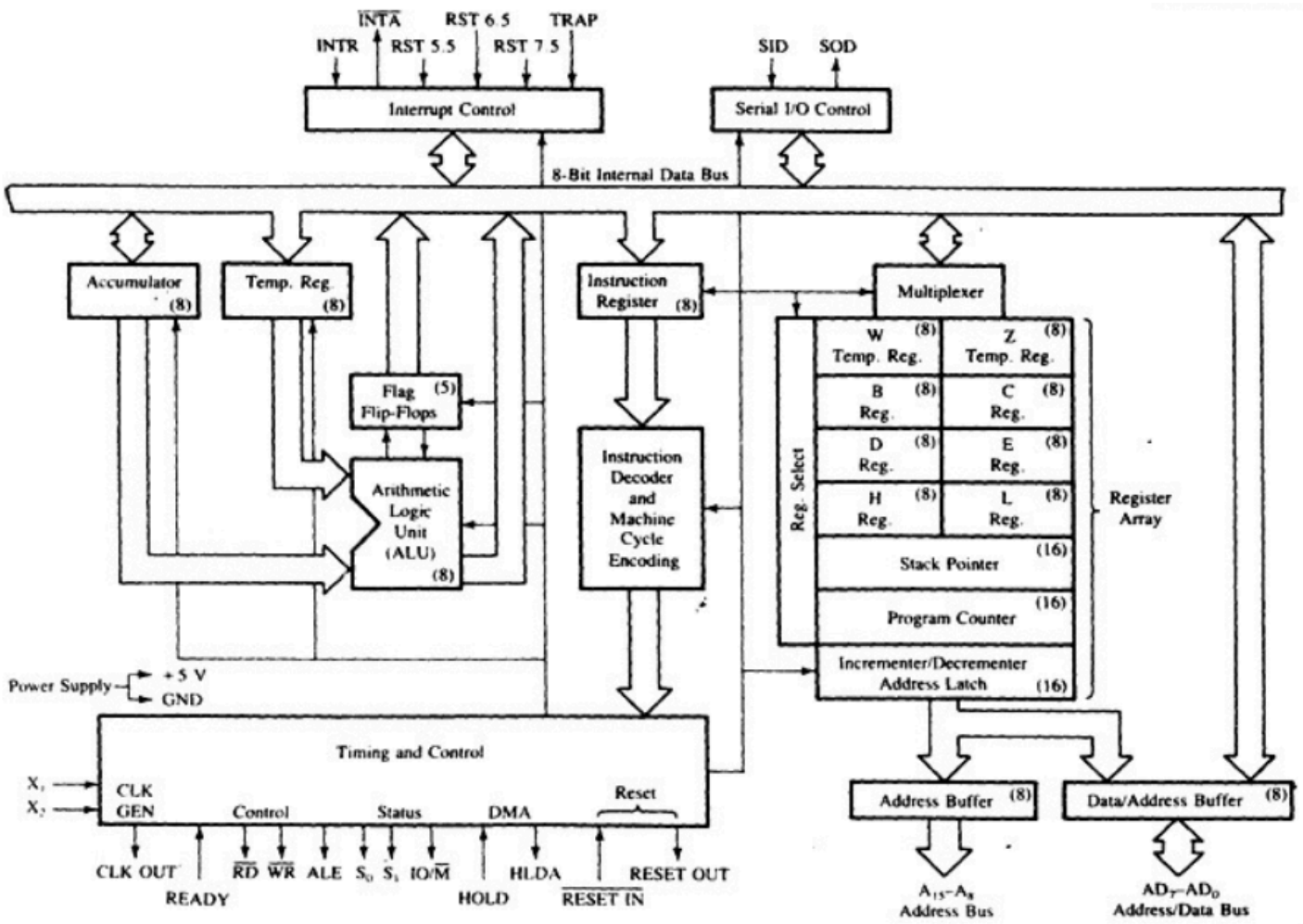
The memory is a series of locations.
Each location is identified by its own unique address.
Every location contains 1 Byte (8 bits) of data. There is a very good reason for this, and you will learn it when we discuss the topic of memory banking in 8086.

I/O DEVICES

I/O devices are used to enter programs and data as inputs and display or print the results as outputs. We are all familiar with devices such as the keyboard, mouse, printer, monitor etc. Every form of computer system has a set of I/O devices for human interaction. A device like a touch screen performs dual functions of both input and output.

The µP, Memory and I/O are all connected to each other using the System Bus.

ARCHITECTURE OF 8085



Registers

Program Counter (PC, 16-bits):

It is a 16-bit Special-Purpose register. It holds **address** of the **next instruction**.
PC is incremented by the INR/DCR after every instruction byte is fetched.

Stack Pointer (SP, 16-bits):

It is a 16-bit Special-Purpose register. It holds **address** of the **top of the Stack**.
Stack is a set of memory locations operating in LIFO manner.
SP is **decremented** on every **PUSH** operation and **incremented** on every **POP**.

B, C, D, E, H, L registers 8-bits each:

These are 8-bit General-Purpose registers**.**
They can also be used to store 16-bit data in register pairs.
The possible register **pairs** are **BC** pair, **DE** pair and **HL** pair.
The **HL** pair also holds the **address** for the Memory Pointer **"M"**.

Temporary Registers (WZ, 16-bits):

This is a 16-bit register pair.
It is **used by µP** to hold **temporary** values in some instructions like CALL/JMP etc.
The **programmer** has **no access** to this register pair.

INR/DCR Register (16-bits):

This is a 16-bit shift register.
It is used to **increment PC after every instruction byte is fetched** and **increment** or **decrement SP** after a Pop or a Push operation respectively.
It is not available to the programmer.

A - Accumulator (8-bits):

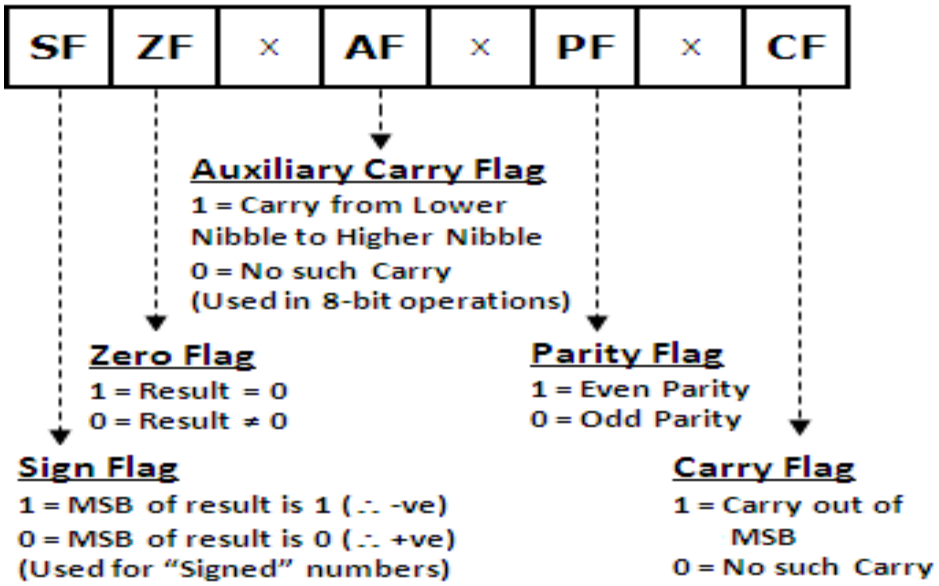
It is an 8-bit programmable register.
The user can read or write this register.
It has two **special properties**:

- It **holds one** of the **operands** during most of the arithmetic operations.
- It **holds** the **result** of most of the arithmetic and logic operations

Temp Register (8-bits):

This is an 8-bit register.
It is **used by µP** for storing one of the operands during an operation.
The **programmer** has **NO ACCESS** to this register.

FLAG REGISTER OF 8085



S - Sign Flag:

It is **set** (1) when **MSB** of the result is **1** (i.e. result is a **-**VE number).
It is reset (0) when MSB of the result is 0 (i.e. result is a **+**VE number).

Z - Zero Flag:

It is **set** when the **result** is **= zero**.
It is reset when the result is not = zero.

AC - Auxiliary Carry Flag:

It is **set** when an **Auxiliary Carry** / Borrow is **generated**.
It is reset when an Auxiliary Carry / Borrow is not generated.
Auxiliary Carry is the Carry generated **between** the **lower nibble and the higher nibble** for an 8-bit operation. It is not affected after a 16- bit operation. It is used only in DAA operation.

P - Parity Flag:

It is **set (1)** when result has **even parity**. It is reset when result has odd parity.

C - Carry Flag:

It is **set** when a **Carry / Borrow** is **generated from the MSB**.
It is reset when a Carry / Borrow is not generated from the MSB.

 # In the exam, Show at least 2 examples from Bharat Sir's lecture notes

INTERRUPTS OF 8085

This Block is responsible for controlling the **hardware interrupts** of 8085.
8085 supports the following hardware interrupts:

TRAP:

This is an **edge as well as level triggered**, **vectored** interrupt.
It cannot be masked by SIM instruction and can neither be disabled by DI instruction.
It has the **highest priority**.
Its vector address is **0024H**.

RST 7.5:

This is an **edge triggered**, **vectored** interrupt.
It can be masked by SIM instruction and can also be disabled by DI instruction.
It has the **second highest priority**.
Its vector address is **003CH**.

RST 6.5:

This is a **level triggered**, **vectored** interrupt.
It can be masked by SIM instruction and can also be disabled by DI instruction.
It has the **third highest priority**.
Its vector address is **0034H**.

RST 5.5:

This is a **level triggered**, **vectored** interrupt.
It can be masked by SIM instruction and can also be disabled by DI instruction.
It has the **fourth highest priority**.
Its vector address is **002CH**.

INTR:

This is a **level triggered**, **non-vectored** interrupt.
It cannot be masked by SIM instruction but can be disabled by DI instruction.
It has the **lowest priority**.
It has an **acknowledgement signal $\overline{\text{INTA}}$** .
The address for the ISR is **fetched from external hardware**.

$\overline{\text{INTA}}$:

This is an **acknowledgement** signal **for INTR** (only).
This signal is used to **get** the Op-Code (and hence the ISR address) from External hardware in order to execute the ISR. ☺ In case of doubts, contact Bharat Sir: - 98204 08217.

ALL Interrupts **EXCEPT TRAP** can be **disabled** though the **DI** instruction.
These interrupts can be **enabled** again by the **EI** Instruction.
Interrupts can be individually **masked or unmasked by SIM instruction**.
TRAP and INTR are not affected by SIM instruction.

SERIAL CONTROL

This Block is responsible for transferring data Serially to and from the µP.

SID - Serial In Data:
µP receives data, bit-by-bit through this line.

SOD - Serial Out Data:
µP sends out data, bit-by-bit through this line.
Serial transmission can be done by **RIM** and **SIM** Instructions.

ALU – ARITHMETIC LOGIC UNIT

8085 has an **8-bit ALU**.
It performs 8-bit arithmetic operations like Addition and Subtraction.
It also performs logical operations like AND, OR, EX-OR NOT etc.
It takes **input** from the **Accumulator** and the **Temp** register.
The **output** of most of the ALU operations is stored back **into** the **Accumulator**.

INSTRUCTION REGISTER AND DECODER

Instruction Register:
The 8085 places the contents of the PC onto the Address bus and fetches the instruction.
This fetched instruction is stored into the Instruction register.

Instruction Decoder:
The fetched instruction from the Instruction register enters the Instruction Decoder.
Here the instruction is decoded and the decode information is given to the Timing and Control Circuit where the instruction is executed..

TIMING AND CONTROL CIRCUIT

The timing and control circuit issues the various internal and external control signals for executing and instruction.
The external pins connected to this circuit are as follows:

X1 and X2:
These pins provide the **Clock Input to the µP**.
Clock is provided from a crystal oscillator.

ClkOut:
8085 provides the **Clock input** to all **other peripherals** through the ClockOut pin.
This takes care of **synchronizing** all peripherals with 8085.

$\overline{\text{ResetIn}}$:
This is an active low signal activated when the manual reset signal is applied to the µP. This signal **resets the µP**. On Reset PC contains **0000H**. Hence, the **Reset Vector Address** of 8085 is 0000H.

ResetOut:

This signal is connected to the reset input of all the peripherals.
It is used to **reset the peripherals once** the **µP** is **reset**.

READY:

This is an active high input.
It is used to **synchronize** the µP with **"Slower"** Peripherals.
The **µP samples** the **Ready** input in the beginning of every Machine Cycle.
If it is found to be **LOW**, the **µP executes** one **WAIT CYCLE** after which it re-samples the ready pin till it finds the Ready pin HIGH.
∴ The **µP remains** in the **WAIT STATE until** the **READY** pin becomes **high** again.
Hence, **if** the **Ready** pin is **not required** it should be **connected** to the **Vcc,** and not, left unconnected, **otherwise** would cause the µP to execute **infinite wait cycles**. *#Please refer Bharat Sir's Lecture Notes for this ...*

ALE - Address Latch Enable:

This signal is **used to latch address** from the multiplexed Address-Data Bus (**AD0-AD7**). When the Bus contains **address**, **ALE** is **high**, **else** it is **low**.

IO/ $\overline{M}$:

This signal is used to distinguish between an **IO** and a **Memory operation**.
When this signal is high it is an IO operation else it is a Memory operation.

$\overline{RD}$:

This is an active low signal used to indicate a **read operation**.

$\overline{WR}$:

This is an active low signal used to indicate a **write operation**.

S_1 and S_0:

These lines denote the status of the µP

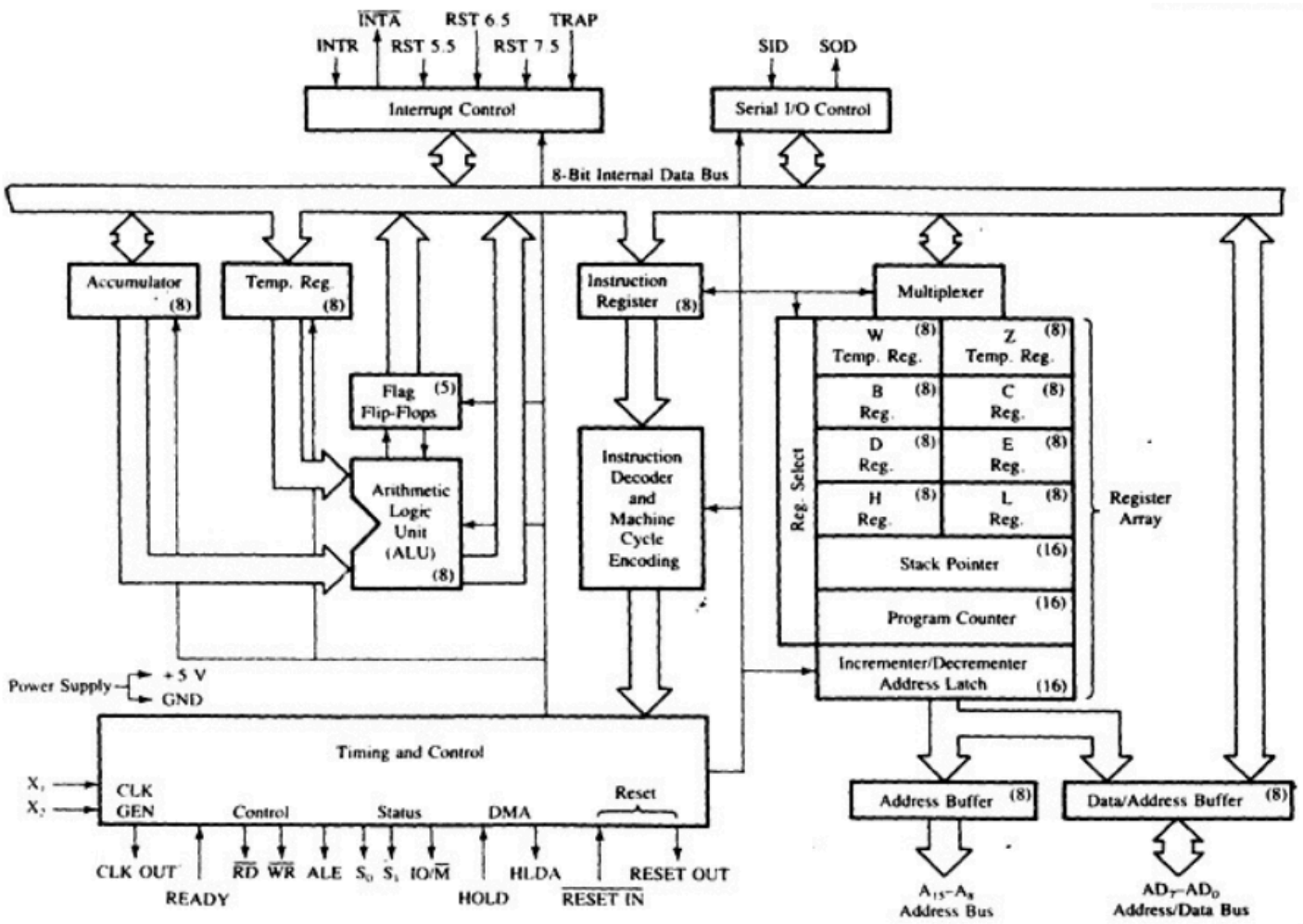
S1	S0	STATUS
0	0	Idle
0	1	Write
1	0	Read
1	1	Opcode fetch

HOLD and HLDA:

The Hold and Hold Acknowledge signals are used for **Direct Memory Access** (DMA).
The **DMA Controller issued** the **Hold** signal to the µP.
In response the **µP releases** the **System bus**.
After releasing the system bus the **µP acknowledges** the Hold signal with **HLDA** signal.
The **DMA Transfer** thus **begins**.
DMA **Transfer** is **terminated** by **releasing** the **HOLD** signal.

*Note: **"Programmer's Model"** simply means all registers in the architecture that are available to the programmer. So if they ask Programmers model, draw the internal part of the architecture without the pins, and explain all registers.*

ARCHITECTURE OF 8085



Registers

Program Counter (PC, 16-bits):

It is a 16-bit Special-Purpose register. It holds **address** of the **next instruction**.
PC is incremented by the INR/DCR after every instruction byte is fetched.

Stack Pointer (SP, 16-bits):

It is a 16-bit Special-Purpose register. It holds **address** of the **top of the Stack**.
Stack is a set of memory locations operating in LIFO manner.
SP is **decremented** on every **PUSH** operation and **incremented** on every **POP**.

B, C, D, E, H, L registers 8-bits each:

These are 8-bit General-Purpose registers**.**
They can also be used to store 16-bit data in register pairs.
The possible register **pairs** are **BC** pair, **DE** pair and **HL** pair.
The **HL** pair also holds the **address** for the Memory Pointer **"M"**.

Temporary Registers (WZ, 16-bits):

This is a 16-bit register pair.
It is **used by µP** to hold **temporary** values in some instructions like CALL/JMP etc.
The **programmer** has **no access** to this register pair.

INR/DCR Register (16-bits):

This is a 16-bit shift register.
It is used to **increment PC after every instruction byte is fetched** and **increment** or **decrement SP** after a Pop or a Push operation respectively.
It is not available to the programmer.

A - Accumulator (8-bits):

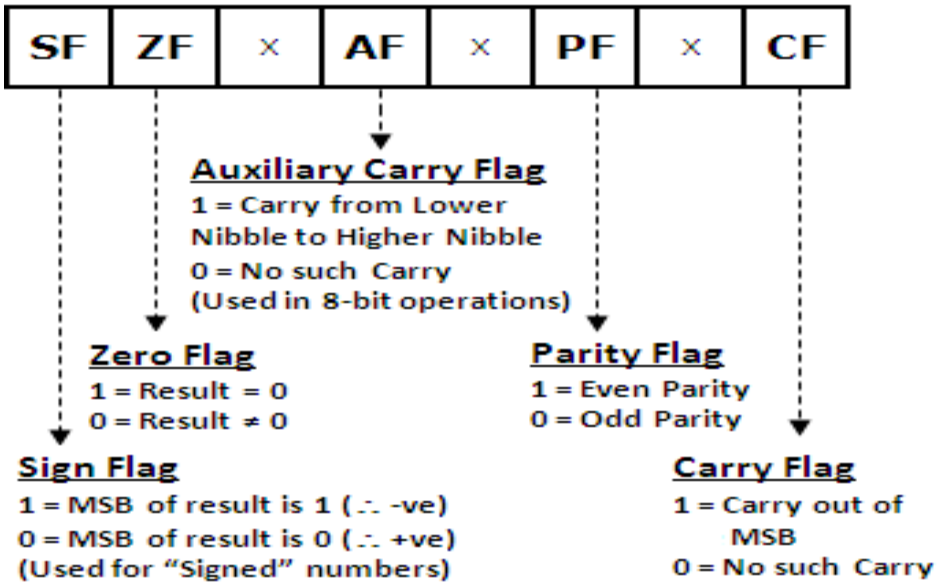
It is an 8-bit programmable register.
The user can read or write this register.
It has two **special properties**:

- It **holds one** of the **operands** during most of the arithmetic operations.
- It **holds** the **result** of most of the arithmetic and logic operations

Temp Register (8-bits):

This is an 8-bit register.
It is **used by µP** for storing one of the operands during an operation.
The **programmer** has **NO ACCESS** to this register.

FLAG REGISTER OF 8085



S - Sign Flag:

It is **set** (1) when **MSB** of the result is **1** (i.e. result is a **-**VE number).
It is reset (0) when MSB of the result is 0 (i.e. result is a **+**VE number).

Z - Zero Flag:

It is **set** when the **result** is **= zero**.
It is reset when the result is not = zero.

AC - Auxiliary Carry Flag:

It is **set** when an **Auxiliary Carry** / Borrow is **generated**.
It is reset when an Auxiliary Carry / Borrow is not generated.
Auxiliary Carry is the Carry generated **between** the **lower nibble and the higher nibble** for an 8-bit operation. It is not affected after a 16- bit operation. It is used only in DAA operation.

P - Parity Flag:

It is **set (1)** when result has **even parity**. It is reset when result has odd parity.

C - Carry Flag:

It is **set** when a **Carry / Borrow** is **generated from the MSB**.
It is reset when a Carry / Borrow is not generated from the MSB.

 # In the exam, Show at least 2 examples from Bharat Sir's lecture notes

INTERRUPTS OF 8085

This Block is responsible for controlling the **hardware interrupts** of 8085.
8085 supports the following hardware interrupts:

TRAP:

This is an **edge as well as level triggered**, **vectored** interrupt.
It cannot be masked by SIM instruction and can neither be disabled by DI instruction.
It has the **highest priority**.
Its vector address is **0024H**.

RST 7.5:

This is an **edge triggered**, **vectored** interrupt.
It can be masked by SIM instruction and can also be disabled by DI instruction.
It has the **second highest priority**.
Its vector address is **003CH**.

RST 6.5:

This is a **level triggered**, **vectored** interrupt.
It can be masked by SIM instruction and can also be disabled by DI instruction.
It has the **third highest priority**.
Its vector address is **0034H**.

RST 5.5:

This is a **level triggered**, **vectored** interrupt.
It can be masked by SIM instruction and can also be disabled by DI instruction.
It has the **fourth highest priority**.
Its vector address is **002CH**.

INTR:

This is a **level triggered**, **non-vectored** interrupt.
It cannot be masked by SIM instruction but can be disabled by DI instruction.
It has the **lowest priority**.
It has an **acknowledgement signal $\overline{\text{INTA}}$** .
The address for the ISR is **fetched from external hardware**.

$\overline{\text{INTA}}$:

This is an **acknowledgement** signal **for INTR** (only).
This signal is used to **get** the Op-Code (and hence the ISR address) from External hardware in order to execute the ISR. ☺ In case of doubts, contact Bharat Sir: - 98204 08217.

ALL Interrupts **EXCEPT TRAP** can be **disabled** though the **DI** instruction.
These interrupts can be **enabled** again by the **EI** Instruction.
Interrupts can be individually **masked or unmasked by SIM instruction**.
TRAP and INTR are not affected by SIM instruction.

SERIAL CONTROL

This Block is responsible for transferring data Serially to and from the µP.

SID - Serial In Data:
µP receives data, bit-by-bit through this line.

SOD - Serial Out Data:
µP sends out data, bit-by-bit through this line.
Serial transmission can be done by **RIM** and **SIM** Instructions.

ALU – ARITHMETIC LOGIC UNIT

8085 has an **8-bit ALU**.
It performs 8-bit arithmetic operations like Addition and Subtraction.
It also performs logical operations like AND, OR, EX-OR NOT etc.
It takes **input** from the **Accumulator** and the **Temp** register.
The **output** of most of the ALU operations is stored back **into** the **Accumulator**.

INSTRUCTION REGISTER AND DECODER

Instruction Register:
The 8085 places the contents of the PC onto the Address bus and fetches the instruction.
This fetched instruction is stored into the Instruction register.

Instruction Decoder:
The fetched instruction from the Instruction register enters the Instruction Decoder.
Here the instruction is decoded and the decode information is given to the Timing and Control Circuit where the instruction is executed..

TIMING AND CONTROL CIRCUIT

The timing and control circuit issues the various internal and external control signals for executing and instruction.
The external pins connected to this circuit are as follows:

X1 and X2:
These pins provide the **Clock Input to the µP**.
Clock is provided from a crystal oscillator.

ClkOut:
8085 provides the **Clock input** to all **other peripherals** through the ClockOut pin.
This takes care of **synchronizing** all peripherals with 8085.

$\overline{\text{ResetIn}}$:
This is an active low signal activated when the manual reset signal is applied to the µP. This signal **resets the µP**. On Reset PC contains **0000H**. Hence, the **Reset Vector Address** of 8085 is 0000H.

ResetOut:

This signal is connected to the reset input of all the peripherals.
It is used to **reset the peripherals once** the **µP** is **reset**.

READY:

This is an active high input.
It is used to **synchronize** the µP with **"Slower"** Peripherals.
The **µP samples** the **Ready** input in the beginning of every Machine Cycle.
If it is found to be **LOW**, the **µP executes** one **WAIT CYCLE** after which it re-samples the ready pin till it finds the Ready pin HIGH.
∴ The **µP remains** in the **WAIT STATE until** the **READY** pin becomes **high** again.
Hence, **if** the **Ready** pin is **not required** it should be **connected** to the **Vcc,** and not, left unconnected, **otherwise** would cause the µP to execute **infinite wait cycles**. *#Please refer Bharat Sir's Lecture Notes for this ...*

ALE - Address Latch Enable:

This signal is **used to latch address** from the multiplexed Address-Data Bus (**AD0-AD7**). When the Bus contains **address**, **ALE** is **high**, **else** it is **low**.

IO/ $\overline{M}$:

This signal is used to distinguish between an **IO** and a **Memory operation**.
When this signal is high it is an IO operation else it is a Memory operation.

$\overline{RD}$:

This is an active low signal used to indicate a **read operation**.

$\overline{WR}$:

This is an active low signal used to indicate a **write operation**.

S_1 and S_0:

These lines denote the status of the µP

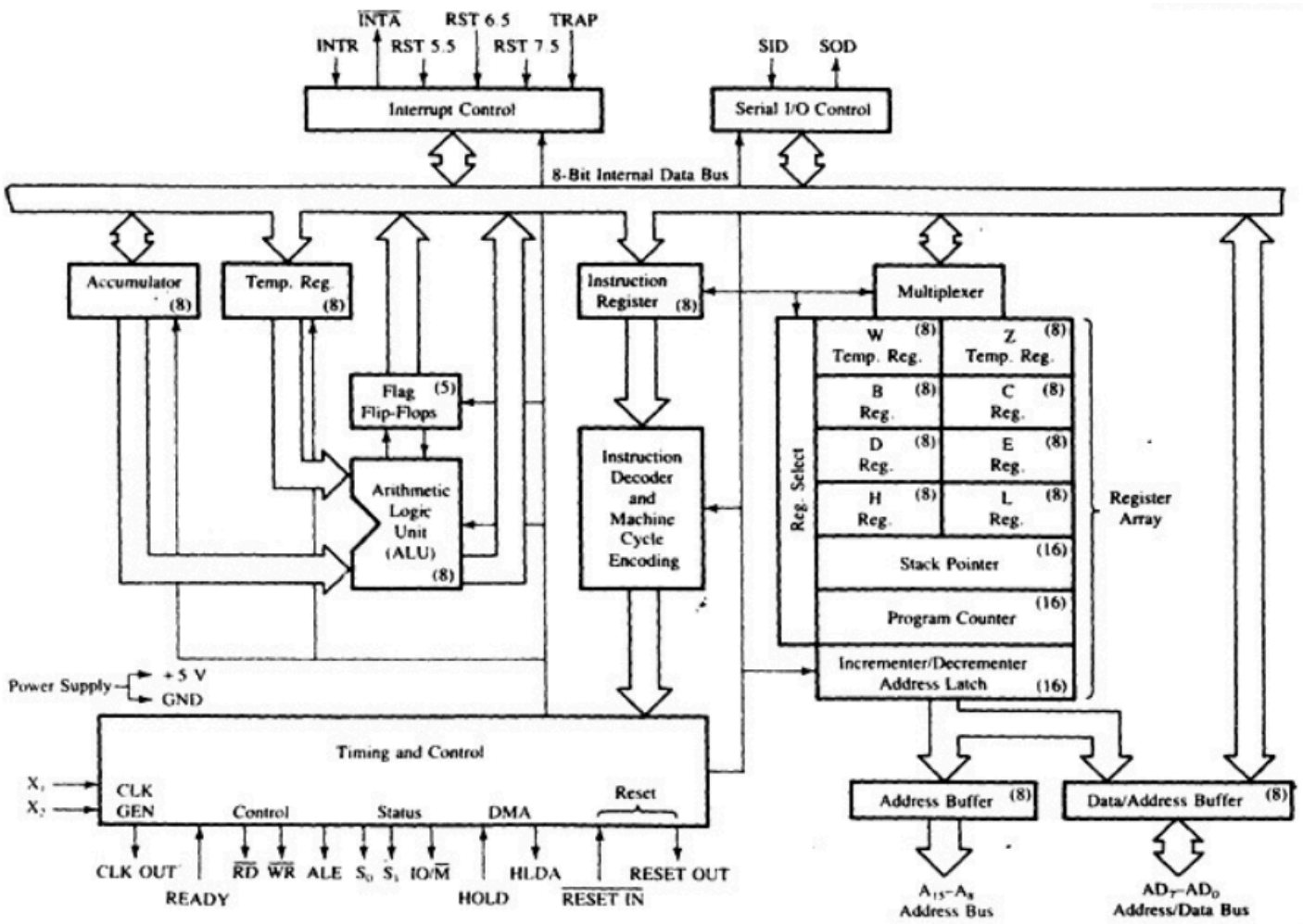
S1	S0	STATUS
0	0	Idle
0	1	Write
1	0	Read
1	1	Opcode fetch

HOLD and HLDA:

The Hold and Hold Acknowledge signals are used for **Direct Memory Access** (DMA).
The **DMA Controller issued** the **Hold** signal to the µP.
In response the **µP releases** the **System bus**.
After releasing the system bus the **µP acknowledges** the Hold signal with **HLDA** signal.
The **DMA Transfer** thus **begins**.
DMA **Transfer** is **terminated** by **releasing** the **HOLD** signal.

*Note: **"Programmer's Model"** simply means all registers in the architecture that are available to the programmer. So if they ask Programmers model, draw the internal part of the architecture without the pins, and explain all registers.*

ARCHITECTURE OF 8085



Registers

Program Counter (PC, 16-bits):

It is a 16-bit Special-Purpose register. It holds **address** of the **next instruction**.
PC is incremented by the INR/DCR after every instruction byte is fetched.

Stack Pointer (SP, 16-bits):

It is a 16-bit Special-Purpose register. It holds **address** of the **top of the Stack**.
Stack is a set of memory locations operating in LIFO manner.
SP is **decremented** on every **PUSH** operation and **incremented** on every **POP**.

B, C, D, E, H, L registers 8-bits each:

These are 8-bit General-Purpose registers**.**
They can also be used to store 16-bit data in register pairs.
The possible register **pairs** are **BC** pair, **DE** pair and **HL** pair.
The **HL** pair also holds the **address** for the Memory Pointer **"M"**.

Temporary Registers (WZ, 16-bits):

This is a 16-bit register pair.
It is **used by µP** to hold **temporary** values in some instructions like CALL/JMP etc.
The **programmer** has **no access** to this register pair.

INR/DCR Register (16-bits):

This is a 16-bit shift register.
It is used to **increment PC after every instruction byte is fetched** and **increment** or **decrement SP** after a Pop or a Push operation respectively.
It is not available to the programmer.

A - Accumulator (8-bits):

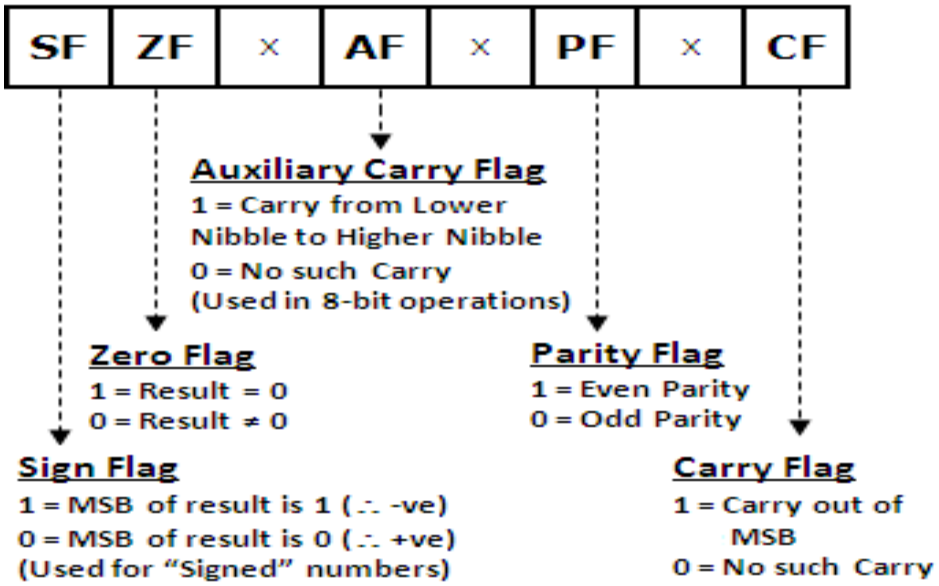
It is an 8-bit programmable register.
The user can read or write this register.
It has two **special properties**:

- It **holds one** of the **operands** during most of the arithmetic operations.
- It **holds** the **result** of most of the arithmetic and logic operations

Temp Register (8-bits):

This is an 8-bit register.
It is **used by µP** for storing one of the operands during an operation.
The **programmer** has **NO ACCESS** to this register.

FLAG REGISTER OF 8085



S - Sign Flag:

It is **set** (1) when **MSB** of the result is **1** (i.e. result is a **-**VE number).
It is reset (0) when MSB of the result is 0 (i.e. result is a **+**VE number).

Z - Zero Flag:

It is **set** when the **result** is **= zero**.
It is reset when the result is not = zero.

AC - Auxiliary Carry Flag:

It is **set** when an **Auxiliary Carry** / Borrow is **generated**.
It is reset when an Auxiliary Carry / Borrow is not generated.
Auxiliary Carry is the Carry generated **between** the **lower nibble and the higher nibble** for an 8-bit operation. It is not affected after a 16- bit operation. It is used only in DAA operation.

P - Parity Flag:

It is **set (1)** when result has **even parity**. It is reset when result has odd parity.

C - Carry Flag:

It is **set** when a **Carry / Borrow** is **generated from the MSB**.
It is reset when a Carry / Borrow is not generated from the MSB.

 # In the exam, Show at least 2 examples from Bharat Sir's lecture notes

INTERRUPTS OF 8085

This Block is responsible for controlling the **hardware interrupts** of 8085.
8085 supports the following hardware interrupts:

TRAP:

This is an **edge as well as level triggered**, **vectored** interrupt.
It cannot be masked by SIM instruction and can neither be disabled by DI instruction.
It has the **highest priority**.
Its vector address is **0024H**.

RST 7.5:

This is an **edge triggered**, **vectored** interrupt.
It can be masked by SIM instruction and can also be disabled by DI instruction.
It has the **second highest priority**.
Its vector address is **003CH**.

RST 6.5:

This is a **level triggered**, **vectored** interrupt.
It can be masked by SIM instruction and can also be disabled by DI instruction.
It has the **third highest priority**.
Its vector address is **0034H**.

RST 5.5:

This is a **level triggered**, **vectored** interrupt.
It can be masked by SIM instruction and can also be disabled by DI instruction.
It has the **fourth highest priority**.
Its vector address is **002CH**.

INTR:

This is a **level triggered**, **non-vectored** interrupt.
It cannot be masked by SIM instruction but can be disabled by DI instruction.
It has the **lowest priority**.
It has an **acknowledgement signal $\overline{\text{INTA}}$** .
The address for the ISR is **fetched from external hardware**.

$\overline{\text{INTA}}$:

This is an **acknowledgement** signal **for INTR** (only).
This signal is used to **get** the Op-Code (and hence the ISR address) from External hardware in order to execute the ISR. ☺ In case of doubts, contact Bharat Sir: - 98204 08217.

ALL Interrupts **EXCEPT TRAP** can be **disabled** though the **DI** instruction.
These interrupts can be **enabled** again by the **EI** Instruction.
Interrupts can be individually **masked or unmasked by SIM instruction**.
TRAP and INTR are not affected by SIM instruction.

SERIAL CONTROL

This Block is responsible for transferring data Serially to and from the µP.

SID - Serial In Data:
µP receives data, bit-by-bit through this line.

SOD - Serial Out Data:
µP sends out data, bit-by-bit through this line.
Serial transmission can be done by **RIM** and **SIM** Instructions.

ALU – ARITHMETIC LOGIC UNIT

8085 has an **8-bit ALU**.
It performs 8-bit arithmetic operations like Addition and Subtraction.
It also performs logical operations like AND, OR, EX-OR NOT etc.
It takes **input** from the **Accumulator** and the **Temp** register.
The **output** of most of the ALU operations is stored back **into** the **Accumulator**.

INSTRUCTION REGISTER AND DECODER

Instruction Register:
The 8085 places the contents of the PC onto the Address bus and fetches the instruction.
This fetched instruction is stored into the Instruction register.

Instruction Decoder:
The fetched instruction from the Instruction register enters the Instruction Decoder.
Here the instruction is decoded and the decode information is given to the Timing and Control Circuit where the instruction is executed..

TIMING AND CONTROL CIRCUIT

The timing and control circuit issues the various internal and external control signals for executing and instruction.
The external pins connected to this circuit are as follows:

X1 and X2:
These pins provide the **Clock Input to the µP**.
Clock is provided from a crystal oscillator.

ClkOut:
8085 provides the **Clock input** to all **other peripherals** through the ClockOut pin.
This takes care of **synchronizing** all peripherals with 8085.

$\overline{\text{ResetIn}}$:
This is an active low signal activated when the manual reset signal is applied to the µP. This signal **resets the µP**. On Reset PC contains **0000H**. Hence, the **Reset Vector Address** of 8085 is 0000H.

ResetOut:

This signal is connected to the reset input of all the peripherals.
It is used to **reset the peripherals once** the **µP** is **reset**.

READY:

This is an active high input.
It is used to **synchronize** the µP with **"Slower"** Peripherals.
The **µP samples** the **Ready** input in the beginning of every Machine Cycle.
If it is found to be **LOW**, the **µP executes** one **WAIT CYCLE** after which it re-samples the ready pin till it finds the Ready pin HIGH.
∴ The **µP remains** in the **WAIT STATE until** the **READY** pin becomes **high** again.
Hence, **if** the **Ready** pin is **not required** it should be **connected** to the **Vcc,** and not, left unconnected, **otherwise** would cause the µP to execute **infinite wait cycles**. *#Please refer Bharat Sir's Lecture Notes for this ...*

ALE - Address Latch Enable:

This signal is **used to latch address** from the multiplexed Address-Data Bus (**AD0-AD7**). When the Bus contains **address**, **ALE** is **high**, **else** it is **low**.

IO/ $\overline{M}$:

This signal is used to distinguish between an **IO** and a **Memory operation**.
When this signal is high it is an IO operation else it is a Memory operation.

$\overline{RD}$:

This is an active low signal used to indicate a **read operation**.

$\overline{WR}$:

This is an active low signal used to indicate a **write operation**.

S_1 and S_0:

These lines denote the status of the µP

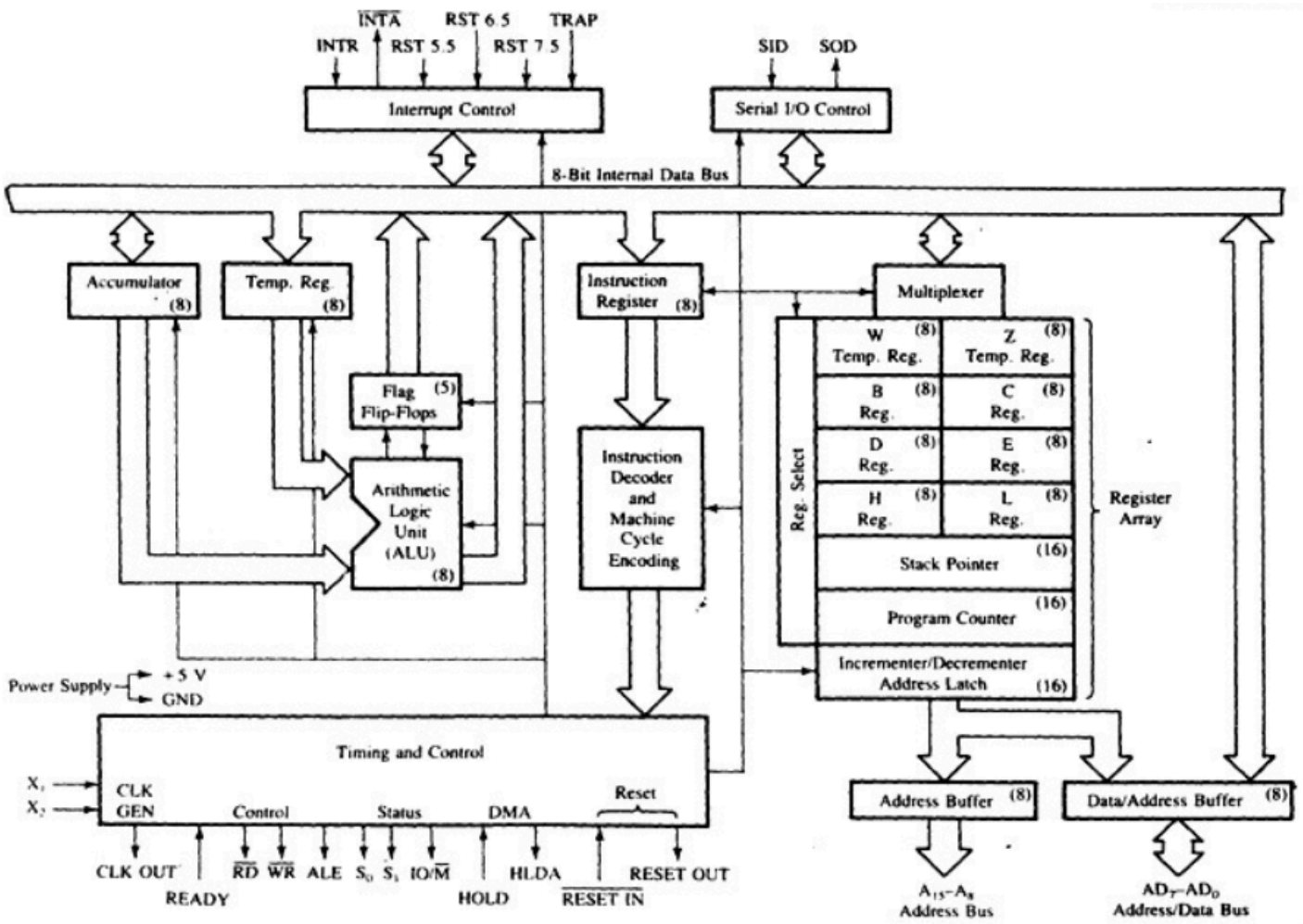
S1	S0	STATUS
0	0	Idle
0	1	Write
1	0	Read
1	1	Opcode fetch

HOLD and HLDA:

The Hold and Hold Acknowledge signals are used for **Direct Memory Access** (DMA).
The **DMA Controller issued** the **Hold** signal to the µP.
In response the **µP releases** the **System bus**.
After releasing the system bus the **µP acknowledges** the Hold signal with **HLDA** signal.
The **DMA Transfer** thus **begins**.
DMA **Transfer** is **terminated** by **releasing** the **HOLD** signal.

*Note: **"Programmer's Model"** simply means all registers in the architecture that are available to the programmer. So if they ask Programmers model, draw the internal part of the architecture without the pins, and explain all registers.*

ARCHITECTURE OF 8085



Registers

Program Counter (PC, 16-bits):

It is a 16-bit Special-Purpose register. It holds **address** of the **next instruction**.
PC is incremented by the INR/DCR after every instruction byte is fetched.

Stack Pointer (SP, 16-bits):

It is a 16-bit Special-Purpose register. It holds **address** of the **top of the Stack**.
Stack is a set of memory locations operating in LIFO manner.
SP is **decremented** on every **PUSH** operation and **incremented** on every **POP**.

B, C, D, E, H, L registers 8-bits each:

These are 8-bit General-Purpose registers**.**
They can also be used to store 16-bit data in register pairs.
The possible register **pairs** are **BC** pair, **DE** pair and **HL** pair.
The **HL** pair also holds the **address** for the Memory Pointer **"M"**.

Temporary Registers (WZ, 16-bits):

This is a 16-bit register pair.
It is **used by µP** to hold **temporary** values in some instructions like CALL/JMP etc.
The **programmer** has **no access** to this register pair.

INR/DCR Register (16-bits):

This is a 16-bit shift register.
It is used to **increment PC after every instruction byte is fetched** and **increment** or **decrement SP** after a Pop or a Push operation respectively.
It is not available to the programmer.

A - Accumulator (8-bits):

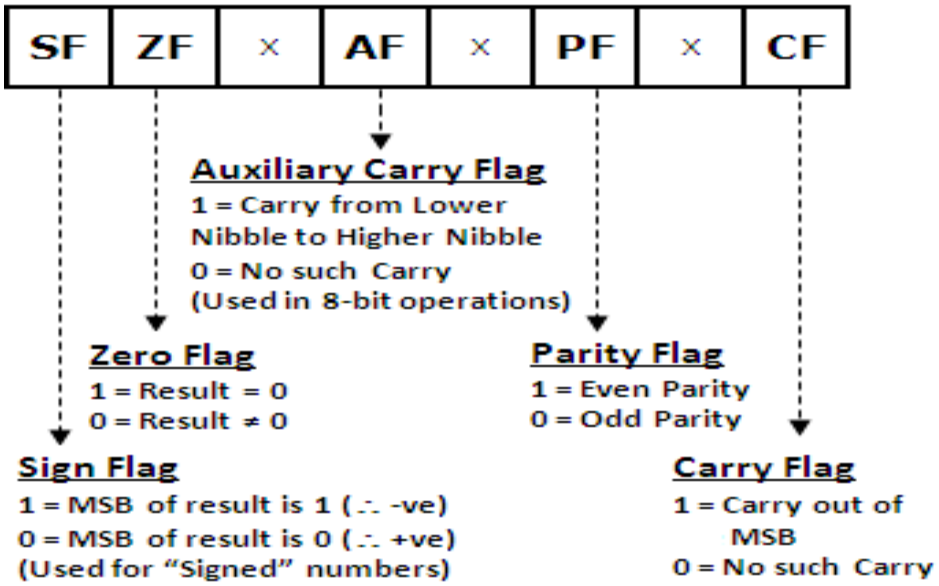
It is an 8-bit programmable register.
The user can read or write this register.
It has two **special properties**:

- It **holds one** of the **operands** during most of the arithmetic operations.
- It **holds** the **result** of most of the arithmetic and logic operations

Temp Register (8-bits):

This is an 8-bit register.
It is **used by µP** for storing one of the operands during an operation.
The **programmer** has **NO ACCESS** to this register.

FLAG REGISTER OF 8085



S - Sign Flag:

It is **set** (1) when **MSB** of the result is **1** (i.e. result is a **-**VE number).
It is reset (0) when MSB of the result is 0 (i.e. result is a **+**VE number).

Z - Zero Flag:

It is **set** when the **result** is **= zero**.
It is reset when the result is not = zero.

AC - Auxiliary Carry Flag:

It is **set** when an **Auxiliary Carry** / Borrow is **generated**.
It is reset when an Auxiliary Carry / Borrow is not generated.
Auxiliary Carry is the Carry generated **between** the **lower nibble and the higher nibble** for an 8-bit operation. It is not affected after a 16- bit operation. It is used only in DAA operation.

P - Parity Flag:

It is **set (1)** when result has **even parity**. It is reset when result has odd parity.

C - Carry Flag:

It is **set** when a **Carry / Borrow** is **generated from the MSB**.
It is reset when a Carry / Borrow is not generated from the MSB.

 # In the exam, Show at least 2 examples from Bharat Sir's lecture notes

INTERRUPTS OF 8085

This Block is responsible for controlling the **hardware interrupts** of 8085.
8085 supports the following hardware interrupts:

TRAP:

This is an **edge as well as level triggered**, **vectored** interrupt.
It cannot be masked by SIM instruction and can neither be disabled by DI instruction.
It has the **highest priority**.
Its vector address is **0024H**.

RST 7.5:

This is an **edge triggered**, **vectored** interrupt.
It can be masked by SIM instruction and can also be disabled by DI instruction.
It has the **second highest priority**.
Its vector address is **003CH**.

RST 6.5:

This is a **level triggered**, **vectored** interrupt.
It can be masked by SIM instruction and can also be disabled by DI instruction.
It has the **third highest priority**.
Its vector address is **0034H**.

RST 5.5:

This is a **level triggered**, **vectored** interrupt.
It can be masked by SIM instruction and can also be disabled by DI instruction.
It has the **fourth highest priority**.
Its vector address is **002CH**.

INTR:

This is a **level triggered**, **non-vectored** interrupt.
It cannot be masked by SIM instruction but can be disabled by DI instruction.
It has the **lowest priority**.
It has an **acknowledgement signal $\overline{\text{INTA}}$** .
The address for the ISR is **fetched from external hardware**.

$\overline{\text{INTA}}$:

This is an **acknowledgement** signal **for INTR** (only).
This signal is used to **get** the Op-Code (and hence the ISR address) from External hardware in order to execute the ISR. ☺ In case of doubts, contact Bharat Sir: - 98204 08217.

ALL Interrupts **EXCEPT TRAP** can be **disabled** though the **DI** instruction.
These interrupts can be **enabled** again by the **EI** Instruction.
Interrupts can be individually **masked or unmasked by SIM instruction**.
TRAP and INTR are not affected by SIM instruction.

SERIAL CONTROL

This Block is responsible for transferring data Serially to and from the µP.

SID - Serial In Data:
µP receives data, bit-by-bit through this line.

SOD - Serial Out Data:
µP sends out data, bit-by-bit through this line.
Serial transmission can be done by **RIM** and **SIM** Instructions.

ALU – ARITHMETIC LOGIC UNIT

8085 has an **8-bit ALU**.
It performs 8-bit arithmetic operations like Addition and Subtraction.
It also performs logical operations like AND, OR, EX-OR NOT etc.
It takes **input** from the **Accumulator** and the **Temp** register.
The **output** of most of the ALU operations is stored back **into** the **Accumulator**.

INSTRUCTION REGISTER AND DECODER

Instruction Register:
The 8085 places the contents of the PC onto the Address bus and fetches the instruction.
This fetched instruction is stored into the Instruction register.

Instruction Decoder:
The fetched instruction from the Instruction register enters the Instruction Decoder.
Here the instruction is decoded and the decode information is given to the Timing and Control Circuit where the instruction is executed..

TIMING AND CONTROL CIRCUIT

The timing and control circuit issues the various internal and external control signals for executing and instruction.
The external pins connected to this circuit are as follows:

X1 and X2:
These pins provide the **Clock Input to the µP**.
Clock is provided from a crystal oscillator.

ClkOut:
8085 provides the **Clock input** to all **other peripherals** through the ClockOut pin.
This takes care of **synchronizing** all peripherals with 8085.

$\overline{\text{ResetIn}}$:
This is an active low signal activated when the manual reset signal is applied to the µP. This signal **resets the µP**. On Reset PC contains **0000H**. Hence, the **Reset Vector Address** of 8085 is 0000H.

ResetOut:

This signal is connected to the reset input of all the peripherals.
It is used to **reset the peripherals once** the **µP** is **reset**.

READY:

This is an active high input.
It is used to **synchronize** the µP with **"Slower"** Peripherals.
The **µP samples** the **Ready** input in the beginning of every Machine Cycle.
If it is found to be **LOW**, the **µP executes** one **WAIT CYCLE** after which it re-samples the ready pin till it finds the Ready pin HIGH.
∴ The **µP remains** in the **WAIT STATE until** the **READY** pin becomes **high** again.
Hence, **if** the **Ready** pin is **not required** it should be **connected** to the **Vcc,** and not, left unconnected, **otherwise** would cause the µP to execute **infinite wait cycles**. *#Please refer Bharat Sir's Lecture Notes for this ...*

ALE - Address Latch Enable:

This signal is **used to latch address** from the multiplexed Address-Data Bus (**AD0-AD7**). When the Bus contains **address**, **ALE** is **high**, **else** it is **low**.

IO/ $\overline{M}$:

This signal is used to distinguish between an **IO** and a **Memory operation**.
When this signal is high it is an IO operation else it is a Memory operation.

$\overline{RD}$:

This is an active low signal used to indicate a **read operation**.

$\overline{WR}$:

This is an active low signal used to indicate a **write operation**.

S_1 and S_0:

These lines denote the status of the µP

S1	S0	STATUS
0	0	Idle
0	1	Write
1	0	Read
1	1	Opcode fetch

HOLD and HLDA:

The Hold and Hold Acknowledge signals are used for **Direct Memory Access** (DMA).
The **DMA Controller issued** the **Hold** signal to the µP.
In response the **µP releases** the **System bus**.
After releasing the system bus the **µP acknowledges** the Hold signal with **HLDA** signal.
The **DMA Transfer** thus **begins**.
DMA **Transfer** is **terminated** by **releasing** the **HOLD** signal.

*Note: **"Programmer's Model"** simply means all registers in the architecture that are available to the programmer. So if they ask Programmers model, draw the internal part of the architecture without the pins, and explain all registers.*

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

ADDRESSING MODES OF 8085

Addressing Modes are the different ways by which the µP address (specifies) the operands in an instruction. 8085 supports the following Addressing Modes:

1) Immediate Addressing Mode

In this mode, the **Data** is specified **in** the **Instruction** itself.

```
Eg:   MVI A, 35H      ; Move immediately the value 35 into the Accumulator.
                      ; i.e. A ← 35H
      LXI B, 4000H    ; Move immediately the value 4000 into the register pair BC.
                      ; i.e. BC ← 4000H
```

Advantage:
Programmer can easily **identify** the **operands**.

Disadvantage:
Always more than one byte hence requires **more space**.
The µP requires **two or three machine cycles** to fetch the instruction hence **slow**.

2) Register Addressing Mode

In this mode, the **Data** is specified **in Registers**.

```
Eg:   MOV B, C        ; Move the Contents of C-Register into B-Register.
                      ; i.e. B ← C
      INR B           ; Increments the contents of B-Register.
                      ; i.e. B ← B + 1
```

Advantage:
The µP requires **only one machine cycle** to Fetch the instruction.

Disadvantage:
Operands **cannot** be easily **identified**.

3) Direct Addressing Mode

In this mode, the **Address** of the operand is specified **in** the **Instruction** itself.

```
Eg:   LDA 2000H       ; Loads the Accumulator with the Contents of Location 2000.
                      ; i.e. A ← [2000]
      STA 2000H       ; Stores the Contents of the Accumulator at the Location 2000.
                      ; i.e. [2000] ← A
```

Advantage:
The programmer **can identify** the address of the operand.

Disadvantage:
These are **three byte instructions** hence require three fetch cycles.

Cell: 98204 08217
BharatSir@hotmail.com

4) Indirect Addressing Mode

In this mode, the **Address** of the operand is specified **in Registers.**
Hence, the instruction indirectly points to the operands.
Even the Memory Pointer "M" can be used as it is pointed by the HL register pair.

```
Eg:   STAX B      ; Stores the contents of the Accumulator at the location
                  ; pointed by the contents of BC pair.
                  ; i.e. [[BC]] ← A.
                  ; So if contents of BC pair = 4000 i.e. [BC] = 4000 then
                  ; [4000] ← A. #Please refer Bharat Sir's Lecture Notes for  this ...
      INR M       ; Increments the contents of the location pointed by HL pair
                  ; (i.e. M) i.e. [[HL]] ← [[HL]] + 1
```

Advantage:
Address of the operand is **not fixed** and hence can be used in a **loop**.
Size of the instruction is **small** as compared to direct addressing mode.

Disadvantage:
Requires initialization of the register pair hence requires atleast one more instruction.

5) Implied Addressing Mode

In this mode, the **Operand** is **implied** in the instruction.
This instruction will work only on that implied operand, and not on any other operand.

```
Eg:   STC         ; Sets the Carry Flag in the Flag register.
                  ; Cy ← 1.
      CMC         ; Complements the Carry Flag in the Flag register.
```

Advantage:
Instructions are generally **only one byte**.

Disadvantage:
Programmer **cannot** easily **identify** the value of the operand.

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

INSTRUCTION SET OF 8085

Some Common Notations:

1) Addr → 16 bit address.
2) Data → 8 bit data.
3) Data 16 → 16 bit data.
4) R, r1, r2 → one of the registers.
5) Rp → register pair. BC pair is called B, DE → D and HL → L
6) Port → 8 bit IO addres

Data Transfer Group

1) MOV r1, r2

The contents of register r2 is moved into register r1.

Eg: MOV A,B ; A ← B

Addr. Mode	Flags Affected	Cycles	T-States
Register	None	1	4

2) MOV r1, M

The contents of the memory location pointed by HL (memory pointer) is moved into register r1.

Eg: MOV B,M ; B ← [[HL]] i.e. B ← M

Addr. Mode	Flags Affected	Cycles	T-States
Indirect	None	2	7

3) MOV M, r2

The contents of register r2 is moved into the memory location pointed by HL (memory pointer).

Eg: MOV M,B ; [[HL]] ← B i.e. M ← B

Addr. Mode	Flags Affected	Cycles	T-States
Register	None	2	7

4) MVI r1, 8-bit data

The 8-bit data is immidiately moved into the register specified in the instruction.

Eg: MVI C, 23 ; C ← 23H

Addr. Mode	Flags Affected	Cycles	T-States
Immidiate	None	2	7

5) LXI rp, 16-bit data

The 16-bit data is immidiately moved into the register pair specified in the instruction.

Eg: MVI B, 2300H ; BC ← 2300H i.e. B← 23, C← 00

Addr. Mode	Flags Affected	Cycles	T-States
Immidiate	None	3	10

6) MVI M, 8-bit data

The 8-bit data is immidiately moved into the memory location pointed by HL (memory pointer).

Eg: MVI M, 23 ; [[HL]] ← 23H

Addr. Mode	Flags Affected	Cycles	T-States
Immediate	None	3	10

Cell: 98204 08217
BharatSir@hotmail.com

7) LDA 16-bit address

The accumulator is loaded withb the contents of the memory location having the given address.
Eg: LDA 2000H ; A ← [2000]

Addr. Mode	Flags Affected	Cycles	T-States
Direct	None	4	13

8) STA 16-bit address

The accumulator is stored into the memory location having the given address.
Eg: STA 2000H ; [2000] ← A

Addr. Mode	Flags Affected	Cycles	T-States
Direct	None	4	13

9) LHLD 16-bit address

The HL pair is loaded with the contents of the loactions ppointed by the given address and address + 1. ☺ In case of doubts, contact Bharat Sir: - 98204 08217.
Eg: LHLD 2000 ; HL ← [2000] & [2001] i.e. L ← [2000], H ← [2001]

Addr. Mode	Flags Affected	Cycles	T-States
Direct	None	5	16

10) SHLD 16-bit address

The HL pair is stored into the loactions ppointed by the given address and address + 1.
Eg: SHLD 2000 ; [2000] & [2001] ← HL i.e. [2000] ← L, [2001] ← H

Addr. Mode	Flags Affected	Cycles	T-States
Direct	None	5	16

11) LDAX rp

The accumulator is loaded with the contents of memory location pointed by value of the given register.
Eg: LDAX B ; A ← [[BC]] i.e. if [BC] = 2000, A gets the value from location
; 2000 i.e. A ← [2000]

Addr. Mode	Flags Affected	Cycles	T-States
Indirect	None	2	7

12) STAX rp

The accumulator is stored into the location pointed by value of the given register.
Eg: STAX B ; [[BC]] ← A i.e. if [BC] = 2000, location 2000 will get the
; value of A i.e. [2000] ← A.

Addr. Mode	Flags Affected	Cycles	T-States
Indirect	None	2	7

13) PCHL

The Program Counter gets the contents of the HL register pair.
This statement causes a branch in the sequence of the program.
Eg: PCHL ; PC ← HL

Addr. Mode	Flags Affected	Cycles	T-States
Register	None	1	**6**

14) SPHL

The Stack Pointer gets the contents of the HL register pair.
This statement reloates the stack in the 64 KB memory.

Eg: SPHL ; SP ← HL

Addr. Mode	Flags Affected	Cycles	T-States
Register	None	1	**6**

15) XCHG

This instruction exchanges the contents of HL pair and DE pair.

Eg: XCHG ; HL ←→ DE

#Please refer Bharat Sir's Lecture Notes for this ...

Addr. Mode	Flags Affected	Cycles	T-States
Register	None	1	4

16) XTHL

This instruction exchanges the DE pair with the contents of location pointed by the SP and SP+1.

Eg: XTHL ; HL ←→ [[SP]] and [[SP]+1]
; i.e. if [SP]=2000 then L ←→ [2000] and H ←→ [2001]

#Please refer Bharat Sir's Lecture Notes for this ...

Addr. Mode	Flags Affected	Cycles	T-States
Register	None	5	16

Arithmetic Group

Addition

1) ADD R

This instruction adds the contents of register R with the accumulator, stores result in the accumulator.

Eg: ADD B ; A ← A + B

Addr. Mode	Flags Affected	Cycles	T-States
Register	All	1	**4**

2) ADD M

This instruction adds the contents of the memory location pointed by HL, with the accumulator, and stores the result in the accumulator.

Eg: ADD M ; A ← A + [[HL]]

Addr. Mode	Flags Affected	Cycles	T-States
Indirect	All	2	**7**

3) ADI 8-bit data

This instruction adds the immidiate data with the accumulator, and stores the result in the accumulator.

Eg: ADI 25 ; A ← A + 25

Addr. Mode	Flags Affected	Cycles	T-States
Immediate	All	2	**7**

4) ADC R

This instruction adds the contents of the register R with the accumulator, and also adds the carry flag, and stores the result in the accumulator. It is used while adding large numbers.

Eg: ADD B ; A ← A + B + Cy

Addr. Mode	Flags Affected	Cycles	T-States
Register	All	1	**4**

5) ADC M

Eg: SPHL ; SP ← HL

Addr. Mode	Flags Affected	Cycles	T-States
Register	None	1	**6**

15) XCHG

This instruction exchanges the contents of HL pair and DE pair.

Eg: XCHG ; HL ←→ DE

#Please refer Bharat Sir's Lecture Notes for this ...

Addr. Mode	Flags Affected	Cycles	T-States
Register	None	1	4

16) XTHL

This instruction exchanges the DE pair with the contents of location pointed by the SP and SP+1.

Eg: XTHL ; HL ←→ [[SP]] and [[SP]+1]
; i.e. if [SP]=2000 then L ←→ [2000] and H ←→ [2001]

#Please refer Bharat Sir's Lecture Notes for this ...

Addr. Mode	Flags Affected	Cycles	T-States
Register	None	5	16

Arithmetic Group

Addition

1) ADD R

This instruction adds the contents of register R with the accumulator, stores result in the accumulator.

Eg: ADD B ; A ← A + B

Addr. Mode	Flags Affected	Cycles	T-States
Register	All	1	**4**

2) ADD M

This instruction adds the contents of the memory location pointed by HL, with the accumulator, and stores the result in the accumulator.

Eg: ADD M ; A ← A + [[HL]]

Addr. Mode	Flags Affected	Cycles	T-States
Indirect	All	2	**7**

3) ADI 8-bit data

This instruction adds the immidiate data with the accumulator, and stores the result in the accumulator.

Eg: ADI 25 ; A ← A + 25

Addr. Mode	Flags Affected	Cycles	T-States
Immediate	All	2	**7**

4) ADC R

This instruction adds the contents of the register R with the accumulator, and also adds the carry flag, and stores the result in the accumulator. It is used while adding large numbers.

Eg: ADD B ; A ← A + B + Cy

Addr. Mode	Flags Affected	Cycles	T-States
Register	All	1	**4**

5) ADC M

Cell: 98204 08217
BharatSir@hotmail.com

This instruction adds the contents of the memory location pointed by HL, with the accumulator, and also adds the carry flag, and stores the result in the accumulator.
Eg: ADD M ; A ← A + [[HL]] + Cy

Addr. Mode	Flags Affected	Cycles	T-States
Indirect	All	2	**7**

6) ACI 8-bit data
This instruction adds the immidiate data with the accumulator, and also adds the carry flag, and stores the result in the accumulator.
Eg: ACI 25 ; A ← A + 25 +Cy

Addr. Mode	Flags Affected	Cycles	T-States
Immediate	All	2	**7**

Subtraction
Simillarly subtraction is also done as above.

7) SUB R
8) SUB M
9) SUI 8-bit data
10) SBB R
11) SBB M
12) SBI 8-bit data

Increment

13) INR R
This instruction increments the contents of the specified register.
The incremented value is stored back in the same register.
Eg: INR B ; B ← B + 1

Addr. Mode	Flags Affected	Cycles	T-States
Register	All except carry	1	4

14) INR M
This instruction increments the contents of memory location pointed by HL pair.
The incremented value is stored back at the same location.
Eg: INR M ; M ← M + 1 i.e. [[HL]] ← [[HL]] +1

Addr. Mode	Flags Affected	Cycles	T-States
Indirect	All except carry	3	10

15) INX Rp
This instruction increments the contents of the specified register **pair**.
The incremented value is stored back in the same register **pair**.
Eg: INX BC ; BC ← BC + 1 i.e. if [BC]=3000 then [BC] becomes 3001.

Addr. Mode	Flags Affected	Cycles	T-States
Register	**NONE**	1	**6**

Decrement

16) DCR R
This instruction decrements the contents of the specified register.
The decremented value is stored back in the same register.
Eg: DCR B ; B ← B + 1

Addr. Mode	Flags Affected	Cycles	T-States

Cell: 98204 08217
BharatSir@hotmail.com

Register	All except carry	1	4

17) DCR M

This instruction decrements the contents of memory location pointed by HL pair.
The decremented value is stored back at the same location.
Eg: DCR M ; M ← M - 1 i.e. [[HL]] ← [[HL]] - 1

Addr. Mode	Flags Affected	Cycles	T-States
Indirect	All except carry	3	10

18) DCX Rp

This instruction decrements the contents of the specified register **pair**.
The decremented value is stored back in the same register **pair**.
Eg: DCX BC ; BC ← BC - 1 i.e. if [BC]=3001 then [BC] becomes 3000.

Addr. Mode	Flags Affected	Cycles	T-States
Register	**NONE**	1	**6**

Others

19) DAD Rp

This instruction adds the contents of the given register pair with HL pair.
The result is stored in the HL pair.
Eg: DAD B ; HL ← HL + BC

Addr. Mode	Flags Affected	Cycles	T-States
Register	**Only Carry**	3	**10**

20) DAA

This instruction is used to get the answer in BCD form.
This instruction does the Following actions :
#Please refer Bharat Sir's Lecture Notes for this ...

Addr. Mode	Flags Affected	Cycles	T-States
Implied	**ALL**	1	**4**

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

Logic Group

AND

1) ANA R

Logically AND the contents of the specified register with accumulator, store result in accumulator.

Eg: ANA B ; A ← A AND B

Addr. Mode	Flags Affected	Cycles	T-States
Register	**ALL**	1	**4**

2) ANA M

Logically AND the contents of the memory location pointed by HL pair, with the accumulator.

Eg: ANA M ; A ← A AND M

Addr. Mode	Flags Affected	Cycles	T-States
Indirect	**ALL**	2	**7**

3) ANI 8-bit data

Logically AND the immidiate 8-bit data, with the accumulator.

Eg: ANA 25 ; A ← A AND 25

Addr. Mode	Flags Affected	Cycles	T-States
Immidiate	**ALL**	2	**7**

Simillarly we have the other logical instrctions as follows:

OR

4) ORA R
5) ORA M
6) ORI 8-bit data

X-OR

7) XRA R
8) XRA M
9) XRI 8-bit data

Important Note (Use of Logic Instructions):

To **"Clear any bit"**, we must **"AND" that bit with "0"** and the remaining bits with "1".
Eg: ANI F0H will Clear the Lower Nibble of A while the Higher Nibble will remain the same.

To **"Set any bit"**, we must **"OR" that bit with "1"** and the remaining bits with "0".
Eg: ORI 0FH will Set the Lower Nibble of A while the Higher Nibble will remain the same.

To **"Complement any bit"**, we must **"XOR" that bit with "1"** and the remaining bits with "0".
Eg: XRI 0FH will Complenet the Lower Nibble of A while the Higher Nibble will remain the same.

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

Compare

10) CMP R

Compares the contents of register R and accumulator.
Comparision essntially is subtraction. Hence, this instruction performs A – R.
It is very important to **remember** that the **result** of this comparision is **NOT stored** in **accumulator**, only the Flags are afftected. ☺ In case of doubts, contact Bharat Sir: - 98204 08217.
Eg: CMP B ; Compares A and B i.e. A – B (and not B - A)
We decide which one of the two is greater by checking the flags affected as follows:

Conclusion	Zero Flag 'Z'	Carry Flag 'Cy'
A > B	0	0
A = B	1	0
A < B	0	1

Addr. Mode	Flags Affected	Cycles	T-States
Register	**ALL**	1	**4**

Simillarly we have the other comparision instrctions as follows:
11) CMP M
12) CPI 8-bit data

13) STC

Sets the carry flag.
Cy ← 1.

Addr. Mode	Flags Affected	Cycles	T-States
Implied	**Only Carry**	1	**4**

14) CMC

Complements the carry flag.
Cy ← Cy.

Addr. Mode	Flags Affected	Cycles	T-States
Implied	**Only Carry**	1	**4**

15) CMA

Complements the accumulator.
A ← 1's complement of A.

Addr. Mode	Flags Affected	Cycles	T-States
Implied	**None**	1	**4**

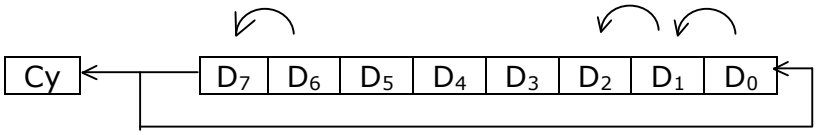
BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

Rotate Instructions

16) RLC

The Contents of accumulator are rotated left by 1.
The MSB goes to the Carry AND the LSB.

Carry **Accumulator**

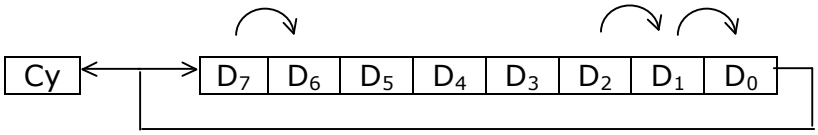


Addr. Mode	Flags Affected	Cycles	T-States
Implied	**Carry**	1	**4**

17) RRC

The Contents of accumulator are rotated right by 1.
The LSB goes to the Carry AND the MSB.

Carry **Accumulator**

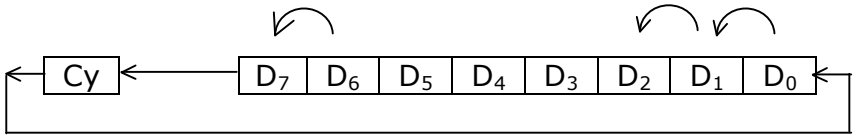


Addr. Mode	Flags Affected	Cycles	T-States
Implied	**Carry**	1	**4**

18) RAL

The Contents of accumulator are rotated left by 1.
The MSB goes to the Carry and THE CARRY goes to LSB.

Carry **Accumulator**

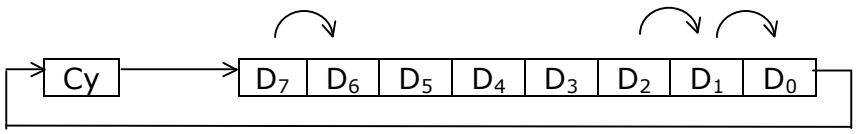


Addr. Mode	Flags Affected	Cycles	T-States
Implied	**Carry**	1	**4**

19) RAR

The Contents of accumulator are rotated right by 1.
The LSB goes to the Carry and the CARRY goes to the MSB.

Carry **Accumulator**



Addr. Mode	Flags Affected	Cycles	T-States
Implied	**Carry**	1	**4**

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

Branch Group

Conditions

Condition	Description	True if:
NZ	No Zero	Z=0
Z	Zero	**Z=1**
NC	No Carry	C=0
C	Carry	**C=1**
PO	Parity Odd	P=0
PE	Parity Even	**P=1**
P	Plus	S=0
M	Minus	**S=1**

1) JMP 16-bit address (Unconditional Jump)
Loads PC with the 16-bit address specified in the instruction.
Eg: JMP 2500 ; PC ← 2500

Addr. Mode	Flags Affected	Cycles	T-States
Immidiate	**None**	3	**10**

2) J Condition **16-bit address (Conditional Jump)**
It is the same as UnConditional JUMP except that the action takes place ONLY if the condition is true.
Eg: JZ 2500 ; PC ← 2500 if Z =1

Addr. Mode	Flags Affected	Cycles	T-States
Immidiate	**None**	2/3	**7/10**

3) CALL 16-bit address (Unconditional Call) ***--- Very Imp.***
Loads PC with the 16-bit address specified in the instruction.
Before doing so, it also Pushes the Current PC into the Stack.
Eg: JMP 2500 ; SP ← SP – 1
[SP] ← PC_H
SP ← SP - 1
[SP] ← PC_L
PC ← 2500
#Please refer Bharat Sir's Lecture Notes for this ...

Addr. Mode	Flags Affected	Cycles	T-States
Immidiate	**None**	5	**18**

4) C Condition **16-bit address (Conditional Call)**
It is the same as UnConditional Call except that the action takes place ONLY if the condition is true.
Eg: CNZ 2500 ; IF Z = 0 then
SP ← SP – 1
[SP] ← PC_H
SP ← SP - 1
[SP] ← PC_L
PC ← 2500
#Please refer Bharat Sir's Lecture Notes for this ...

Addr. Mode	Flags Affected	Cycles	T-States
Immidiate	**None**	2/5	**9/18**

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

5) RET (Unconditional Return)

This instruction is written at the end of the sub-routine and enables the control to go back to the main program.

We enter a subroutine using CALL instruction in which we push the return address into the stack.

In RET instruction we do the opposite i.e. we POP the return address from the stack into PC.

Eg: RET ; $PC_L \leftarrow [SP]$
$SP \leftarrow SP + 1$
$PC_H \leftarrow [SP]$
$SP \leftarrow SP + 1$

#Please refer Bharat Sir's Lecture Notes for this ...

Addr. Mode	Flags Affected	Cycles	T-States
Indirect	**None**	3	**10**

6) R Condition **(Conditional Return)**

It is the same as UnConditional Return except that the action takes place ONLY if the condition is true.

Eg: RC ; IF C = 1 then
$PC_L \leftarrow [SP]$
$SP \leftarrow SP + 1$
$PC_H \leftarrow [SP]$
$SP \leftarrow SP + 1$

#Please refer Bharat Sir's Lecture Notes for this ...

Addr. Mode	Flags Affected	Cycles	T-States
Indirect	**None**	1/3	**6/12**

7) RSTn (Restart n)

This instruction is very similar to the CALL instruction.

Here the branch address, instead of being specified directly in the instruction, is calculated as $(n \times 8)$.

The current PC is Pushed into the stack.

The new value of PC is $(n \times 8)$.

The value of n = 0,1,2 … 7.

These instructions are called as Software Interrupts.

Operationally it is thus simillar to CALL except that it is a **1 byte instruction**.

Eg: RST1 ; $SP \leftarrow SP - 1$
$[SP] \leftarrow PC_H$
$SP \leftarrow SP - 1$
$[SP] \leftarrow PC_L$
$PC \leftarrow 0008$ ($\because 1 \times 8 = 0008$)

#Please refer Bharat Sir's Lecture Notes for this ...

Addr. Mode	Flags Affected	Cycles	T-States
Indirect	**None**	3	**12**

***Please Note**: PCHL also causes a branch in the program flow (to the location pointed by the HL Pair), but is already included in the data transfer group. It is a **very important** instruction and during programming, you should remember that it can also cause a branch.*

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

Stack, I/O and Machine Control Instructions

Stack Instructions

1) PUSH Rp

It pushes the given register pair into the stack.
Remember, PUSH is the only operation (not the only instruction), that accesses the higher byte of a 2 byte number before the lower byte. This is because the lower byte needs to be accessed first during POP and the stack operates in LIFO.
Also remember that NO stack operation uses 8-bit operands so we cannot push a single register, we have to push a register pair.

Eg: PUSH B ; SP ← SP – 1
[SP] ← B
SP ← SP - 1
[SP] ← C

Addr. Mode	Flags Affected	Cycles	T-States
Register	**None**	3	**12**

2) PUSH PSW

It pushes the PSW (Program Status Word) into the stack.
The PSW is the combination of the accumulator and the Flag register, accumulator being the higher byte.
∴ **PSW → AF**
PSW can **ONLY** be used in PUSH and POP instructions.

Eg: PUSH PSW ; SP ← SP – 1
[SP] ← A
SP ← SP - 1
[SP] ← F

Addr. Mode	Flags Affected	Cycles	T-States
Register	**None**	3	**12**

3) POP Rp

It pops the top 2 elements (2 × 8-bit ∴ 16-bit) from the stack into the given register pair.
The lower byte comes out first as the higher byte was pushed in first and stack operates in LIFO manner.

Eg: POP B ; C ← [SP]
SP← SP + 1
B ← [SP]
SP← SP + 1

Addr. Mode	Flags Affected	Cycles	T-States
Register	**None**	3	**10**

4) POP PSW

It pops the top 2 elements (2 × 8-bit ∴ 16-bit) from the stack into the PSW.

Eg: POP PSW ; F ← [SP]
SP← SP + 1
A ← [SP]
SP← SP + 1

Addr. Mode	Flags Affected	Cycles	T-States
Register	**None**	3	**10**

Please Note: There are other instructions like XTHL, SPHL, LXI SP, CALL RET etc which directly or indirectly affect the stack and are already included in various groups above.

Cell: 98204 08217
BharatSir@hotmail.com

Input/Output

5) IN 8-bit I/O Port address

8085 has 256 I/O Ports having 8-bit addresses 00H … FFH.
This instruction is used to read data from an I/O Port, whose address is given in the instruction.
This data can be read into the Accuulator ONLY.
Eg: IN 80 ; A ← $[80]_{I/O}$

Addr. Mode	Flags Affected	Cycles	T-States
Direct	**None**	3	**10**

6) OUT 8-bit I/O Port address

This instruction is used to send data from the accumulator to an I/O Port, whose address is in the instruction.
This data can be sent from the Accuulator ONLY.
Eg: OUT 80 ; $[80]_{I/O}$ ← A

Addr. Mode	Flags Affected	Cycles	T-States
Direct	**None**	3	**10**

Machine Control Instructions

7) SIM (Set Interrupt Mask)

This instruction is used to set the interrupt masking pattern for the µP.
The appropriate bit pattern is **loaded into** the accumulator an then this instruction is executed.
It is basically used to **mask/unmask** the **interrupts** except TRAP and INTR.
It can also be used to send a '**bit**' out through the serial out pin **SID**.
Eg: SIM ; µP accepts the masking pattern through the accumulator.

Addr. Mode	Flags Affected	Cycles	T-States
Implied	**None**	1	**4**

8) RIM (Read Interrupt Mask)

This instruction is used to read the interrupt masking pattern for the µP.
After executing this instruction, the µP loads the bit pattern **into** the accumulator.
It can also be used to receive a '**bit**' out through the serial in pin **SID**.
Eg: RIM ; µP loads the masking pattern into the accumulator.

Addr. Mode	Flags Affected	Cycles	T-States
Implied	**None**	1	**4**

Please Note: For the bit patterns of SIM and RIM instruction, please refer the chapter on Interrupts.

9) EI (Enable Interrupts)

This instruction is used to enable the interrupts in the µP.
This instrusetion sets the INTE flip-flop (Interrupt Enable Flip-Flop).
This instruction effects all the interrupts except TRAP.
Eg: EI ; $INTE_{F/F}$ ← 1

Addr. Mode	Flags Affected	Cycles	T-States
Implied	**None**	1	**4**

10) DI (Disable Interrupts)

This instruction is used to disable the interrupts in the µP.
This instrusetion resets the INTE flip-flop.
This instruction effects all the interrupts except TRAP.
Eg: DI ; $INTE_{F/F}$ ← 0

Cell: 98204 08217
BharatSir@hotmail.com

Addr. Mode	Flags Affected	Cycles	T-States
Implied	**None**	1	**4**

11) NOP (No Operation)

This instruction performs no operation, but consumes time of the µP.
It is the simplest method of causing a software delay.

Eg: NOP ; ----------

Addr. Mode	Flags Affected	Cycles	T-States
Implied	**None**	1	**4**

12) HLT (Halt)

This instruction signifies the end of the program.
It causes the µP to stop fetching any further instruction, hence program execution is stopped.
It makes the Halt Flip Flop inside the µP = 1.
µP checks the Halt Flip Flop in the beginning (1^{st} T-State) of the next Machine Cycle and Stops all operations.

Eg: HLT ; Halt Flip-Flop ← 1

Addr. Mode	Flags Affected	Cycles	T-States
Implied	**None**	1 + 1T	**5**

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

MACHINE CYCLES AND TIMING DIAGRAMS

Instruction Cycle:
This is the time required by the µP to fetch and execute one complete instruction.
The instruction cycle is in two parts:
1. Fetch Cycle
2. Execute Cycle

Fetch Cycle:
This is the time required by the µP to fetch all bytes of an instruction
The length of the fetch cycle is thus determined by the no of bytes in an instruction.

Execution Cycle:
This is the time required by the µP to execute a fetched instruction.

T-State:
A T-State is one clock cycle of the µP.
∴T = Clock Period = 1/Clock Frequency

Machine Cycle:
It is the time required by the µP doing one operation and accessing one byte from the external module (Memory or I/O)

Machine Cycles of 8085

The Machine cycles of 8085 are given below:

Name	IO/M	RD	WR	S1	S0	INTA	T-States
Opcode Fetch	0	0	1	1	1	1	**4/6**
Mem Read	0	0	1	1	0	1	3
Mem Write	0	1	0	0	1	1	3
IO Read	1	0	1	1	0	1	3
IO Write	1	1	0	0	1	1	3
Int. Acknowledge	1	1	1	1	1	0	**3 or 6**
Bus Idle	0	1	1	0	0	1	3

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

Opcode Fetch

- This cycle is used to **fetch** the **Opcode from** the **memory**.
- This is the **First** Machine Cycle of every instruction.
- It is a **compulsory** Machine Cycle
- It is **generally** of **4 T**-States but **some** instructions require a **6 T**-State Opcode Fetch.

During T1

- **A15-A8** contains the higher byte of the address (**PCH**)
- As **ALE** is **high AD7-AD0** contains the lower byte of the address(**PCL**).
- Since it is an Opcode fetch cycle, **S1** and **S0** go **high**.
- Since it is a memory operation, **IO/M** goes **low**.

During T2

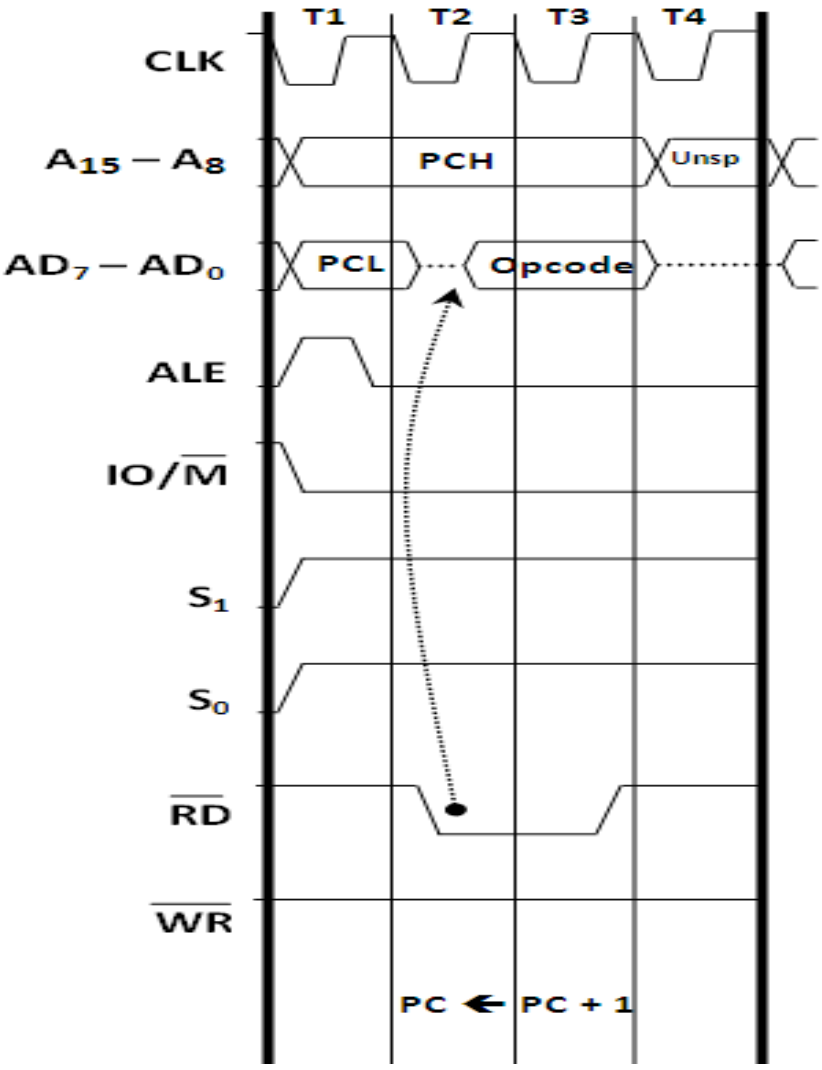
- As **ALE** goes **low** address is removed from AD7-AD0.
- As **RD** goes **low**, **data** appears **on AD7-AD0**. ☺ In case of doubts, contact Bharat Sir: - 98204 08217.
- The µP examines the state of the READY pin. If it is low then the µP enters wait-state by executing wait cycles and remains in the wait-state until READY goes high.

During T3

- Data remains on AD7-AD0 till RD is low.

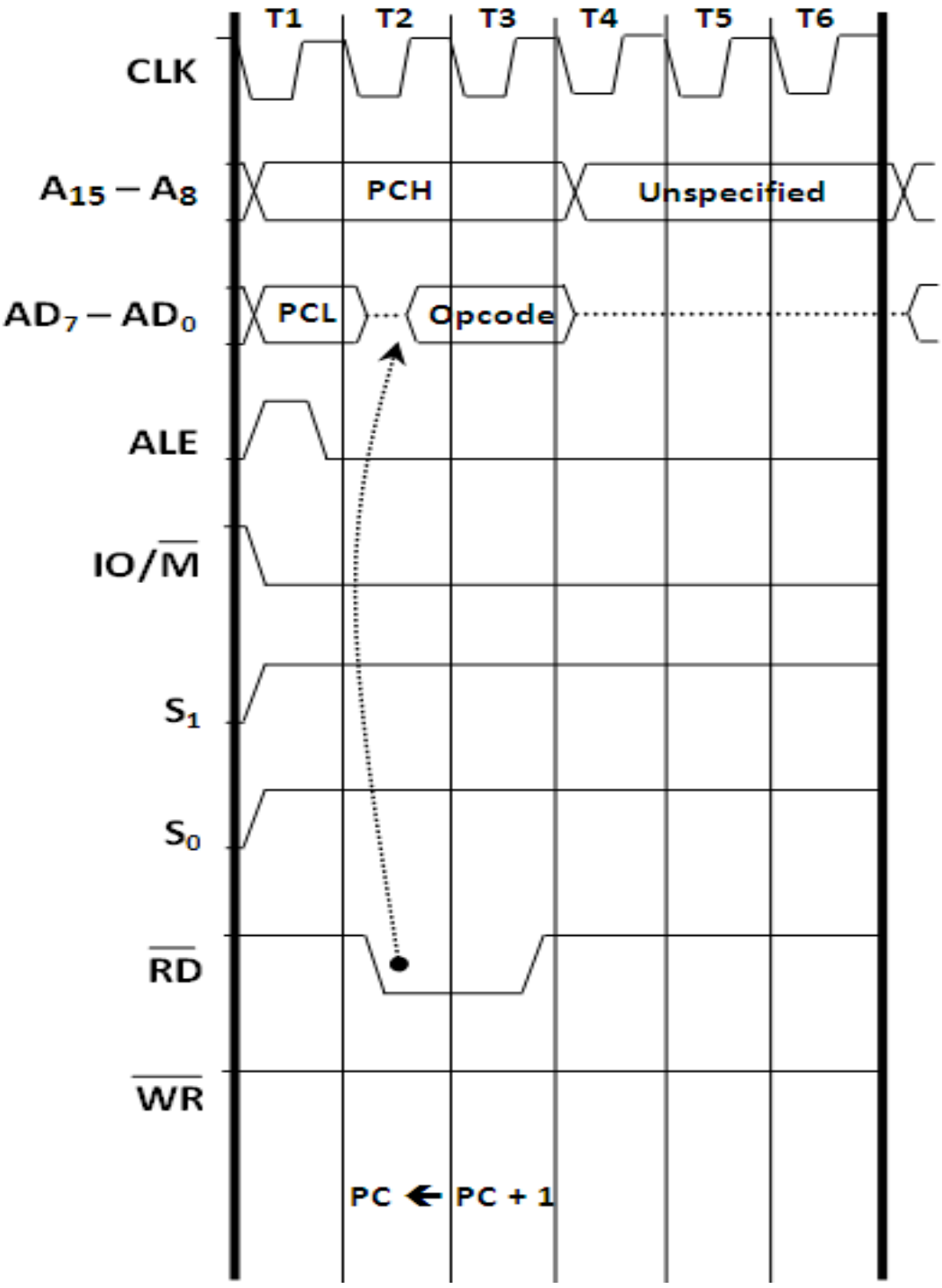
During T4

- T4 state is used by the µP to decode the Opcode.



BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

Opcode Fetch of 6 T-States



BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

Memory Read

- This cycle is used to **fetch one byte from** the **memory**.
- This cycle can be used to fetch the operand bytes of an instruction or any data from the memory.
- It is a not compulsory Machine Cycle
- It requires **3 T-States**.

During T1

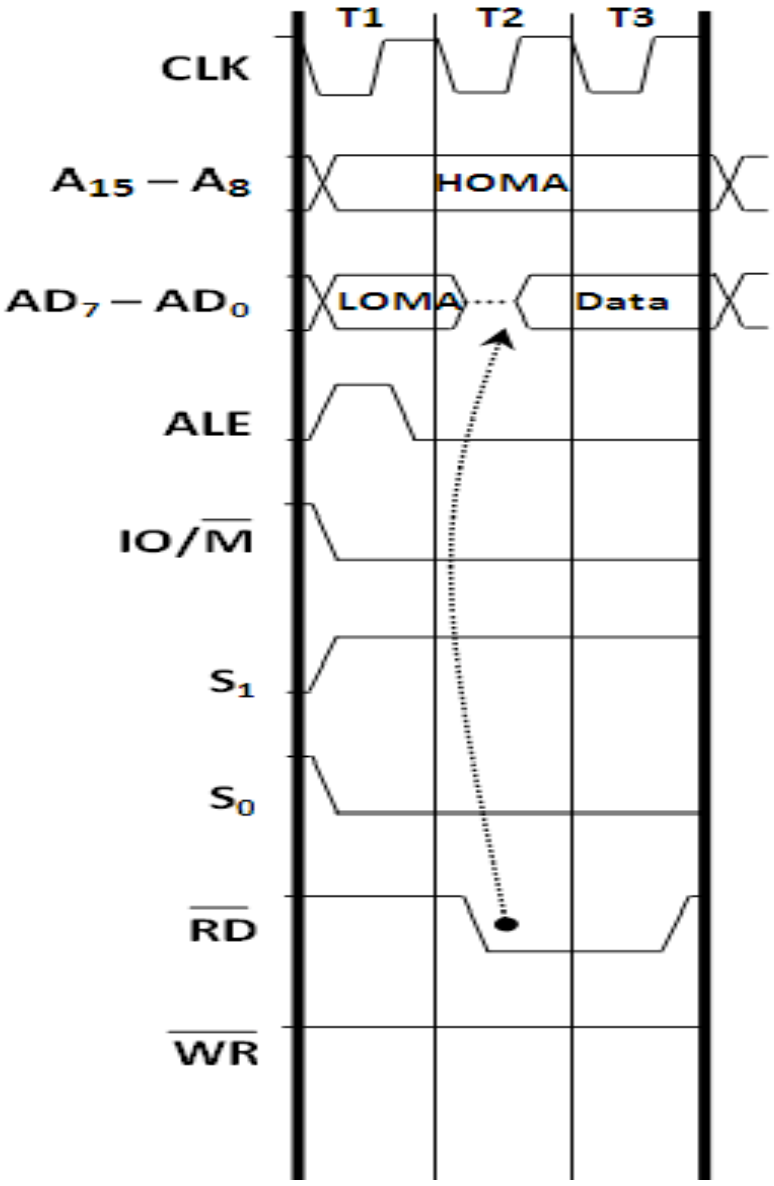
- **A15-A8** contains the higher byte of the address (**PCH**)
- As **ALE** is **high**, **AD7-AD0** contains the lower byte of the address (**PCL**).
- Since it is a Memory Read cycle, **S1** goes **high**.
- Since it is a memory operation, **IO/M** goes **low**.

During T2

- **ALE** goes **low**.
- Address is removed from AD7-AD0.
- As **RD** goes **low, data** appears **on AD7-AD0**.

During T3

- Data remains on AD7-AD0 till RD is low.



BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

Memory Write

- This cycle is used to **send** (write) **one byte into** the **memory**.
- It is a not compulsory Machine Cycle
- It requires **3 T-States**.

During T1

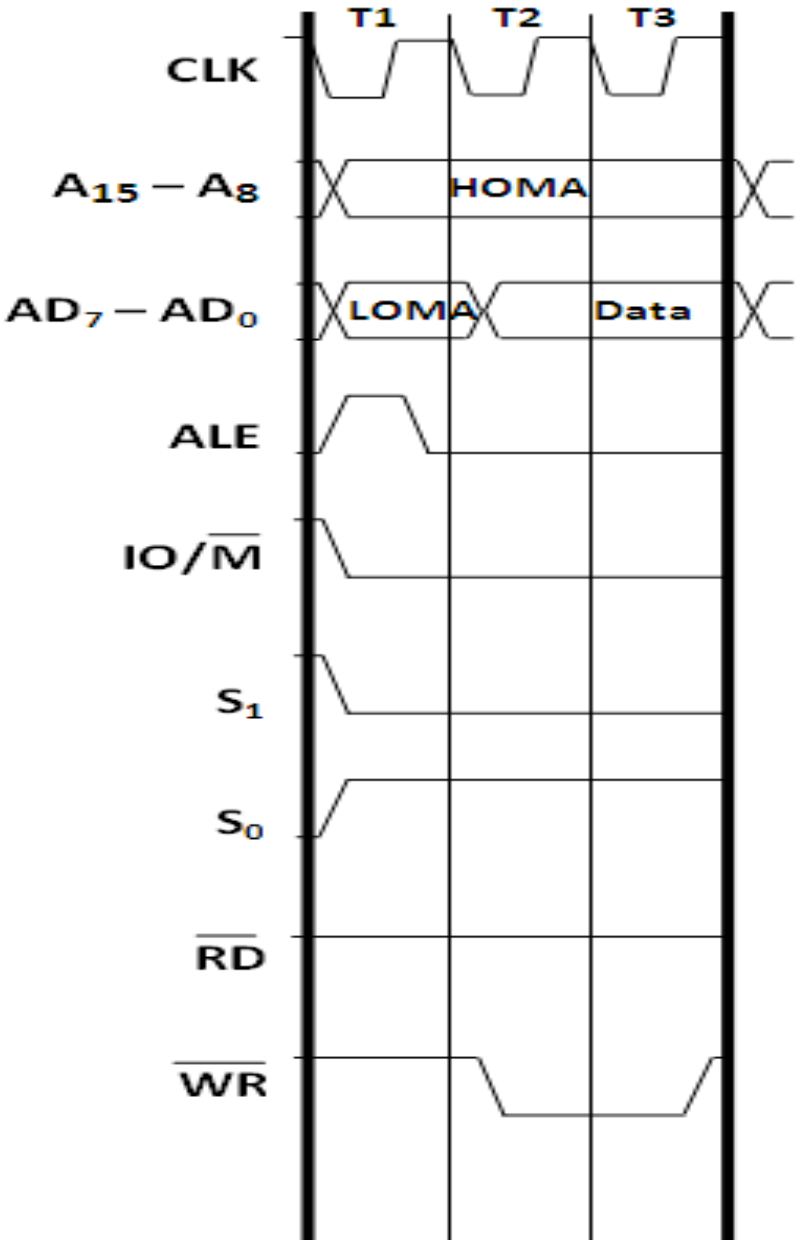
- **A15-A8** contains the higher byte of the address (**PCH**)
- As **ALE** is **high**, **AD7-AD0** contains the lower byte of the address (**PCL**).
- Since it is a Memory Write cycle, **S0** goes **high**.
- Since it is a memory operation, **IO/M** goes **low**.

During T2

- **ALE** goes **low**.
- Address is removed from AD7-AD0.
- **Data** appears **on AD7-AD0** and **WR** goes **low**.

During T3

- Data remains on AD7-AD0 till WR is low.



BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

IO Read

- This cycle is used to **fetch one byte from** an **IO Port**.
- It is a not compulsory Machine Cycle
- It requires **3 T-States**.

During T1

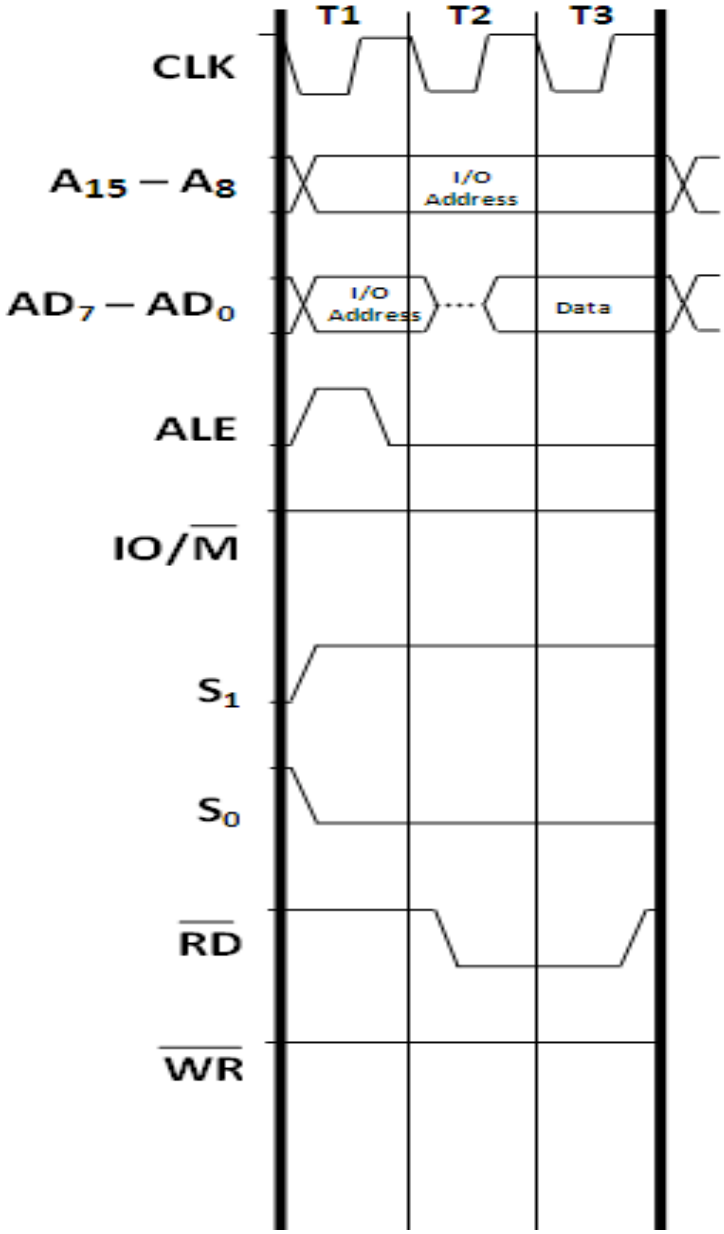
- The **lower 8 bits of the IO Port Address** are **duplicated into** the higher order address bus **A15-A8**.
- As **ALE** is **high AD7-AD0 contains** the **lower byte of** the **address**
- Since it is an IO **Read** cycle **S1** goes **high**.
- Since it is an **IO** operation **IO/M** goes **high**.

During T2

- **ALE** goes **low.**
- Address is removed from AD7-AD0.
- As **RD** goes **low**, **data** appears on **AD7-AD0**.

During T3

- Data remains on AD7-AD0 till RD is low.



BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

IO Write

- This cycle is used to **send** (write) **one byte into** an **IO Port**.
- It is a not compulsory Machine Cycle
- It requires **3 T-States.**

During T1

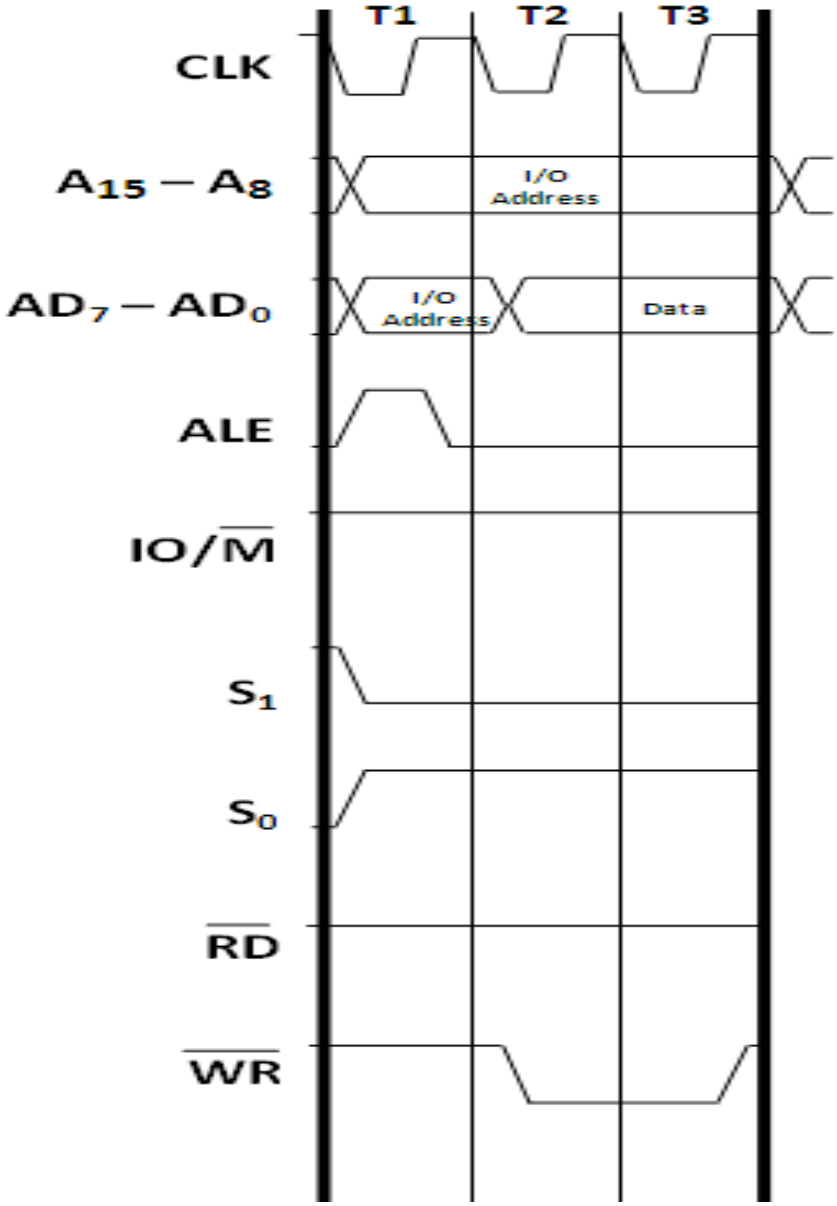
- The **lower 8 bits of the IO Port Address** are **duplicated into** the higher order address bus **A15-A8**.
- As **ALE** is **high AD7-AD0 contains** the **lower byte of** the **address**
- Since it is an IO **Write** cycle, **S0** goes **high**.
- Since it is an **IO** operation, **IO/M** goes **high**.

During T2

- **ALE** goes **low.**
- Address is removed from AD7-AD0.
- **Data** appears **on AD7-AD0** and **WR** goes **low**.

During T3

- Data remains on AD7-AD0 till RD is low.



BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

TIMING DIAGRAMS OF INSTRUCTIONS

Note: Any Operation that is performed within the μP does not require a machine cycle and hence is not shown in timing diagrams.

1) MVI B, 25H

B ← 25 H

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
Memory Read	PC + 1	25	**3**
		Total	**7**

2) LXI B, 2000H

BC ← 2000 H

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
Memory Read	PC + 1	00	**3**
Memory Read	PC + 2	20	**3**
		Total	**10**

3) LDA 2000H

A ← [2000H]

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
Memory Read	PC + 1	00 (Z)	**3**
Memory Read	PC + 2	20 (W)	**3**
Memory Read	2000	[2000]	**3**
		Total	**13**

4) STA 3000H

A → [3000H]

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
Memory Read	PC + 1	00 (Z)	**3**
Memory Read	PC + 2	30 (W)	**3**
Memory Write	3000	A	**3**
		Total	**13**

5) LDAX B

A ← [BC]

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
Memory Read	BC	[BC]	**3**
		Total	**7**

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

6) STAX D

A → [DE]

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
Memory Write	DE	A	**3**
		Total	**7**

7) LHLD 2000H

L ← [2000H], H ← [2001H]

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
Memory Read	PC + 1	00 (Z)	**3**
Memory Read	PC + 2	20 (W)	**3**
Memory Read	2000	[2000]	**3**
Memory Read	2001	[2001]	**3**
		Total	**16**

8) SHLD 5140H

L → [5140H], H → [5141H]

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
Memory Read	PC + 1	40 (Z)	**3**
Memory Read	PC + 2	51 (W)	**3**
Memory Write	5140	L	**3**
Memory Write	5141	H	**3**
		Total	**16**

9) MOV B,C

B ← C

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
		Total	**4**

10) PCHL

PC ← HL

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**6**
		Total	**6**

Cell: 98204 08217
BharatSir@hotmail.com

11) SPHL

SP ← HL

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**6**
		Total	**6**

12) ADD B {all 8 bit arithmetic operations using register addressing mode}

A ← A + B

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
		Total	**4**

13) INR B

B ← B + 1

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
		Total	**4**

14) INX B

BC ← BC + 1

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**6**
		Total	**6**

Instructions Involving "M" – Memory Pointer

15) MVI B, 25H

B ← B + 1

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
Memory Read	PC + 1	25	**3**
Memory Write	HL	25	**3**
		Total	**10**

16) MOV B, M

B ← M

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
Memory Read	HL	M	**3**
		Total	**7**

Cell: 98204 08217
BharatSir@hotmail.com

17) MOV M, B

M ← B

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
Memory Write	HL	B	**3**
		Total	**7**

18) INR M {Very Important}

M ← M + 1

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
Memory Read	HL	M	**3**
Memory Write	HL	M + 1	**3**
		Total	**10**

19) ADD M

A ← A + M

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
Memory Read	HL	M	**3**
		Total	**7**

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

Stack Operations

20) PUSH B

SP ← SP – 1 … internal operation
[SP] ← B … memory write
SP ← SP – 1 … internal operation
[SP] ← C … memory write

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**6**
Memory Write	SP – 1	B	**3**
Memory Write	SP – 2	C	**3**
		Total	**12**

21) POP B

C ← [SP] … memory read
SP ← SP + 1 … internal operation
B ← [SP] … memory read
SP ← SP + 1 … internal operation

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
Memory Read	SP	[SP]	**3**
Memory Read	SP +1	[SP + 1]	**3**
		Total	**10**

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

Branch Operations

22) JMP 2000H

PC ← 2000 H

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
Memory Read	PC + 1	00 (Z)	**3**
Memory Read	PC + 2	20 (W)	**3**
		Total	**10**

23) JC 2000H

If CF = 1 then condition is true hence,

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
Memory Read	PC + 1	00 (Z)	**3**
Memory Read	PC + 2	20 (W)	**3**
		Total	**10**

If CF = 0 then condition is false hence,

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
Memory Read (Idle)	---	---	**3**
		Total	**7**

24) Call 2000H

SP ← SP – 1 … internal operation
[SP] ← PCH … Memory Write
SP ← SP – 1 … internal operation
[SP] ← PCL … Memory Write
PC ← 2000 H … internal operation

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**6**
Memory Read	PC + 1	00 (Z)	**3**
Memory Read	PC + 2	20 (W)	**3**
Memory Write	SP – 1	PCH	**3**
Memory Write	SP – 2	PCL	**3**
		Total	**18**

25) CC 2000H

If CF=1 then condition is true hence,

SP ← SP – 1 … internal operation
[SP] ← PCH … Memory Write
SP ← SP – 1 … internal operation
[SP] ← PCL … Memory Write
PC ← 2000 H … internal operation

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**6**
Memory Read	PC + 1	00 (Z)	**3**
Memory Read	PC + 2	20 (W)	**3**
Memory Write	SP – 1	PCH	**3**
Memory Write	SP – 2	PCL	**3**
		Total	**18**

If CF = 0 then condition is false hence,

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**6**
Memory Read (Idle)	---	---	**3**
		Total	**9**

26) RET

PCL ← [SP] … Memory Read
SP ← SP + 1 … internal operation
PCH ← [SP] … Memory Read
SP ← SP + 1 … internal operation

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
Memory Read	SP	[SP]	**3**
Memory Read	SP + 1	[SP + 1]	**3**
		Total	**12**

27) RC

If CF = 1 then condition is true hence,

PCL ← [SP] … Memory Read
SP ← SP + 1 … internal operation
PCH ← [SP] … Memory Read
SP ← SP + 1 … internal operation

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**6**
Memory Read	SP	[SP]	**3**
Memory Read	SP + 1	[SP + 1]	**3**
		Total	**12**

If CF = 0 then condition is false hence,

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**6**
		Total	**9**

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

28) RSTn

SP ← SP – 1 … internal operation
[SP] ← PCH … Memory Write
SP ← SP – 1 … internal operation
[SP] ← PCL … Memory Write
PC ← (n x 8) … internal operation

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**6**
Memory Write	SP – 1	PCH	**3**
Memory Write	SP – 2	PCL	**3**
		Total	**12**

I/O Operations

29) IN 80H

A ← $[80]_{I/O}$

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
Memory Read	PC + 1	80	**3**
I/O Read	80	[80]	**3**
		Total	**10**

30) OUT 80H

A → $[80]_{I/O}$

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
Memory Read	PC + 1	80	**3**
I/O Write	80	A	**3**
		Total	**10**

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

Additional Instructions

31) DAD D

HL ← HL + DE

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
Bus Idle	---	---	**3**
Bus Idle	---	---	**3**
		Total	**10**

32) HLT

Halt F/F ← 1

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4T +1T**
		Total	**5**

33) XTHL

Z ← [SP] … Memory Read
W ← [SP + 1] … Memory Read
[SP + 1] ← H … Memory Write
[SP] ← L … Memory Write
HL ← WZ … internal operation

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
Memory Read	SP	[SP]	**3**
Memory Read	SP + 1	[SP + 1]	**3**
Memory Write	SP + 1	H	**3**
Memory Write	SP	L	**3**
		Total	**16**

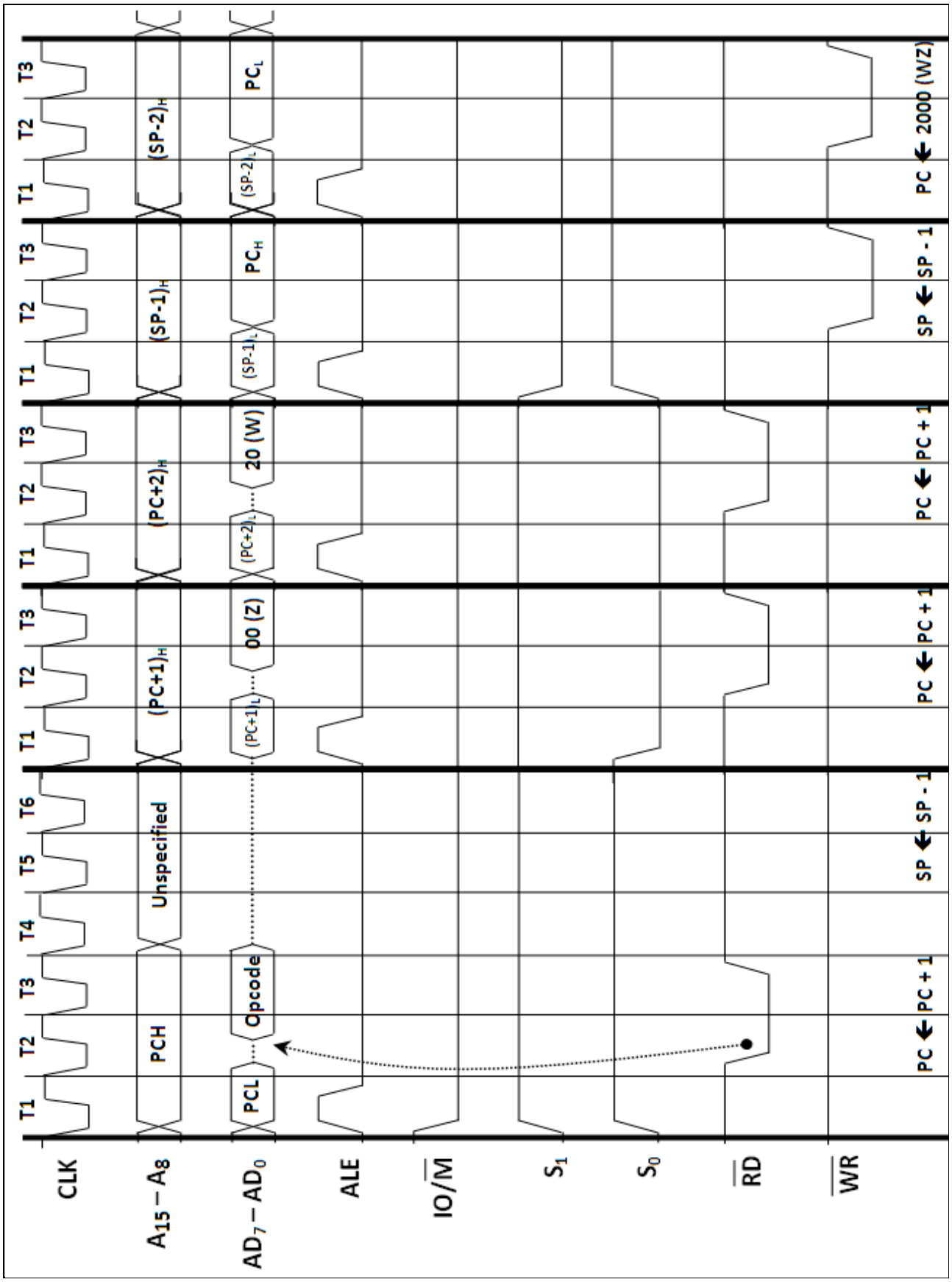
34) XCHG

DE ←→ HL … internal operation

Machine Cycle	Address Bus	Data Bus	T-States
Opcode Fetch	PC	Opcode (Eg: 3E)	**4**
		Total	**4**

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

Call 2000 H



BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

8085 PROGRAMMING

Q 1) **WAP to add the contents of locations 4000H and 4001H. Store the sum at 4002H and the carry at 4003H.**

Soln:

```
        MVI   B, 00H
        LXI   H, 4000H
        MOV   A, M
        INX   H
        ADD   M
        JNC   SKIP
        INC   B
SKIP:   INX   H
        MOV   M, A
        INX   H
        MOV   M, B
        RST1
```

Q 2) **WAP to add two BCD numbers stored in the locations 4000H and 4001H. Store the result at 4001H and 4002H.**

Soln:

```
        MVI   B, 00H
        LXI   H, 4000H
        MOV   A, M
        INX   H
        ADD   M
        DAA
        JNC   SKIP
        INC   B
SKIP:   INX   H
        MOV   M, A
        INX   H
        MOV   M, B
        RST1
```

Q 3) **WAP to add a series of 10 numbers stored from location 4000H. Store the result immediately after the series.**

Soln:

```
        SUB   A
        MOV   B, A
        LXI   H, 4000H
        MVI   C, 0AH

BACK:   ADD   M
        JNC   SKIP
        INC   B
SKIP:   INX   H
        DCR   C
        JNZ   BACK

        MOV   M, A
        INX   H
        MOV   M, B
        RST1
```

Cell: 98204 08217
BharatSir@hotmail.com

Q 4) **WAP to find the largest in a given series of 10 numbers starting from location 4000H. Store the result immediately after the series.**

Soln:

```
      LXI   H, 4000H
      MVI   C, 0AH
      SUB   A
BACK: CMP   M
      JNC   SKIP
      MOV   A, M
SKIP: INX   H
      DCR   C
      JNZ   BACK
      INX   H
      RST1
```

Q 5) **WAP to find the number of +ve, -ve and zeros in a given series of 10 numbers. Store the result immediately after the series.**

Soln:

```
      SUB   A
      MOV   B, A        ; B = No of zeros
      MOV   C, A        ; C = No of +ves
      MOV   D, A        ; D = No of -ves
      LXI   H, 4000H
      MVI   E, 0AH

BACK: CPM   M           ; A - M i.e. 00H - Current number
      JZ    ZERO        ; Current number must be zero
      JC    POSV        ; Current number must be greater than zero
NEGV: INR   D           ; Current number must be less than zero
      JMP   NEXT
POSV: INR   C
      JMP   NEXT
ZERO: INR   B

NEXT: INX   H
      DCR   E
      JNZ   BACK
      RST1
```

Q 6) **SORT ACSENDING a series of 10 numbers starting from location 2100H.**

Soln:

```
      MVI   B, 09H
BCK2: LXI   H, 2100H
      MVI   C, 09H

BCK1: MOV   E, M        ; Current number in E
      INX   H
      MOV   A, M        ; Next number in A
      CMP   E           ; A - E
      JNC   SKIP        ; If next number is greater then don't bother
      MOV   M, E        ; else exchange the two numbers
      DCX   H
      MOV   M, A
      INX   H

SKIP: DCR   C
      JNZ   BCK1
      DCR   B
      JNZ   BCK2
      RST1
```

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

BLOCK TRANSFER PROGRAMS:

Q 7) WAP to perform BLOCK TRANSFER of 10 bytes from location 2000H to location 3000H.

Soln:

```
      MVI   L, 09H
      LXI   B, 2000H
      LXI   D, 3000H

BACK: LDAX  B
      STAX  D
      INX   B
      INX   D
      DCR   L
      JNZ   BACK
      RST1
```

Q 8) WAP to perform OVERLAPPING BLOCK TRANSFER of 10 bytes from location 2000H to location 2004H.

Soln:

```
      MVI   L, 09H
      LXI   B, 2009H
      LXI   D, 200DH

BACK: LDAX  B
      STAX  D
      DCX   B
      DCX   D
      DCR   L
      JNZ   BACK
      RST1
```

Q 9) WAP to perform INVERTED BLOCK TRANSFER of 10 bytes from location 2000H to location 3000H.

Soln:

```
      MVI   L, 09H
      LXI   B, 2000H
      LXI   D, 3009H

BACK: LDAX  B
      STAX  D
      INX   B
      DCX   D
      DCR   L
      JNZ   BACK
      RST1
```

Cell: 98204 08217
BharatSir@hotmail.com

Q 10) WAP to MULTIPLY two 8-bit numbers stored in location 2000H and 2001H. Store the result at 2002H and 2003H.

Soln:

```
        LXI   H, 0000H
        LXI   D, 0000H

        LDA   2000H       ; Take multiplicand in A
        ADI   00H         ; Check for zero
        JZ    EXIT        ; If zero, then simply exit
        MOV   E, A        ; Else take multiplicand into E

        LDA   2001H       ; take multiplier in A
        ADI   00H         ; Check for zero
        JZ    EXIT        ; If zero, then simply exit
        MOV   C, A        ; Else take multiplier in C as the count

BACK:   DAD   D           ; Add multiplicand to itself
        DCR   C           ; C number of times
        JNZ   BACK

EXIT:   SHLD  2002H       ; Store the result as required
        RST1
```

Q 11) WAP to divide two 8-bit numbers stored at 2000H and 2001H. Store the result at 2002H and 2003H.

Soln:

```
        LXI   H, 2000H
        MOV   B, M        ; Take dividend in B
        INX   H
        MOV   A, M        ; Take divisor in A
        ADI   00H         ; Check for zero
        JZ    EXIT        ; If zero, its an INVALID operand. Simply exit.

        MOV   A, B        ; A gets the dividend
        MOV   B, M        ; B gets the divisor
        MVI   C, 00H      ; C will be the quotient

BACK:   CMP   B           ; A - B
        JC    DONE        ; no further steps as A < B
        SUB   B           ; A ← A - B
        INR   C
        JMP   BACK

DONE:   STA   2002H       ; Store remainder
        MOV   A, C
        STA   2003H       ; Store quotient
EXIT:   RST1
```

Cell: 98204 08217
BharatSir@hotmail.com

Q 12) WAP to generate a delay of 1 msec using 8085 working at 3 MHz.

Soln:

```
DLAY: MVI   B, XXH      ; 7 T-states … … … Count is calculated later
BACK: DCR   B           ; 4 T-states … … … Decrement Count
      JNZ   BACK        ; 10T (true) / 7T (false)
      RET               ; 10T-states
```

T_D = MT + [(Count)$_d$ x NT] – 3T

Here MT = Time outside the loop = 17T
NT = Time inside the loop = 14T

T_D = 17T + [(Count)$_d$ x 14T] – 3T

Required T_D = 1 msec = 10^{-3} sec
Given 1T = 0.333 µsec = 0.333 x 10^{-6} sec

Substituting the above values we get:

10^{-3} = 17x(0.333x10^{-6}) + [(Count)$_d$ x 14x(0.333x10^{-6})] – 3x(0.333x10^{-6})

Dividing by (0.333 x 10^{-6}) we get:

3003 = 17 + [(Count)$_d$ x 14] + 3

2983 = [(Count)$_d$ x 14]

213 = (Count)$_d$

Count = D5H

Similarly any other required delay can be achieved.
In this method, the max-delay achieved is 1.18 msec with count = FFH.
In the above calculations, the value of "1T" will change if operating frequency is anything other than 3 MHz.
If frequency is not given, then you can assume it to be 3 or 5 MHz.

Q 13) WAP to generate a delay of 0.5 msec using 8085 working at 3 MHz.

Soln: "Home-work"

Answer: Count = 6AH

Cell: 98204 08217
BharatSir@hotmail.com

Q 14) WAP to generate a SQUARE-WAVE of 1 KHz using SOD pin of 8085.

Soln:

```
BACK: MVI   A, 40H      ; SIM Command = 0100  0000
      SIM
      CALL  DLAY
      MVI   A, C0H      ; SIM Command = 1100  0000
      SIM
      CALL  DLAY
      JMP   BACK
```

For a square wave of 1 KHz, the time period is 1 msec.
Hence the required delay is of 0.5 msec.
Assume 8085 is working at 3 MHZ

```
DLAY: MVI   B, XXH      ; 7 T-states … … … Count is calculated later
BACK: DCR   B           ; 4 T-states … … … Decrement Count
      JNZ   BACK        ; 10T (true) / 7T (false)
      RET               ; 10T-states
```

T_D = MT + [(Count)$_d$ x NT] – 3T
Here MT = Time outside the loop = 17T
NT = Time inside the loop = 14T
T_D = 17T + [(Count)$_d$ x 14T] – 3T
Required T_D = 0.5 msec = 0.5 x 10^{-3} sec
1T = 0.333 µsec = 0.333 x 10^{-6} sec
Substituting the above values we get:
$0.5x10^{-3} = 17x(0.333x10^{-6}) + [(Count)_d \times 14x(0.333x10^{-6})] - 3x(0.333x10^{-6})$
Count = 6AH

Q 15) WAP to transfer the value 35H serially with one start bit "0" and one stop bit "1".

Soln: Serial communication happens bit by bit starting from the LSB.
As per the question, we need to send the start bit (0), then the data and finally the stop bit (1).
Hence a total of 10 bits will move out as follows:

```
  0              1 0 1 0 1 1 0 0           1
Start      8-data bits in reverse order    Stop
```

```
      MVI   A, 40H      ; start bit (0)
      SIM
      MVI   A, C0H      ; send a "1"
      SIM
      MVI   A, 40H      ; send a "0"
      SIM
      MVI   A, C0H      ; send a "1"
      SIM
      MVI   A, 40H      ; send a "0"
      SIM
      MVI   A, C0H
      SIM               ; send a "1"
      SIM               ; send a "1" again
      MVI   A, 40H      ; send a "0"
      SIM
      SIM               ; send a "0" again
      MVI   A, C0H      ; send a "1" as the stop bit
      SIM
      RST1
```

Cell: 98204 08217
BharatSir@hotmail.com

Q 16) WAP to transfer the value 35H serially with one start bit (0) and one stop bit (1) at a Baud rate of 2400. Assume 8085 is working at 3 MHz.

Soln: Baud rate is the rate at which data is send.
BR = 2400 means 2400 bits have to be sent in 1 second.
Hence the delay between sending two bits is of (1/2400) seconds.
$T_D = 0.41667 \times 10^{-3}$ sec

```
DLAY: MVI   B, XXH      ; 7 T-states … … … Count is calculated later
BACK: DCR   B           ; 4 T-states … … … Decrement Count
      JNZ   BACK        ; 10T (true) / 7T (false)
      RET               ; 10T-states
```

$T_D = MT + [(Count)_d \times NT] - 3T$
Here MT = Time outside the loop = 17T
NT = Time inside the loop = 14T
$T_D = 17T + [(Count)_d \times 14T] - 3T$

Required $T_D = 0.41667$ msec $= 0.41667 \times 10^{-3}$ sec
$1T = 0.333$ µsec $= 0.333 \times 10^{-6}$ sec
Substituting the above values we get:
$0.41667 \times 10^{-3} = 17 \times (0.333 \times 10^{-6}) + [(Count)_d \times 14 \times (0.333 \times 10^{-6})] - 3 \times (0.333 \times 10^{-6})$
Count = 58H

A total of 10 bits will move out as follows:

```
  0              1 0 1 0 1 1 0 0           1
Start       8-data bits in reverse order   Stop
```

```
   MVI   A, 40H      ; start bit (0)
   SIM
   CALL  DLAY
   MVI   A, C0H      ; send a "1"
   SIM
   CALL  DLAY
   MVI   A, 40H      ; send a "0"
   SIM
   CALL  DLAY
   MVI   A, C0H      ; send a "1"
   SIM
   CALL  DLAY
   MVI   A, 40H      ; send a "0"
   SIM
   CALL  DLAY
   MVI   A, C0H
   SIM               ; send a "1"
   CALL  DLAY
   SIM               ; send a "1" again
   CALL  DLAY
   MVI   A, 40H      ; send a "0"
   SIM
   CALL  DLAY
   SIM               ; send a "0" again
   CALL  DLAY
   MVI   A, C0H      ; send a "1" as the stop bit
   SIM
   CALL  DLAY
   RST1
```

Cell: 98204 08217
BharatSir@hotmail.com

Q 17) WAP to transfer a RANDOM NUMBER stored at location 2000H serially with a start bit (0) and a stop bit (1).

Soln:

```
        LDA   2000H       ; read number in A
        MOV   B, A        ; store number in B
        MVI   C, 08H      ; count

        MVI   A, 40H      ; send "zero" as the start bit
        SIM

BACK:   MOV   A, B        ; get the number
        RRC               ; get its LSB in Carry flag
        MOV   B, A        ; store rotated number in B for next iteration
        JC    ONE         ; if carry flag is "1" then go to send a "one"
        MVI   A, 40H      ; else send a "zero"
        SIM
        JMP NEXT
ONE:    MVI   A, COH      ; send a "one"
        SIM
NEXT:   DCR C
        JNZ BACK          ; repeat for all 8-bits

        MVI A, C0H        ; send a "one" as the stop bit
        SIM
        RST1
```

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

STACKS AND SUBROUTINES

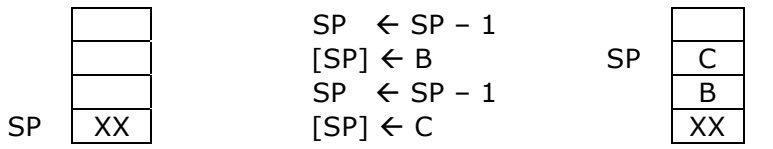
STACKS

A **stack** is a **part of the Memory** that **operates in LIFO** (Last In First Out) manner.
In 8085 the **Stack** is **located anywhere within the 64 KB memory**.
The **top of the Stack** is **pointed by** the **SP** (Stack Pointer) register.
P.S.: The SP contains the **Address** of the top of the Stack and **not** the **top of the Stack**.

Instructions affecting the stack are:
- **LXI SP**, 16 bit value Eg LXI SP 4000.
- **SPHL**
- **INX SP**
- **DCX SP**
- **PUSH** Rp Eg PUSH B
- **POP** Rp Eg POP B
- **CALL** 16 bit address Eg CALL 2000
- **RSTn** Eg RST1
- **RET**
- **XTHL**

Usefulness of Stacks (Uses of Stacks)
Stacks are used in the following ways:

1) For Temporary Storage of data
Stack can be used for **storing data by** the **programmer** as the number of General Purpose registers is very less. The programmer, at any point of time during the program, can store data in the stack and then retrieve it later.
Data can be Pushed into the Stack using Push instruction as below:
Eg : **PUSH B** ; Contents of BC Pair are pushed onto the top of the stack.



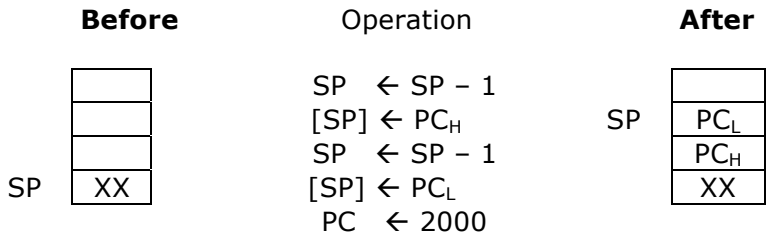
Similarly data can also be removed from the stack using the POP operation as:
Eg : **POP B** ; Contents of the top of the stack are popped into BC Pair.

2) By the µP for storing the return address for CALL instruction
The **µP uses** the Stack **for storing** the **return address** during **CALL** instruction.
During a CALL instruction, **µP** first pushes the return address (i.e. the current contents of PC) into the stack and then loads the branch address into PC, to branch to the subroutine.
Inside the subroutine, when the **µP** gets a RET instruction, it pops back the return address from the stack, and thus successfully returns to the main program.

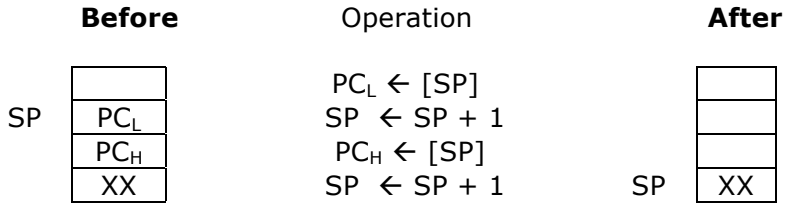
This is illustrated as follows:

Eg : **CALL 2000**

Cell: 98204 08217
BharatSir@hotmail.com



Eg : **RET**



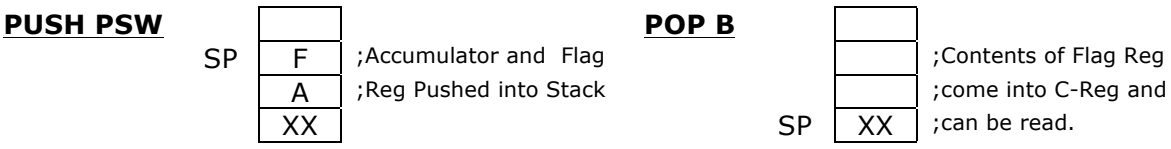
3) To Read/Write the contents of the Flag register

The programmer can Read / Write the contents of the Flag register using the Program Status Word **(PSW)**. The PSW consists of the Accumulator as the higher byte and the Flag register as the lower byte. The PSW is available only for PUSH and POP instructions.

To Read the contents of Flag register:

- The programmer pushes PSW into the stack and then pops it into any register pair (BC).
- The contents of the flag register are thus available in the C register (lower register).

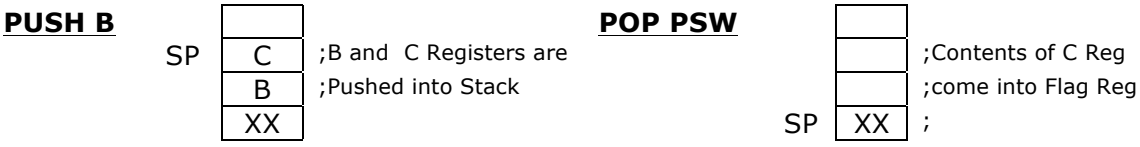
Eg:



To Write into the Flag register:

- The programmer loads the appropriate byte into the lower register of a register pair (eg: C reg of BC pair).
- Then the register pair is pushed into the stack.
- These contents are then popped into the PSW, and thus the byte that was originally loaded into C register is now **written** into the Flag register.

Eg:



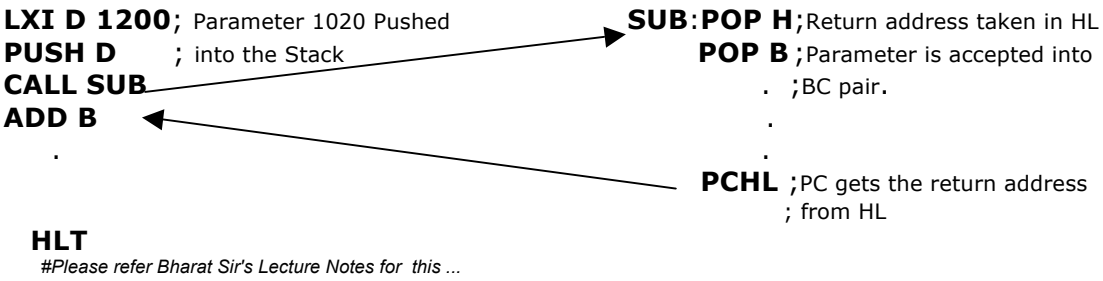
This is the **ONLY method** by which the programmer can **write into** the **Flag register**.

4) **To Pass parameters to a Sub-Routine**

The programmer can use the Stack to pass parameters to Sub-Routines.
Before calling the Sub-Routine the **parameter** is **Pushed** into the Stack and then **inside** the **Sub-Routine** the **parameter** is **Popped from** the **Stack**.

Eg:

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com



#Please refer Bharat Sir's Lecture Notes for this ...

Cell: 98204 08217
BharatSir@hotmail.com

SUBROUTINES

- SubRoutines are parts of a program, which can be re-executed by the programmer.
- SubRoutines are Called (invoked) using the CALL instruction; control returns to the main program using the RET instruction.
- SubRoutines generally perform tasks, which are used regularly by the program such as numerical calculations, Interrupt Service Routines (ISR) etc.
- SubRoutines are useful in the following ways:
  1. Causes **Code Re-Usability** hence reduces the **size** of the Program and also **saves time**.
  2. Makes the program easy to **Maintain** as it becomes **Modular.**

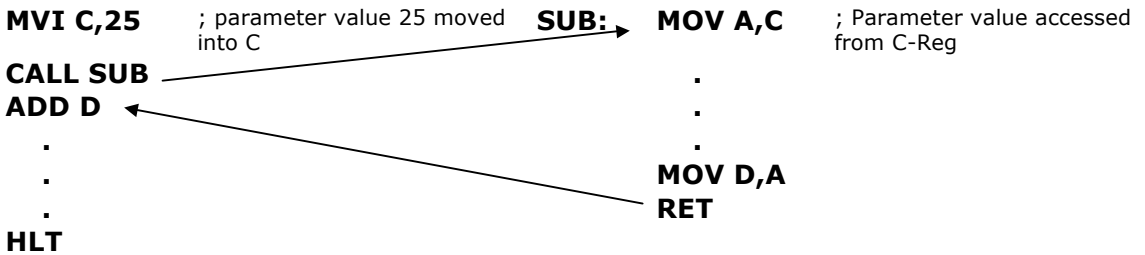
Passing Parameters to Sub-Routines

Passing parameters to SubRoutines is the procedure of passing some values from the main program to the SubRoutines. Passing parameters is very important as it **improves the flexibility of the SubRoutine**.
In 8085 there are 4 methods of passing parameters to SubRoutines.

1) Through Registers

- The parameter value to be passed is moved /loaded into the register before calling the SubRoutine.
- The SubRoutine accesses the value from the same register.

Eg:

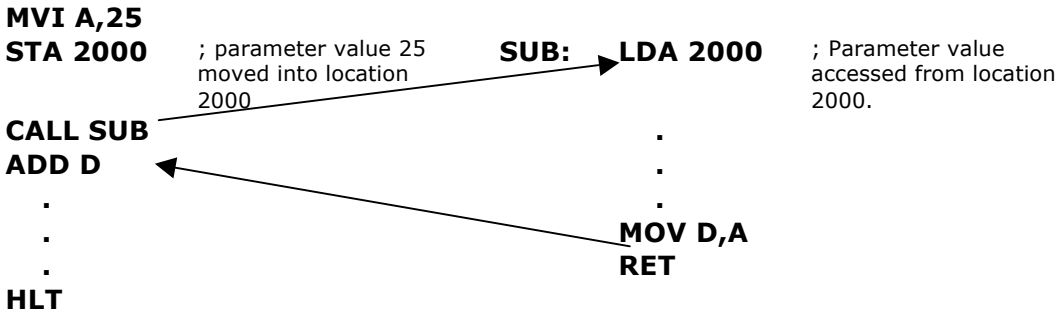


This is the **simplest method** of passing parameters.
The only main **drawback** is that it **occupies** the **general-purpose registers** available to the programmer whish are already very few. **Also** the **number** of parameters passed is **restricted by** the number of **registers available**.

2) Through One or More Memory Locations

- The parameter value to be passed is stored into the Memory Location before calling the SubRoutine.
- The SubRoutine accesses the value from the same memory location.

Eg:



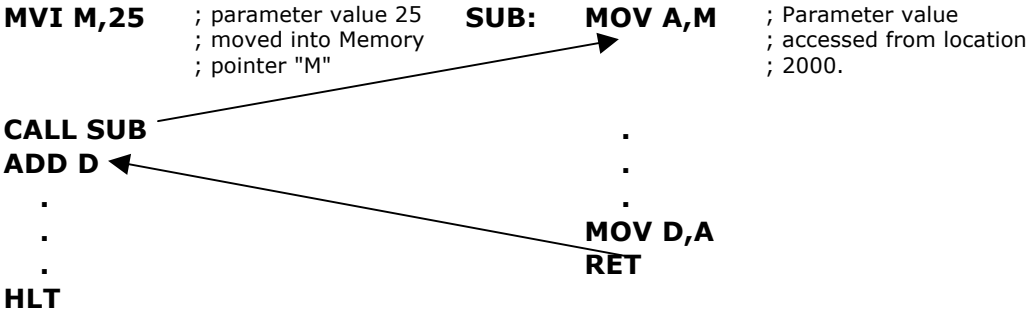
If **more parameters** are to be passed, then **consecutive memory locations** can be used (which are **plenty**), thus **eliminating** the **drawbacks** of the previous method.
The **only drawback** here is that the programmer needs to **remember** the **memory locations** associated with each subroutine. When the number of subroutines is large, this can be troublesome.

Cell: 98204 08217
BharatSir@hotmail.com

3) Through Memory Pointer "M"

- The parameter value to be passed is moved into the Memory pointer **"M"** before calling the SubRoutine. ☺ In case of doubts, contact Bharat Sir: - 98204 08217.
- The HL pair at this point of time may contain any random value.
- The SubRoutine accesses the value from M.
- Care has to be taken that after the value is put into M the value of **HL pair** should not change until the SubRoutine accesses the value of M.

Eg

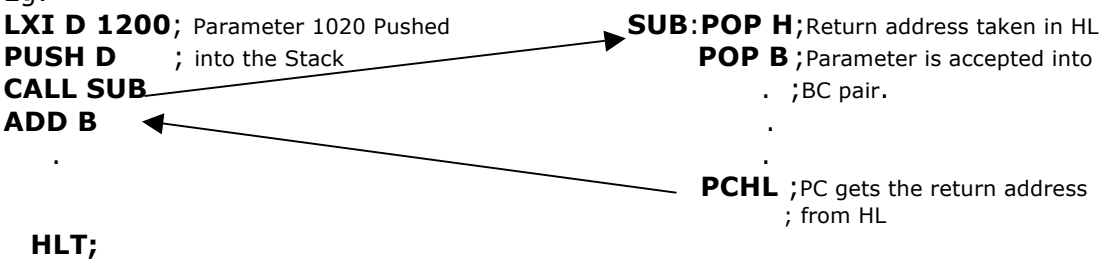


Thus the memory is used but without remembering the memory location.
i.e. In the above example, we still **do not know** the value of **HL** pair, and thus the **address** of the memory location used (thus avoiding the previous drawback).
But, when **more** than one values have to be passed, this method becomes **troublesome**.

4) Through Stack ※ *#Please refer Bharat Sir's Lecture Notes for this ...*

- The **parameter** value **to be passed** is **Pushed** into the Stack before calling the SubRoutine.
- The SubRoutine **Pops** the **passed parameter** from the Stack.
- When the **µP** calls the SubRoutine it **pushes** the **return address** into the stack.
- This address appears above the passed parameter in the stack.
- Due to the **LIFO** property of the stack we cannot access the passed parameter unless we pop the return address.
- Hence inside the SubRoutine **firstly**, the **return address** is **popped into** the **HL** pair.
- Then the **passed parameter** is **popped** into the BC pair.
- Now in this case the **RET** instruction **cannot be used** to come back to the main program as the top of the stack no longer contains the return address (as we had taken it out in the HL pair).
- Thus the return address is obtained from the HL pair using the **PCHL** instruction which **causes** the control to **return to the main program**.

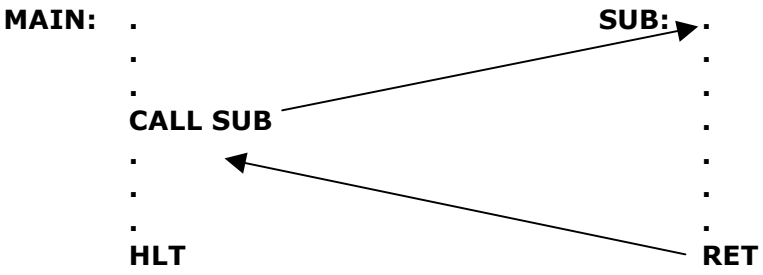
Eg:



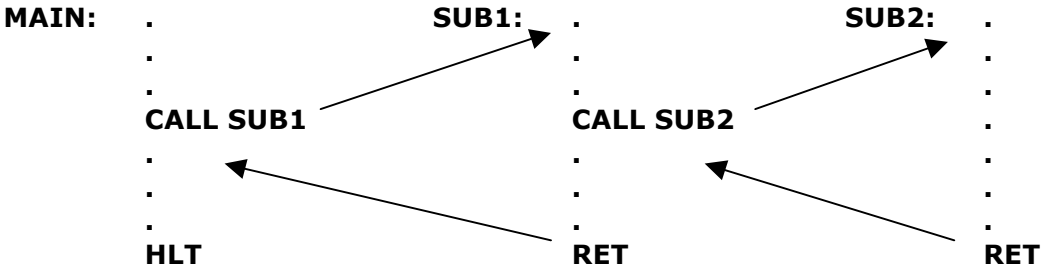
BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

TYPES OF SUB-ROUTINES

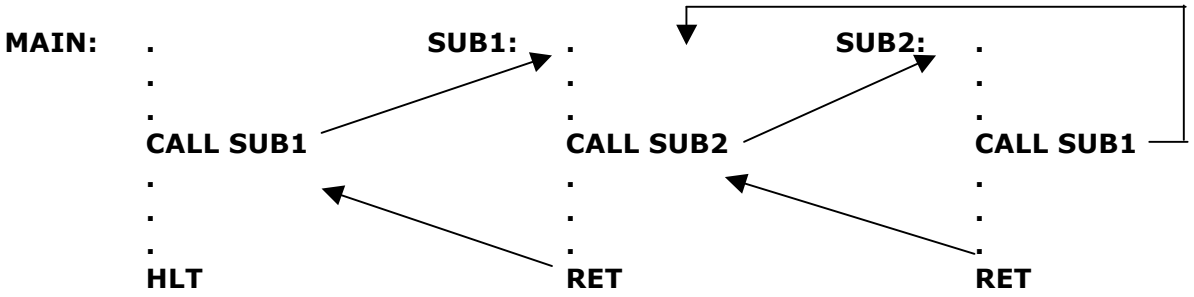
1. **Simple SubRoutines**
   When a SubRoutine does not call another SubRoutine it is called a Simple SubRoutine.



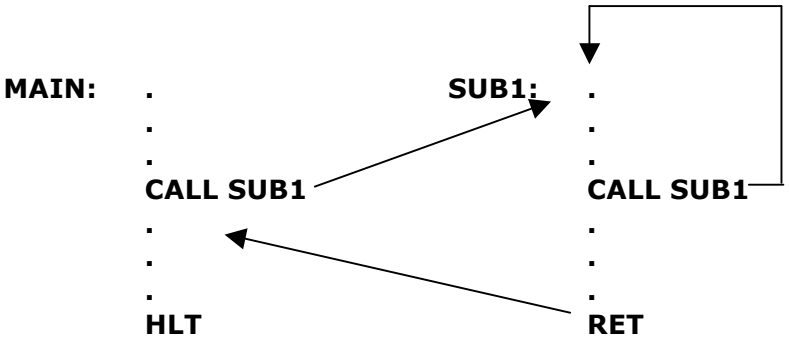
2. **Nested SubRoutines**
   When a SubRoutine calls another SubRoutine it is called a Nested SubRoutine.



3. **Re-entrant SubRoutines**
   When a SubRoutine is re-entered by another SubRoutine it is called a Re-entrant SubRoutine.



4. **Recursive SubRoutines**
   When a SubRoutine calls itself, it is called a Recursive SubRoutine.



BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

DELAYS AND DELAY ROUTINES

Delays in a computer system can be of two types:
1) Hardware delays
2) Software delays

	HARDWARE DELAYS	SOFTWARE DELAYS
1	Caused by external hardware (Timer IC like 8254)	Caused by a software delay routine (program)
2	μP is not involved in causing the delay and hence is free for other applications.	μP is busy as it is executing the delay routine so cannot be used otherwise.
3	Multiple delay routines are possible	Multiple delay routines are not possible
4	Flexibility is low as h/w is involved.	Flexibility is high as delay caused by s/w.
5	External h/w required so ckt. becomes more complex and expensive	External h/w not required so ckt. is simple and less expensive

SOFTWARE DELAYS

Software delays are produced in programs by one of the various software **"Delay Routines"**.

As the amount of delay required varies from application to application different types of delay routines are present, as follows:

1) Using NOP Instruction
2) Using One 8-bit register
3) Using One 16-bit register
   (Nested delay routines):
4) Using Two 8-bit registers
5) Using One 8-bit register and One 16-bit register
6) Using Two 16-bit registers

1) Using NOP Instruction

Eg: NOP;

1-byte instruction
Opcode fetch --- 4 T-states.
T_D = 4T.
T = 1/(Clk freq)
∴ assuming 8085 working at 3 MHz, T = 1/(3 MHz) = 0.333 μ sec.
∴ T_D = 4 × 0.333 = 1.332 μ sec.

∴ T_D = 1.332 μ sec.
This is the maximum delay that can be achieved by writing a NOP instruction.

2) Using One 8-bit register

Delay:	**MVI B**, 8-bit count	7T
Loop:	**DCR B**	4T
	JNZ Loop	10/7T
	RET	10T

NT MT

T_D = MT + [(Count)$_d$ × NT] – 3T

Cell: 98204 08217
BharatSir@hotmail.com

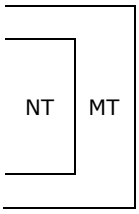
NT = No of T-states inside the loop {here NT = 10T + 4T = **14T**}
MT = No of T-states outside the loop {here MT = 7T + 10T = **17T**}
$Count_{max}$ = 255 {8-bit count in decimal}

$\therefore T_{D\,max}$ = 17T + [255 × 14T] – 3T
$\therefore T_{D\,max}$ = 3584T.
assuming 8085 working at 3 MHz i.e. T = 333 n sec.

$\therefore$ **$T_{D\,max}$ = 1.18 m sec.**

3) Using One 16-bit register

Delay:	**LXI B**, 16-bit count	10T
Loop:	**DCX B**	6T
	MOV A, C	4T
	ORA C	4T
	JNZ Loop	10/7T
	RET	10T



T_D = MT + [$(Count)_d$ × NT] – 3T

here **NT** = 6T + 4T + 4T + 10T = **24T**
MT = 10T + 10T = **20T**

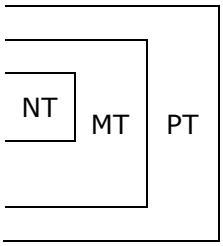
$Count_{max}$ = 65535 {16-bit count in decimal}

$\therefore T_{D\,max}$ = 20T + [65535 × 24T] – 3T
$\therefore T_{D\,max}$ = 1572057T.
assuming 8085 working at 3 MHz i.e. T = 333 n sec.

$\therefore$ **$T_{D\,max}$ = 0.525 sec.**

4) Using Two 8-bit registers

Delay:	**MVI C**, Count2	7T
Loop2:	**MVI B**, Count1	7T
Loop1:	**DCR B**	4T
	JNZ Loop1	10/7T
	DCR C	4T
	JNZ Loop2	10/7T
	RET	10T



T_D = PT + [$(Count2)_d$ × T_{loop1}] – 3T
T_{loop1} = MT + [$(Count1)_d$ × NT] – 3T

NT = No of T-states inside loop1 {here NT = 4T + 10T = **14T**}.
MT = No of T-states outside loop1 but inside loop2 {MT = 7T + 4T + 10T= **17T**}
PT = No of T-states outside loop2 {here PT = 10T + 7T = **17T**}.
$Count1_{max}$ = 255 {8-bit count in decimal}
$Count2_{max}$ = 255 {8-bit count in decimal}

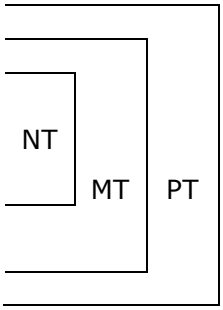
$\therefore T_{D\,max}$ = 914954T.
assuming 8085 working at 3 MHz i.e. T = 333 n sec.

$\therefore$ **$T_{D\,max}$ = 0.304 sec.**

Cell: 98204 08217
BharatSir@hotmail.com

5) Using One 8-bit register and One 16-bit register

Delay:	**MVI D**, Count2	7T
Loop2:	**LXI B**, Count1	10T
Loop1:	**DCX B**	6T
	MOV A, C	4T
	ORA B	4T
	JNZ Loop1	10/7T
	DCR D	4T
	JNZ Loop2	10/7T
	RET	10T



$\mathbf{T_D = PT + [(Count2)_d \times T_{loop1}] - 3T}$
$\mathbf{T_{loop1} = MT + [(Count1)_d \times NT] - 3T}$

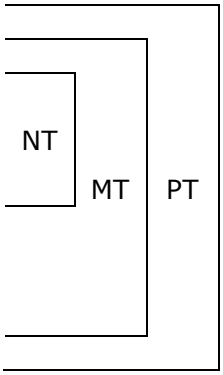
Here **NT** = 6T + 4T + 4T + 10T = **24T**.
MT = MT = 10T + 4T + 10T = **24T**.
PT = 7T + 10T = **17T**.
$Count1_{max}$ = 65535 {16-bit count in decimal}
$Count2_{max}$ = 255 {8-bit count in decimal}

assuming 8085 working at 3 MHz i.e. T = 333 n sec.

∴ $\mathbf{T_{D\ max}}$ **= 133.69 sec.**

6) Using Two 16-bit registers

Delay:	**LXI B**, Count2	10T
Loop2:	**LXI D**, Count1	10T
Loop1:	**DCX D**	6T
	MOV A, E	4T
	ORA D	4T
	JNZ Loop1	10/7T
	DCX B	6T
	MOV A, C	4T
	ORA B	4T
	JNZ Loop2	10/7T
	RET	10T



$\mathbf{T_D = PT + [(Count2)_d \times T_{loop1}] - 3T}$
$\mathbf{T_{loop1} = MT + [(Count1)_d \times NT] - 3T}$

Here **NT** = 6T + 4T + 4T + 10T = **24T**.
MT = MT = 10T + 6T + 4T + 4T + 10T = **34T**.
PT = 10T + 10T = **20T**.
$Count1_{max}$ = 65535 {16-bit count in decimal}
$Count2_{max}$ = 65535 {16-bit count in decimal}

assuming 8085 working at 3 MHz i.e. T = 333 n sec.

∴ $\mathbf{T_{D\ max}}$ **= 9 hours, 32 min, 24 sec.**

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

Summary:

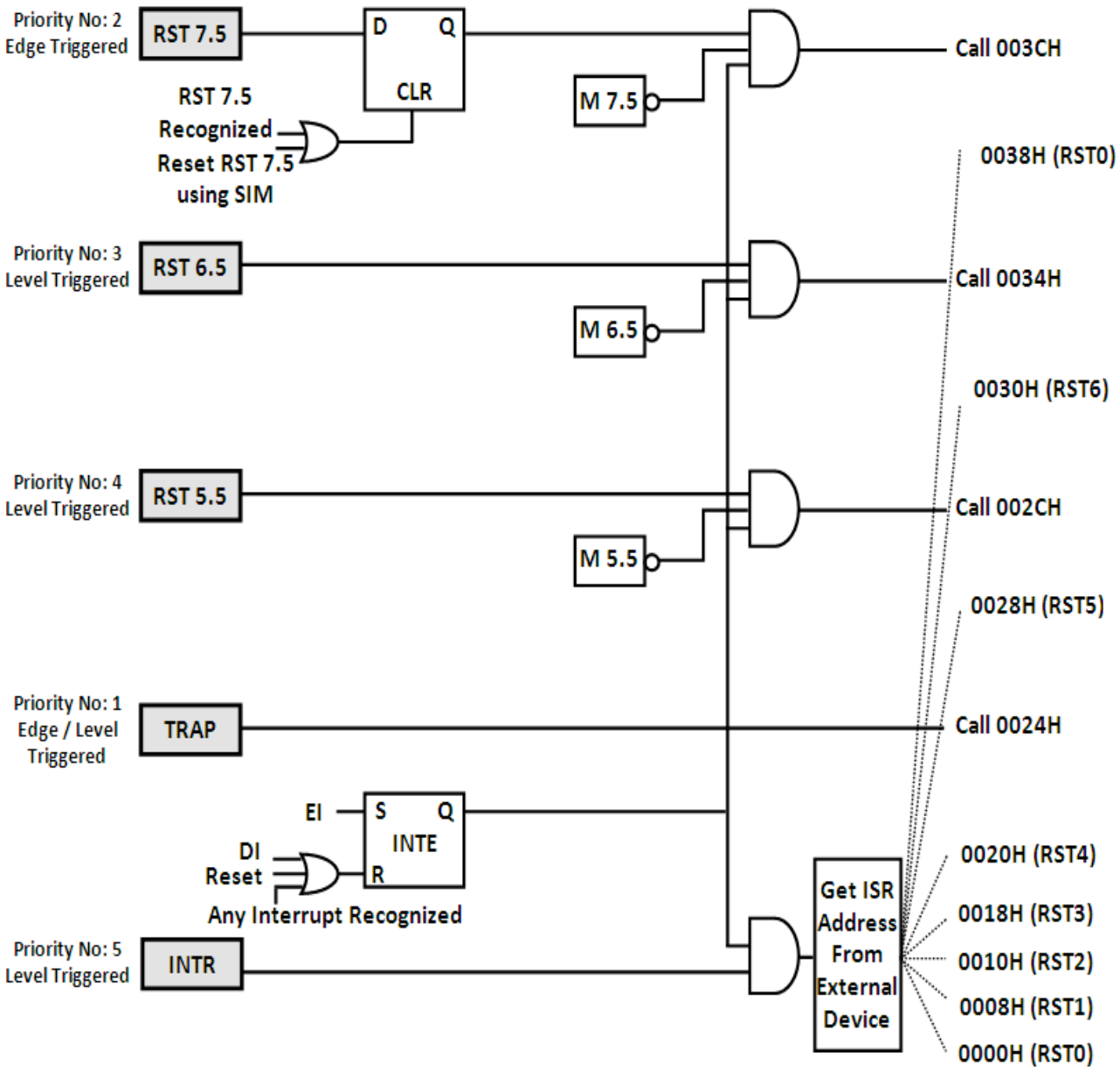
Delay Method	Max duration
NOP	1.332 μ sec
One 8-bit register	1.18 m sec
One 16-bit register	.525 sec
Two 8-bit registers	.304 sec
One 8-bit / one 16-bit reg	133.69 sec
Two 16-bit register	9 hrs, 32 min, 24 sec

Please Note: *All these calculations are w.r.t. 8085 operating at 3 MHz.*
In case in the exam, the frequency is different then calculate 1T = 1/(Clk. freq.), and then calculate the appropriate delay. ☺ In case of doubts, contact Bharat Sir: - 98204 08217.

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

INTERRUPTS OF 8085

- An interrupt is a **special condition** that arises during the working of a µP.
- In response, the **µP services** the interrupt **by executing** a subroutine called as the **Interrupt Service Routine**.
- The **µP checks** for interrupts **during every instruction**.
- When an **interrupt occurs**, the **µP first finishes the current instruction**.
- It then **Pushes** the address of the next instruction **(contents of PC) on the STACK**.
- It **resets** the **INTE flip-flop** so that **no more interrupts are recognized**.
- Thereafter the program control transfers to the **address of** the Interrupt Service Routine (**ISR**) and the **µP** thus **executes the ISR**.



Cell: 98204 08217
BharatSir@hotmail.com

Interrupts are of two types:
- Software Interrupts
- Hardware Interrupts

Software Interrupts:
Interrupts that are **initiated through program** (software) are called as software interrupts.
8085 supports 8 software interrupts:
RSTn where n = 0,1,2, … 7 i.e. RST0, RST1 … upto RST7.
- This instruction causes a **service routine** to be Called from the **address (n*8)**.
- Hence, if RST1 occurs then the program control moves to location 0008 (1*8 = 0008).

The respective addresses for software interrupts are given below.

S/W Interrupt	ISR Address
RST0	00**0**0H
RST1	00**08**H
RST2	00**1**0H
RST3	00**18**H
RST4	00**2**0H
RST5	00**28**H
RST6	00**3**0H
RST7	00**38**H

Hardware Interrupts:
Interrupts that are initiated through a hardware pin are called as hardware interrupts.
8085 supports the following hardware interrupts:
- TRAP
- RST 7.5
- RST 6.5
- RST 5.5
- INTR

Vectored Interrupts:
- Interrupts that **have** a **FIXED Address** for their ISR (Interrupt Service Routine) are called as Vectored Interrupts.
- **Eg: TRAP** is a vectored interrupt. Its vector address is 0024H.

Non-Vectored Interrupts:
- Interrupts that **have** a **Variable Address** for their ISR are called as Non-Vectored Interrupts.
- **Eg: INTR** is a Non-Vectored Interrupt.

Methods of preventing an interrupt from occurring.
- **MASK Individual Bits** through **SIM** Instruction
- **Disable all** Interrupts through **DI** Instruction

MASKING:
- We can prevent an interrupt from occurring by MASKING its individual bit through **SIM Instruction**.
- If an interrupt is masked it will not be serviced.
- One of the **main advantages** of masking as opposed to disabling interrupts is that by masking we can **selectively disable a particular interrupt** while keeping other interrupts active**,** whereas through DI instruction all interrupts are disabled.
- ONLY **RST 7.5, RST 6.5 and RST 5.5** can be masked by this method.

Cell: 98204 08217
BharatSir@hotmail.com

DISABLING INTERRUPTS:
- Interrupts can be disabled through the **DI** Instruction.
- This instruction resets the INTE Flip Flop and hence none of the interrupts can occur (**Except TRAP**).
  I.e. INTE F/F ← 0.
- Once disabled, these interrupts can be **re-enabled** through **EI** instruction, which sets the INTE Flip Flop.
  I.e. INTE F/F ← 1.

Hardware Interrupts (In detail)

TRAP:
- TRAP has the **highest priority.**
- It is **Edge as well as Level triggered** hence the signal must go High and also Remain high for some time for it to be recognized. This prevents any noise signal from being accepted.
- It is a Non-Maskable Interrupt i.e. it can **neither be masked nor be disabled.**
- It is a vectored interrupt and has a **vector address of 0024H.**

RST 7.5:
- RST 7.5 has the **priority lower than TRAP.**
- It is **Edge triggered.**
- It is a **Maskable Interrupt** i.e. it can be masked through the **SIM Instruction**.
- It can also be disabled though the **DI Instruction**.
- It is a vectored interrupt and has a **vector address of 003CH.**
- RST 7.5 can also be **reset** through the **R 7.5** bit in the **SIM** Instruction irrespective of whether it is Masked or not.

RST 6.5:
- RST 6.5 has the **priority lower than RST 7.5.**
- It is **Level triggered.**
- It is a **Maskable Interrupt** i.e. it can be masked through the **SIM Instruction**.
- It can also be disabled though the **DI Instruction**.
- It is a vectored interrupt and has a **vector address of 0034H.**

RST 5.5:
- RST 5.5 has the **priority lower than RST 6.5.**
- It is **Level triggered.**
- It is a **Maskable Interrupt** i.e. it can be masked through the **SIM Instruction**.
- It can also be disabled though the **DI Instruction**.
- It is a vectored interrupt and has a **vector address of 002CH.**

INTR:
- INTR has the **priority lower than RST 5.5.**
- It is **Level triggered.**
- It can only be disabled though the **DI Instruction**.
- **It cannot be masked through the SIM Instruction**.
- It is a **Non-Vectored** interrupt.

Response to INTR:
- When INTR occurs the µP, in response, issues the **first INTA** cycle.
- The **External Hardware sends** an **opcode**, which can be of **RSTn** Instruction **or** of **CALL** instruction.

a) **If** opcode of **RSTn** is sent by the external hardware
   - The µP calculates the address of the ISR as **n*8.**

b) **If** opcode of **CALL** is sent:
   - As Call is a **3-Byte Instruction** the µP send **2 more INTA** signals
   - In response to the **2nd and the 3rd INTA** cycle the external hardware returns the **lower and the higher byte of the address** of the ISR respectively.

Cell: 98204 08217
BharatSir@hotmail.com

- This is how the address is determined when INTR occurs.

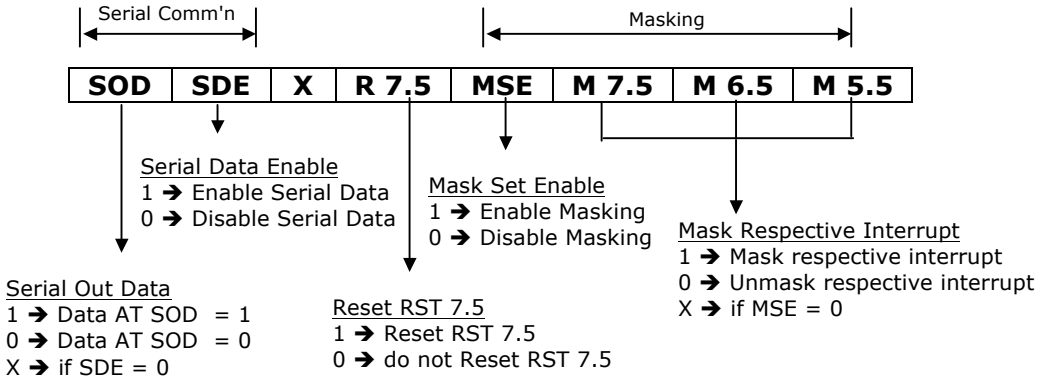
INTA : (Interrupt Acknowledge)
- This is an **active low acknowledge** signal going out of 8085.
- This signal is **given in response to** an interrupt on **INTR ONLY**.
- After the **first INTA** is given, the interrupting **peripheral sends an opcode**.
- **If** the **Opcode** is **of RSTn**, then the **ISR address is calculated as n × 8.**
- **If** the **Opcode** is **of CALL**, then **two more INTA signals** are given and, the lower byte, and then the higher byte of the ISR address are sent by the peripheral.
  #Please refer Bharat Sir's Lecture Notes for this ...

EI and DI Instructions:

INTE F/F : (Interrupt Enable Flip Flop)
- This flip-flop decides if interrupts are enabled in the µP i.e. **if it is set, all interrupts are enabled**.
- It is **set by** the **EI Instruction**.
- It is **reset** in the following 3 ways:
  i. **If µP is reset**.
  ii. If **DI instruction** is executed.
  iii. If **any other interrupt is recognized** by the µP. In this case the INTE F/F is later set in the ISR by EI. ☺ In case of doubts, contact Bharat Sir: - 98204 08217.
- The INTE F/F affects all interrupts **EXCEPT TRAP**, as it cannot be disabled.

SIM Instruction: (Set Interrupt Mask)

This instruction is used for the following purposes:
i. To **Mask** or Un-Mask the **RST7.5, RST6.5 and RST5.5** interrupts.
ii. To send the data out serially (bit - by - bit) through the **SOD** line of the µP.
iii. To **reset RST7.5** interrupt irrespective of whether it is masked or not.



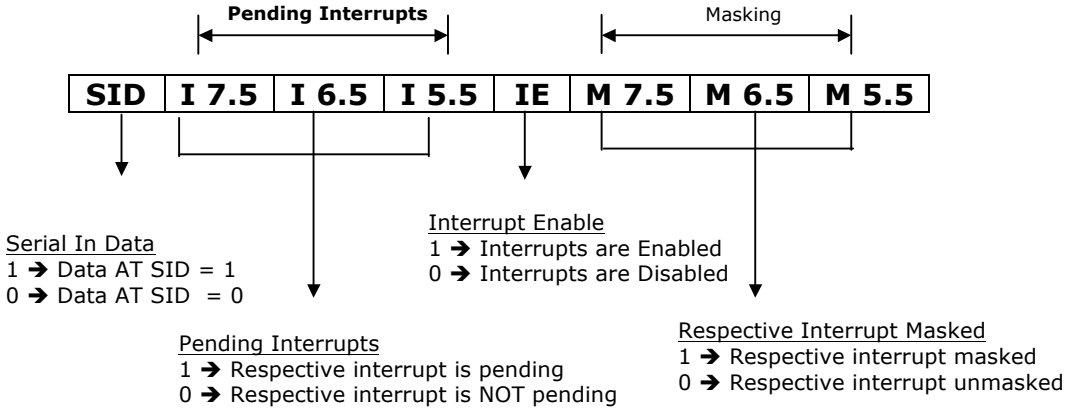
Method of execution:
- The appropriate byte is formed and **loaded into** the **Accumulator**.
- Then the **SIM** Instruction is **executed**.
- The µP reads the contents of the accumulator in the above order.

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

RIM Instruction: (Read Interrupt Mask)

This instruction is used for the following purposes:

i. To **read the Interrupt Mask** of the µP.
ii. To accept data serially through the **SID** pin.
iii. To see the **"Pending Interrupts"** of the µP.



Pending Interrupts:

Pending interrupts are those interrupts, which are waiting to be serviced.
An interrupt becomes pending as a higher priority interrupt is currently being serviced.
RIM Instruction indicates the Pending Status of RST7.5, RST6.5 and RST5.5.

Method of execution:

- Then the **RIM** Instruction is **executed**.
- The **µP loads** the appropriate byte into the **Accumulator.**
- The programmer reads the contents of the Accumulator.

Interrupt Properties

Interrupt	Priority	Triggering	Maskable by SIM	Disabled by DI	Vectored	Vector Address
TRAP	**1**	**Edge / Level**	**No**	**No**	Yes	0024 H
RST 7.5	2	**Edge**	Yes	Yes	Yes	003C H
RST 6.5	3	Level	Yes	Yes	Yes	0034 H
RST 5.5	4	Level	Yes	Yes	Yes	002C H
INTR	5	Level	**No**	Yes	**No**	**Get ISR Address from External Hardware**

Cell: 98204 08217
BharatSir@hotmail.com

Q 14) WAP to generate a SQUARE-WAVE of 1 KHz using SOD pin of 8085.

Soln:

```
BACK: MVI   A, 40H      ; SIM Command = 0100  0000
      SIM
      CALL  DLAY
      MVI   A, C0H      ; SIM Command = 1100  0000
      SIM
      CALL  DLAY
      JMP   BACK
```

For a square wave of 1 KHz, the time period is 1 msec.
Hence the required delay is of 0.5 msec.
Assume 8085 is working at 3 MHZ

```
DLAY: MVI   B, XXH      ; 7 T-states … … … Count is calculated later
BACK: DCR   B           ; 4 T-states … … … Decrement Count
      JNZ   BACK        ; 10T (true) / 7T (false)
      RET               ; 10T-states
```

T_D = MT + [(Count)$_d$ x NT] – 3T
Here MT = Time outside the loop = 17T
NT = Time inside the loop = 14T
T_D = 17T + [(Count)$_d$ x 14T] – 3T
Required T_D = 0.5 msec = 0.5 x 10^{-3} sec
1T = 0.333 µsec = 0.333 x 10^{-6} sec
Substituting the above values we get:
$0.5x10^{-3} = 17x(0.333x10^{-6}) + [(Count)_d \times 14x(0.333x10^{-6})] - 3x(0.333x10^{-6})$
Count = 6AH

Q 15) WAP to transfer the value 35H serially with one start bit "0" and one stop bit "1".

Soln: Serial communication happens bit by bit starting from the LSB.
As per the question, we need to send the start bit (0), then the data and finally the stop bit (1).
Hence a total of 10 bits will move out as follows:

```
  0              1 0 1 0 1 1 0 0           1
Start      8-data bits in reverse order    Stop
```

```
      MVI   A, 40H      ; start bit (0)
      SIM
      MVI   A, C0H      ; send a "1"
      SIM
      MVI   A, 40H      ; send a "0"
      SIM
      MVI   A, C0H      ; send a "1"
      SIM
      MVI   A, 40H      ; send a "0"
      SIM
      MVI   A, C0H
      SIM               ; send a "1"
      SIM               ; send a "1" again
      MVI   A, 40H      ; send a "0"
      SIM
      SIM               ; send a "0" again
      MVI   A, C0H      ; send a "1" as the stop bit
      SIM
      RST1
```

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

8259

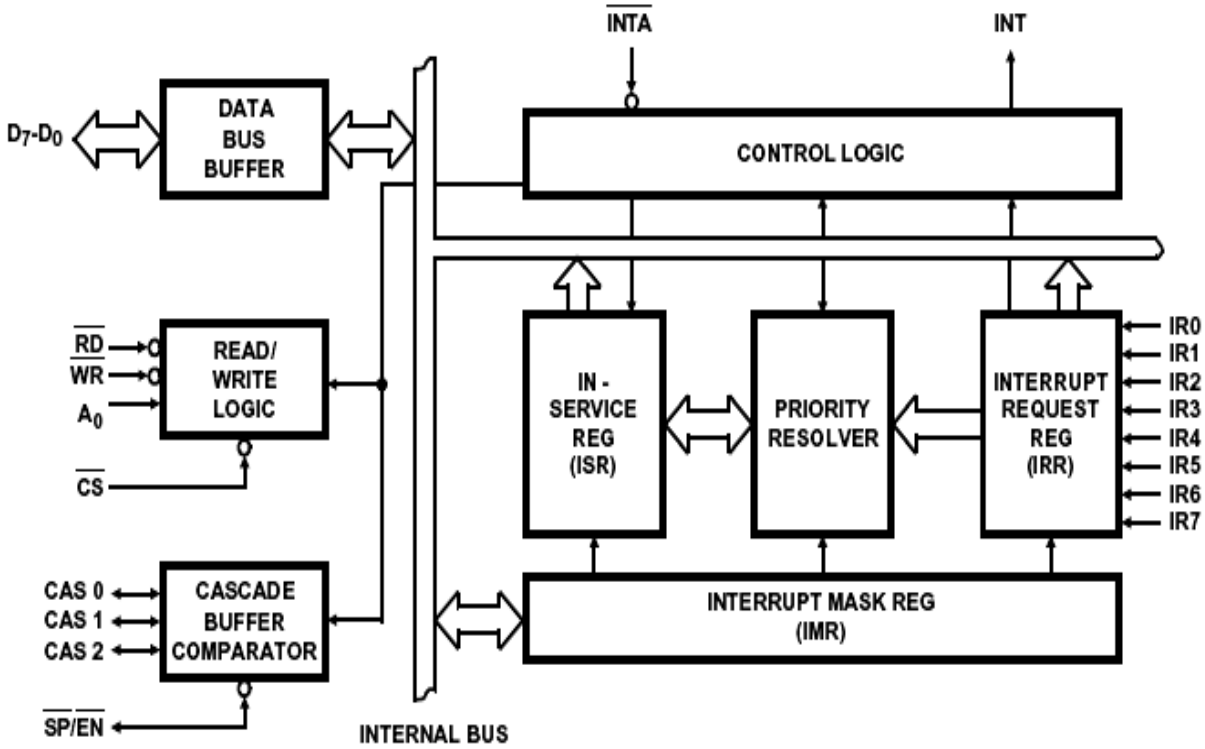
PROGRAMMABLE INTERRUPT CONTROLLER

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

Salient Features of 8259 PIC

- 8259 is a **Programmable Interrupt Controller** (PIC) designed to work with **8085**, **8086** etc.
- A **single 8259** can handle **8 interrupts,** while a **cascaded** configuration of 8 slave 8259's and 1 master 8259 can handle **64 interrupts**.
- 8259 can **handle edge** as well as **level triggered** interrupts.
- 8259 has a **flexible priority** structure.
- In 8259 interrupts can be **masked** individually.
- The **Vector address** of the interrupts is **programmable**.
- Status of interrupts (pending, In-service, masked) can be easily read by the µP.

Architecture of 8259



Cell: 98204 08217
BharatSir@hotmail.com

The architecture of 8259 can be divided into the following parts:

1) Interrupt Request Register (IRR)
- 8259 has **8 interrupt** input lines **IR_7 ... IR_0**.
- The IRR is an **8-bit register** having **one bit** for **each** of the **interrupt** lines.
- When an **interrupt request** occurs on any of these lines, the **corresponding bit** is **set** in the Interrupt Request Register (**IRR**).

2) In-Service Register (InSR)
- It is an **8-bit** register, which **stores** the **level** of the Interrupt Request, which is **currently** being **serviced**.

3) Interrupt Mask Register (IMR)
- It is an **8-bit** register, which stores the **masking pattern** for the interrupts of 8259. It stores **one bit per interrupt level**.

4) Priority Resolver
- It **examines** the **IRR**, **InSR**, and **IMR** and determines which interrupt is of **highest priority** and should be sent to the µP.

5) Control Logic
- It has **INT output** signal **connected to** the **INTR** of the µP, to **send** the **Interrupt** to the µP.
- It also has the **INTA input** signal **connected to** the **INTA** of the µP, to **receive** the interrupt **acknowledge**.
- It is also used to control the remaining blocks.

6) Data Bus Buffer
- It is a bi-directional buffer used to **interface** the internal **data bus** of 8259 with the external (system) data bus.

7) Read/Write Logic
- It is used to accept the RD, WR, A_0 and CS signal.
- It also holds the Initialization Command Words (ICW's) and the Operational Command Words (OCW's).

8) Cascade Buffer / Comparator
- It is used in **cascaded mode** of operation.
- It has two components:

i. CAS_2, CAS_1, CAS_0 lines:
- These lines are **output for the master**, **input for the slave**.
- The **Master sends** the **address of** the **slave** on these lines (hence output).
- The **Salves read** the **address** on these lines (hence input).
- As there are 8 interrupt levels for the Master, there are **3 CAS lines ($\because$ $2^3 = 8$).**

ii. SP/EN (Slave Program/Master Enable):
- In **Buffered Mode**, it **functions** as the **EN line** and is used to **enable** the **buffer**.
- In **Non buffered mode**, it **functions** as the **SP output line**.
- **For Master** 8259 **SP** should be **high**, and **for** the **Slave SP** should be **low**.

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

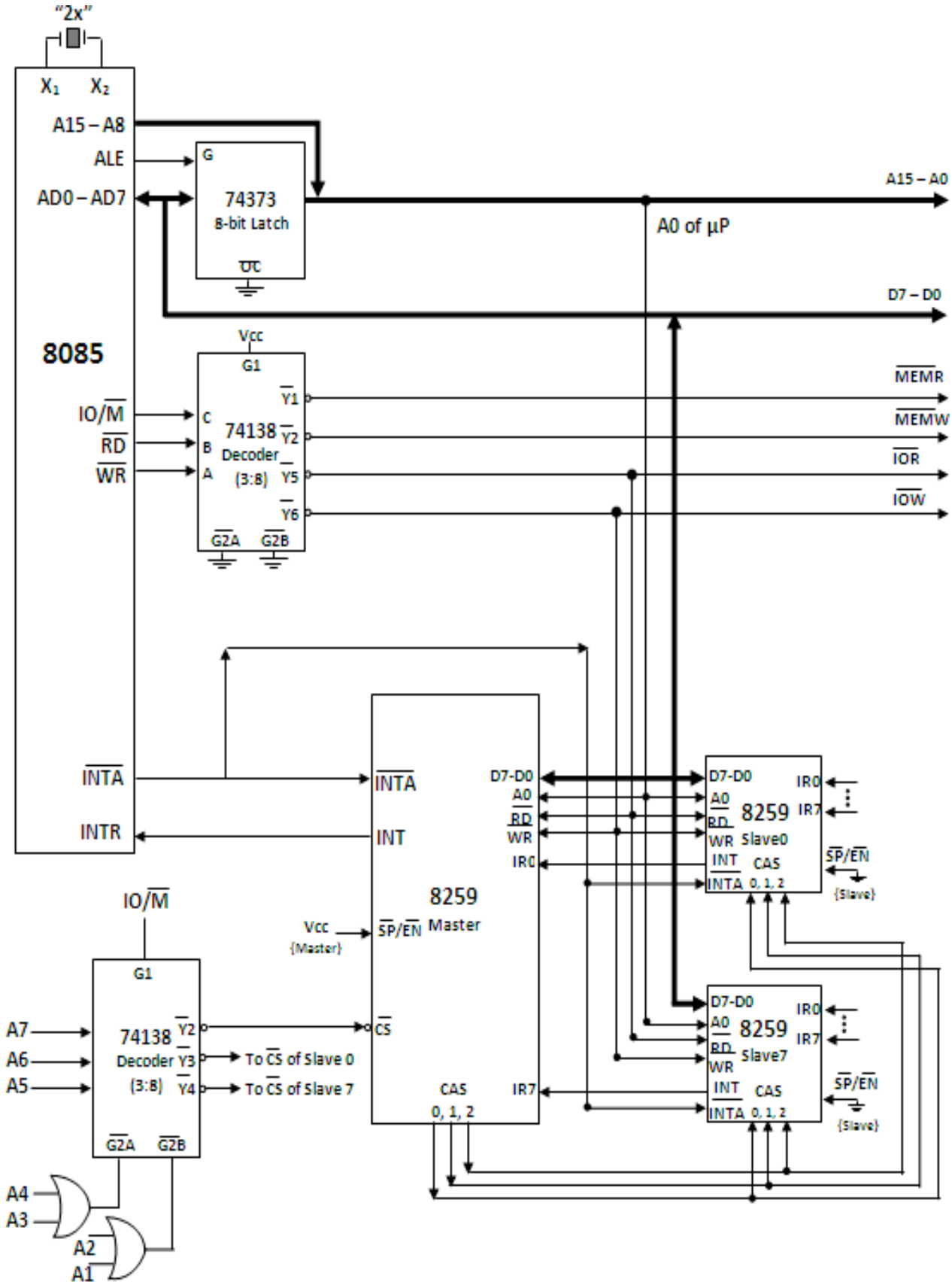
Interfacing and Interrupt Sequence for CASCADED 8259

- When **more than one 8259s** are connected to the µP, it is called as a **Cascaded configuration**.
- A Cascaded configuration **increases** the **number of interrupts** handled by the system.
- As the **maximum** number of **8259s** interfaced can be **9** (1 Master and 8 Slaves) the **Maximum** number of **interrupts** handled can be **64**.
- The **master 8259** has **SP/EN = +5V** and the **slave** has **SP/EN = 0V**.
- **Each slave's INT** output is **connected** to the **IR input** of the **Master**.
- The **INT** output **of** the **Master** is **connected** to the **INTR** input of the **µP**.
- The **master addresses** the individual **slaves through** the **CAS_2, CAS_1, CAS_0 lines** connected from the master to each of the slaves.
- **Each 8259** (Master or Slave) **has** its **own address** and **has to be initialized separately**.

When an **interrupt request** occurs **on** a **SLAVE**, the **following sequence** of events is executed:

1) The **slave 8259 resolves** the **priority** of the interrupt and **sends** the **interrupt to** the **master** 8259.
2) The **master resolves** the **priority** among its slaves and **sends** the **interrupt to** the **µP**.
3) The **µP finishes the current instruction** and **responds** to the interrupt **by sending 3 INTA pulses**.
4) **In response to the first INTA pulse** the **master does** the **following** tasks:
   i. It **sends** the **opcode** of **CALL** instruction **to** the **µP** on the data bus.
   ii. It **sends** the **3-bit salve identification number** on the **CAS lines**.
   iii. The **Master sets** the **corresponding bit** in **its InSR**.
   iv. The **Slave identifies** its number on the **CAS lines** and **sets** the **corresponding bit in its InSR**.
5) In **response to** the **second and third INTA** pulse the **slave** places the **lower byte and** then the **higher byte of** the **ISR address** on the data bus **respectively**.
6) **During** the **third INTA** pulse the **InSR bit** of the **slave** is **cleared** in **AEOI mode**, **otherwise** it is **cleared by** the **EOI command** at the end of the ISR.
7) The **µP pushes** the contents of **PC onto** the **Stack** and **transfers program** to the **address of** the **ISR sent by** the **slave** 8259. **The ISR thus begins**.

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com



BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

I/O for the Cascaded Configuration

I/O Chip	Address Bus								I/O port Address
	A7	A6	A5	A4	A3	A2	A1	A0	
8259 Master									
ICW1	**0**	**1**	**0**	0	0	0	0	**0**	**40 H**
ICW2	**0**	**1**	**0**	0	0	0	0	**1**	**41 H**
8259 Slave 0									
ICW1	**0**	**1**	**1**	0	0	0	0	**0**	**60 H**
ICW2	**0**	**1**	**1**	0	0	0	0	**1**	**61 H**
8259 Slave 7									
ICW1	**1**	**0**	**0**	0	0	0	0	**0**	**80 H**
ICW2	**1**	**0**	**0**	0	0	0	0	**1**	**81 H**

PRIORITY MODES OF 8259

Fully Nested Mode (FNM)

It is the **default mode** of 8259.
It is a **fixed priority** mode.
IR_0 has the **highest** priority and **IR_7** has the **lowest** priority.
It is preferred for **"Single"** 8259.

Special Fully Nested Mode (SFNM)

This mode can be **used for** the **Master 8259 in** a **cascaded** configuration.
Its **priority structure** is fixed and is the **same as FNM** (IR_0 highest and IR_7 lowest).
Additionally, in SFNM, the **Master would recognize a higher priority interrupt from a slave,** *whose another interrupt is currently being serviced*. This is **possible only in SFNM**.

Rotating Priority Modes

There are **two** rotating priority modes:
Automatic Rotation and Specific Rotation

Automatic Rotation Mode

This is a rotating priority mode.
It is **preferred** when **several interrupt** sources are of **equal priority**.
In this mode, **after** a device receives **service**, it **gets** the **lowest priority**.
All other priorities rotate subsequently.
Eg: If IR_2 is has just been serviced, it will get the lowest priority.

Specific Rotation Mode

It is **also** a **rotating** priority mode, **but here** the **user can select** any **IR level for lowest priority**, and thus fix all other priorities.

Special Mask Mode (SMM)

Usually 8259 **prevents interrupt requests lower or equal** to the interrupt, which is **currently** in service.
In SMM 8259 **permits interrupts** of **all levels** (lower or higher) **except** the one **currently** in service.
As we are specially masking the current interrupt, it is called Special Mask Mode.
This mode is preferred when we don't want priority

Poll Mode

Here the **INT line** of 8259 is **not used** hence 8259 cannot interrupt the **µp**.
Instead, the **µP will give Poll command** to 8259 using OCW3.
In **return**, **8259 provides** a **Poll Word** to the µP.
The Poll Word **indicates** the **highest priority interrupt**, which requires service.

Poll Word

I	x	x	x	x	W_2	W_1	W_0

1= Valid Interrupt
0 = No valid interrupt

Level No of the highest priority interrupt to be serviced

W_2	W_1	W_0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Thereafter the **µP services the interrupt**. **For doubts contact Bharat Sir on 98204 08217**
Advantage: The µP's program is not disturbed. It can be used when the ISR is common for several Interrupts. It can be used to increase the total number of interrupts beyond 64.
Drawback: If the polling interval is too large, the interrupts will be serviced after long intervals. If the polling interval is small, lot of time may be wasted in unnecessary polls.

Buffered Mode

In this mode $\overline{\textbf{SP}}$ / $\overline{\textbf{EN}}$ becomes **low** during $\overline{\textbf{INTA}}$ cycle.
This signal is used to enable the buffer.

EOI – (End Of Interrupt)

When the **µP responds** to an interrupt request by **sending the first** $\overline{\textbf{INTA}}$ signal, the **8259 sets** the **corresponding bit** in the In Service Register (**InSR**).
This **begins** the **service** of the interrupt.
When this bit in the In Service Register is **cleared**, **it is called** as **End of Interrupt (EOI)**.

EOI Modes:

1) Normal EOI Mode:

Here an EOI Command is necessary. The EOI Command is given by the programmer at the end of the ISR. It causes 8259 to clear the bit from In Service Register. There are two types of EOI Commands:

Non Specific EOI Command:

Here the programmer doesn't specify the Bit number to be cleared. 8259 automatically clears the highest priority bit from In Service Register.

Specific EOI Command:

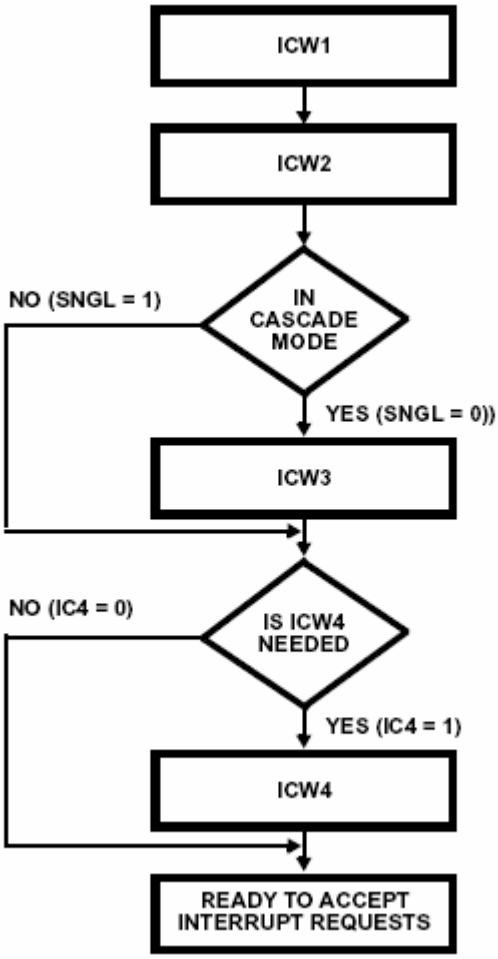
Here the programmer specifies the Bit number to be cleared from In Service Register.

2) Auto EOI Mode (AEOI):

In AEOI mode the EI command is not needed. Instead, 8259 will itself clear the corresponding bit from In Service Register at the end of the 2nd $\overline{\textbf{INTA}}$ pulse.

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

Initialization of 8259



As seen above there are **two types** of **control words**, **Initialization Control Words (ICWs)** and **Operational Control Words (OCWs).**

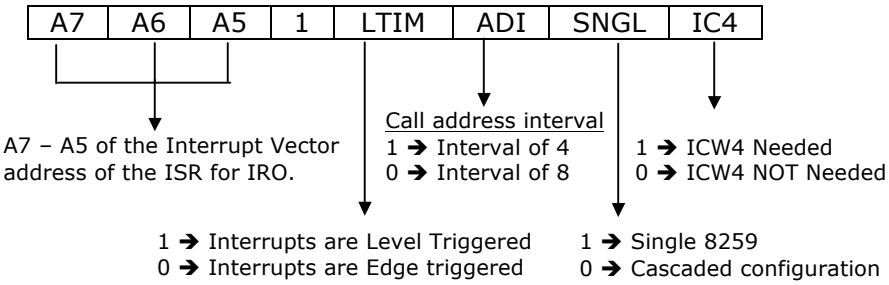
ICWs

- **ICWs** have to be **given during** the **initialization** of 8259 (i.e. **before** the µP can start **using 8259**).
- **ICW1** and **ICW2** are **compulsory**.
- **If Cascaded**, **ICW3** has to be given.
- Whether **ICW4** is **required** or not, is **specified in the ICW1**.
- **If ICW4** is **required**, it has to be **written**.
- It is **important** that the ICWs are **written in** the **above sequence only**.

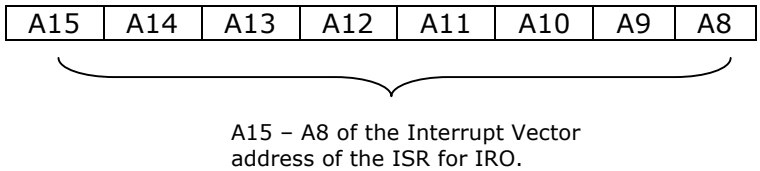
OCWs

- **OCWs** are **given during** the **operation** of 8259 (i.e. **after** the µP has **started using** 8259).
- **OCWs** are **neither compulsory**, **nor** do they have a **specific sequence**.
- They are mainly used to alter the **masking** status and the **operation modes** of 8259.

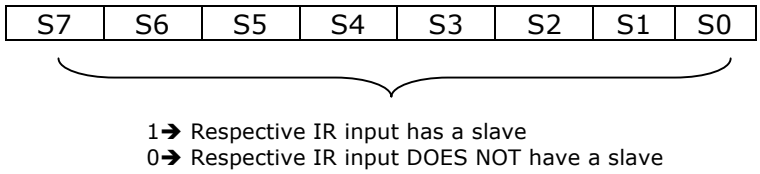
ICW-1 *(A0 = 0)*



ICW-2 *(A0 = 1)*



ICW-3 MASTER *(A0 = 1)*

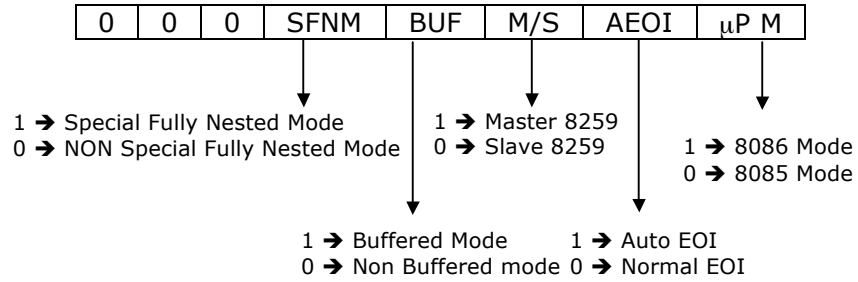


ICW-3 SLAVE *(A0 = 1)*

0	0	0	0	0	**ID_2**	**ID_1**	**ID_0**

ID_2	ID_1	ID_2	Slave ID
0	0	0	**0**
0	0	1	**1**
0	1	0	**2**
0	1	1	**3**
1	0	0	**4**
1	0	1	**5**
1	1	0	**6**
1	1	1	**7**

ICW-4 *(A0 = 1)*



Cell: 98204 08217
BharatSir@hotmail.com

OCW-1 *(A0 = 1)*

M7	M6	M5	M4	M3	M2	M1	M0

1➔ MASK Respective IR input
0➔ UNMASK Respective IR input

OCW-2 *(A0 = 0)*

R	SL	EOI	**0**	**0**	L2	L1	L0

R	SL	EOI	Action	
0	0	1	NON Specific EOI Command	**End Of Interrupt**
0	1	1	Specific EOI Command	
1	0	1	Rotate on NON Specific EOI	**Auto rotation**
1	0	0	Rotate in AUTO EOI Mode (Set)	
0	0	0	Rotate in AUTO EOI Mode (Clear)	
1	1	1	Rotate on Specific EOI Command	**Specific rotation**
1	1	0	Set Priority command	
0	1	0	NOP	

L2	L1	L0	IR Level
0	0	0	IR0
0	0	1	IR1
0	1	0	IR2
0	1	1	IR3
1	0	0	IR4
1	0	1	IR5
1	1	0	IR6
1	1	1	IR7

OCW-3 *(A0 = 0)*

X	ESMM	SMM	**0**	**1**	P	RR	RIS

1 ➔ POLL Command
0 ➔ No Poll

ESMM	SMM	Action
0	0	No Action
0	1	
1	0	Exit SMM
1	1	Enter SMM

RR	RIS	Action
0	0	No Action
0	1	
1	0	Read IRR on next RD pulse
1	1	Read InSR on next RD pulse

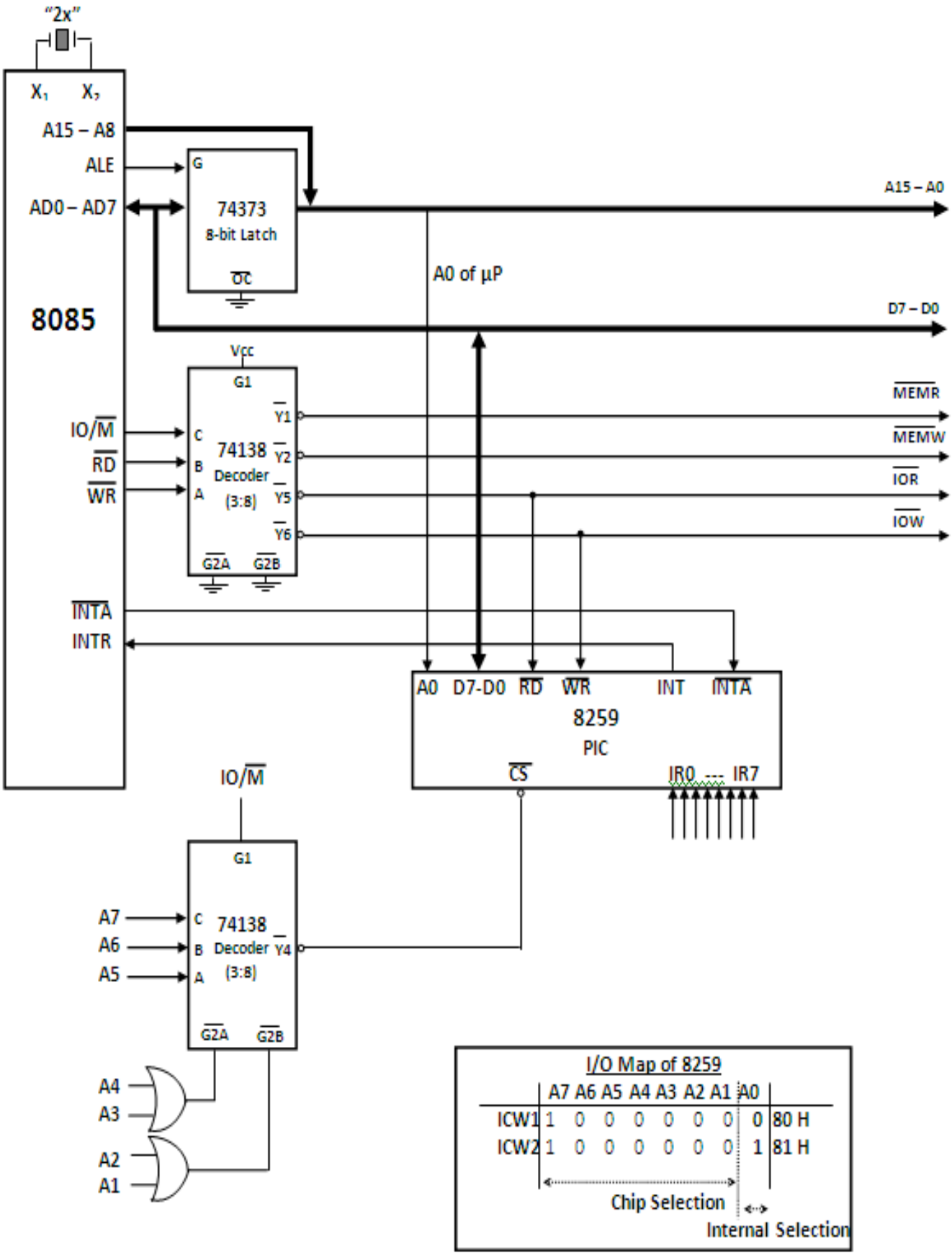
BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

Interfacing and Interrupt Sequence for a SINGLE 8259

INTE flip-flop of the µP is **set** by the **EI** instruction and the **individual interrupts enabled** using **SIM** instruction.
8259 is **initialized** by giving **ICW1** and **ICW2** (compulsory) and **ICW4** (optional). Note that **ICW3** is **not given** as Single 8259 is used. OCWs are given if required.
Once 8259 is **initialized**, the **following sequence** of events takes place when one or more **interrupts occur** on the IR lines of the 8259.

1) The **corresponding bit** for an interrupt is **set in IRR**.
2) The **Priority Resolver checks** the 3 registers:
   IRR (for highest interrupt request)
   IMR (for the masking Status)
   InSR (for the current level serviced)
   and **determines** the **highest priority** interrupt.
   It **sends** the **INT** signal **to** the **µP**.
3) The **µP finishes** the **current instruction** and **acknowledges** the interrupt by **sending** the **first INTA pulse**.
4) On receiving the INTA signal, the **corresponding bit** in the **InSR** is **set** (indicating that the service of this interrupt is started) and the **bit** in the **IRR** is **reset** (to indicate that the request is accepted).
   8259 now **sends** the **Opcode of CALL** instruction **to the µP** on the data bus.
5) The **µP decodes** the **CALL instruction** and **sends 2 more INTA pulses** to the 8259.
6) In response to the two INTA pulses, the **8259 sends the address of the ISR to the µP**. First the lower byte and then the higher byte.
7) Thus, the complete 3-byte CALL Instruction code is released by the 8259.
   In the **AEOI Mode** the **InSR bit is reset** at this point, **otherwise** it **remains set until** an **appropriate EOI** command is given at the End of the ISR.
8) The **µP pushes** the contents of **PC** onto the **Stack** and **transfers program to** the **address** of the **ISR** sent by the 8259. **The ISR thus begins**. *#Please refer Bharat Sir's Lecture Notes for this ...*

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com



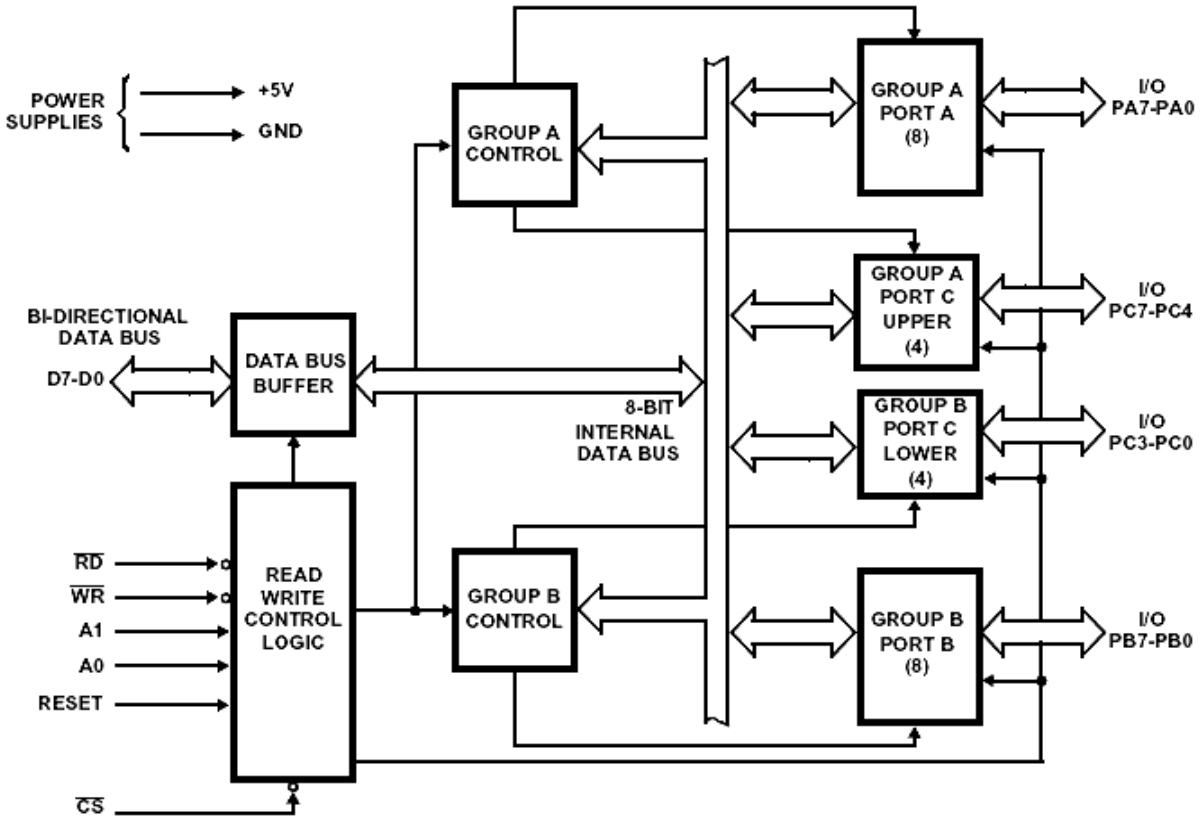
8255

PROGRAMMABLE PERIPHERAL INTERFACE

Salient Features of 8255 PPI

- It is a programmable general-purpose I/O device.
- It has 3 8-bit bi-directional I/O ports - **Port A**, **Port B**, and **Port C**.
- It provides **3 modes of data transfer** – Simple I/O, Handshake I/O and Bi-directional Handshake I/O.
- Additionally it also provides a **Bit Set Reset Mode** to alter individual bits of **Port C**, for bit interface devices.

Architecture of 8255



The architecture of 8259 can be divided into the following parts:

1) Data Bus Buffer

- This is a 8-bit bi-directional buffer used to interface the internal data bus of 8255 with the external (system) data bus.
- The CPU transfers data to and from the 8255 through this buffer.

2) Read/Write Control Logic

- It accepts address and control signals from the µP.
- The Control signals determine whether it is a read or a write operation and also select or reset the 8255 chip.
- The Address bits (A_1, A_0) are used to select the Ports or the Control Word Register as shown:

A_1	A_0	Selection	Sample address
0	0	Port A	80 H (i.e. 1000 00**00**)
0	1	Port B	81 H (i.e. 1000 00**01**)
1	0	Port C	82 H (i.e. 1000 00**10**)
1	1	Control Word	83 H (i.e. 1000 00**11**)

- The Ports are controlled by their respective Group Control Registers.

3) Group A Control

- This Control block controls Port A and Port C_{Upper} i.e. PC_7-PC_4.
- It accepts Control signals from the Control Word and forwards them to the respective Ports.

4) Group B Control

- This Control block controls Port B and Port C_{Lower} i.e. PC_3-PC_0.
- It accepts Control signals from the Control Word and forwards them to the respective Ports.

5) Port A, Port B, Port C

- These are 8-bit Bi-directional Ports.
- They can be programmed to wok in the various modes as follows:

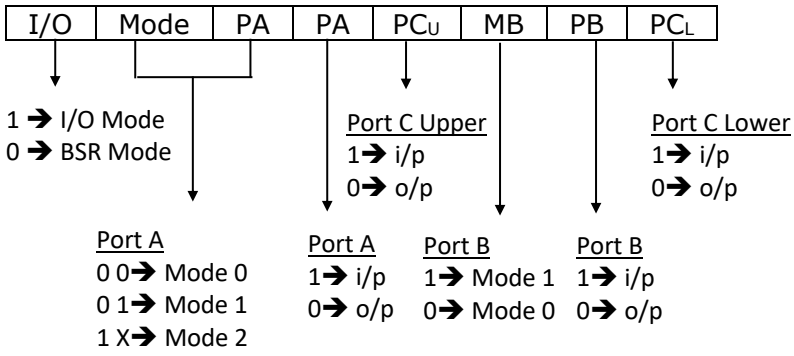
Port	Mode 0	Mode 1	Mode 2
Port A	✔	✔	✔
Port B	✔	✔	✘ (Mode 0 or Mode 1)
Port C	✔	✘ (Handshake signals)	✘ (Handshake signals)

- ONLY Port C can also be programmed to work in Bit Set reset Mode to manipulate its individual bits.

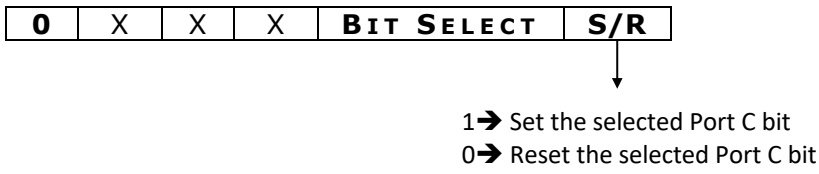
☺ In case of doubts, contact Bharat Sir: - 98204 08217.

6) Control Word of 8255 (I/O Mode)

To do 8-bit data transfer using the Ports A, B or C, 8255 needs to be in the IO mode.
The bit pattern for the control word in the IO mode is as follows:



7) Control Word of 8255 (BSR Mode – Applicable ONLY for Port C)



Bit Select			Bit	
0	0	0	PC_0	Select respective bit of Port C
0	0	1	PC_1	
0	1	0	PC_2	
0	1	1	PC_3	
1	0	0	PC_4	
1	0	1	PC_5	
1	1	0	PC_6	
1	1	1	PC_7	

- The BSR Mode is used ONLY for Port C.
- In this Mode the individual bits of Port C can be set or reset.
- This is very useful for interfacing those devices, which accept bit-wise data.
  Eg: ADC Converters.
- The individual bit is selected and Set/reset through the control word.
- Since the D7 bit of the Control Word is 0, the BSR operation will not affect the I/O operations of 8255.

DATA TRANSFER MODES OF 8255

❖ Mode 0 (Simple Bi-directional I/O)

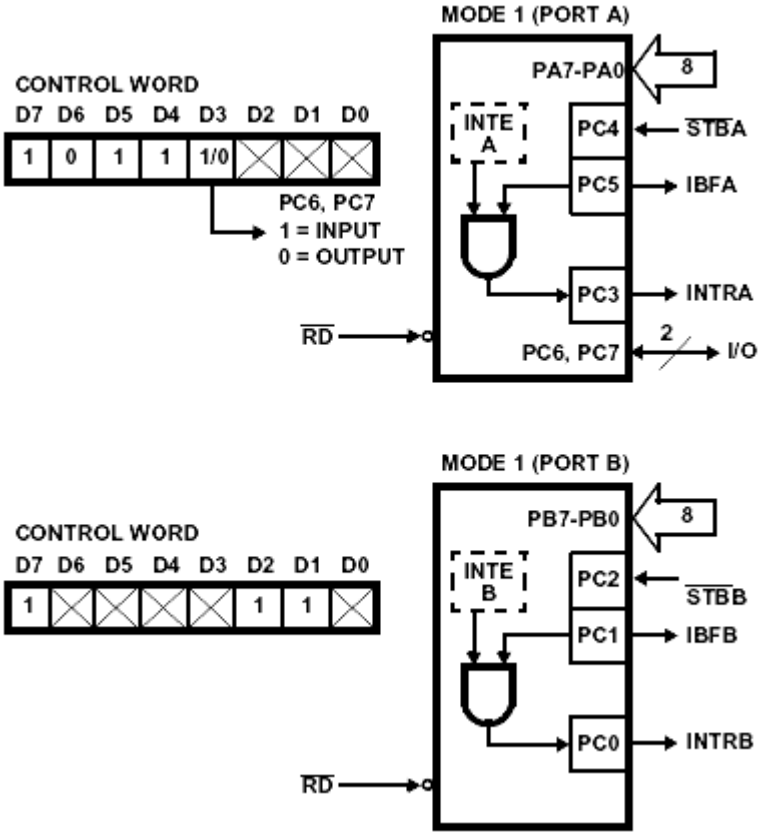
- Port A and Port B used as 2 Simple 8-bit I/O Ports.
- Port C is used as 2 simple 4-bit I/O Ports.
- Each port can be programmed as input or output individually.
- Ports do not have handshake or interrupting capability.
- Hence, **slower** devices cannot be interfaced.

❖ Mode 1 (Handshake I/O)

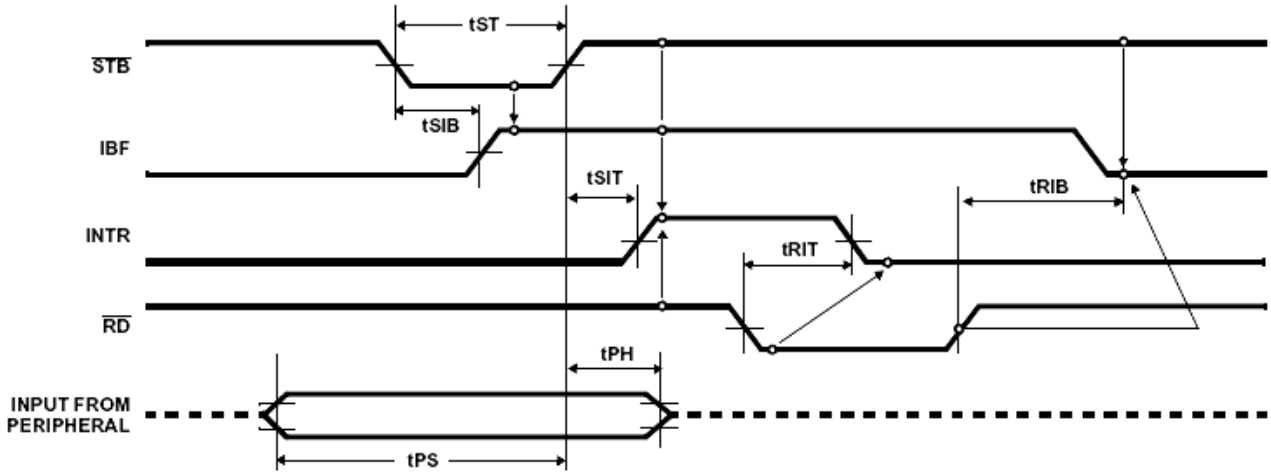
- In Mode 1, handshake signals are exchanged between the devices before the data transfer takes place.
- Port A and Port B used as 2 8-bit I/O Ports that can programmed in Input OR in output mode.
- Each Port uses 3 lines from Port C for handshake. The remaining lines of Port C can be used for simple IO.
- **Interrupt driven** data transfer and **Status driven** data transfer possible.
- Hence, **slower** devices can be interfaced.

The handshake signals are different for input and output modes.
#Please refer Bharat Sir's Lecture Notes for this ...

♦ Mode 1 (Input Handshaking)



Timing Diagram for Mode 1 Input Transfer



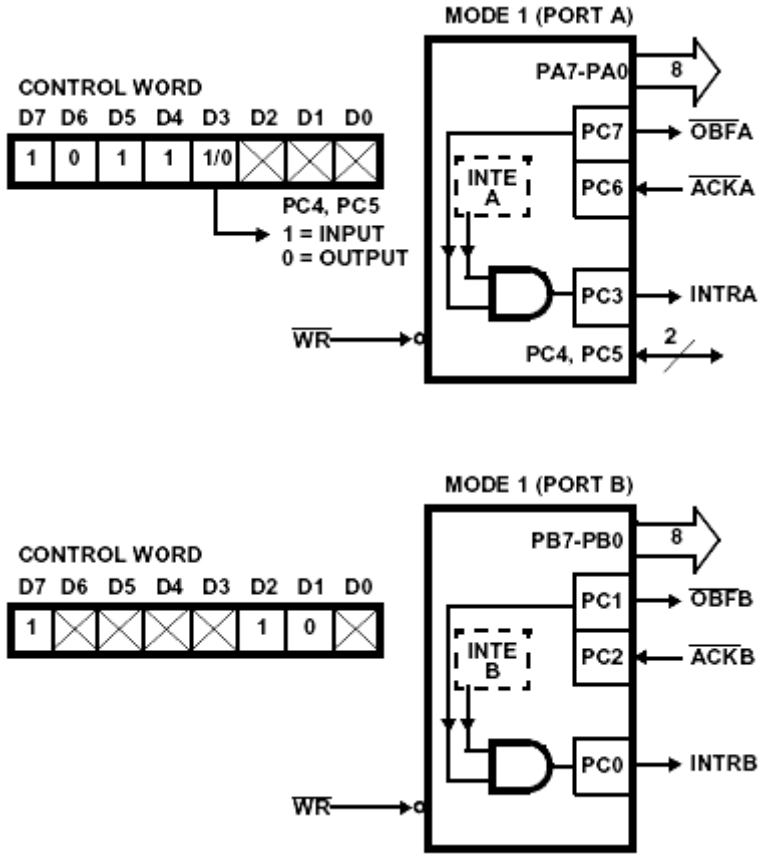
Working:

- **Each port** uses **3 lines** of **Port C** for the following signals:
  STB (Strobe), **IBF** (Input Buffer Full) ➔ Handhsake signals
  INTR (interrupt) ➔ Interrupt signal
- Additionally the **RD** signal of 8255 is also used.

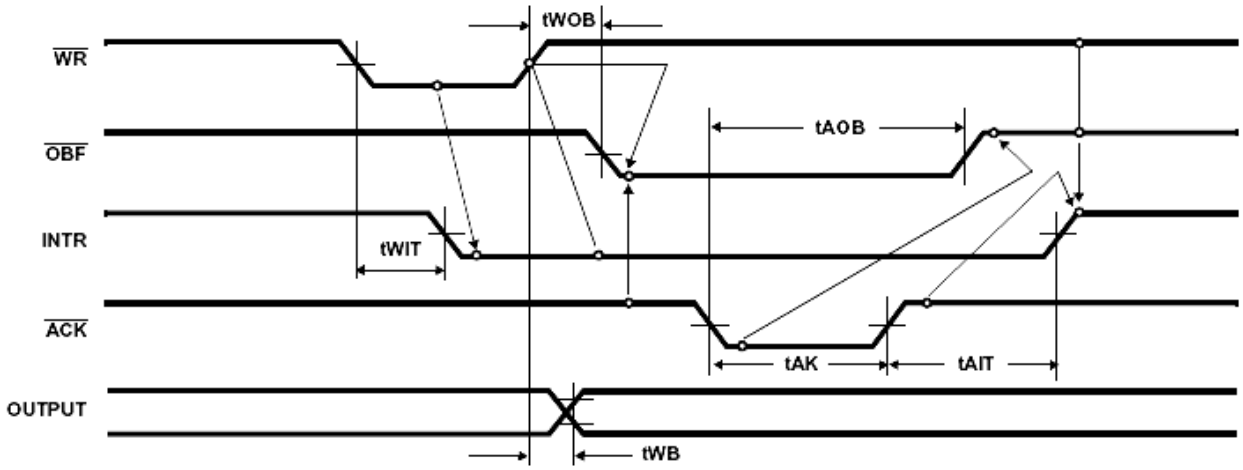
Handshaking takes place in the following manner:

1) The **peripheral** device **places data** on the Port **bus** and informs the Port by **making STB low**.
2) The **input Port accepts** the **data** and informs the peripheral to wait by making **IBF high**. This **prevents** the peripheral from **sending more data** to the 8255 and **hence data loss** is prevented. ☺ In case of doubts, contact Bharat Sir: - 98204 08217.
3) **8255 interrupts** the **µP** through the **INTR** line provided the INTE flip-flop is set.
4) **In response** to the Interrupt, the **µP issues** the **RD** signal and **reads** the **data**. The **data byte** is **thus transferred** to the **µP**.
5) Now, the **IBF** signal **goes low** and the peripheral can **send more data** in the above sequence.

♦ Mode 1 (Output Handshaking)



Timing Diagram for Mode 1 Output Transfer



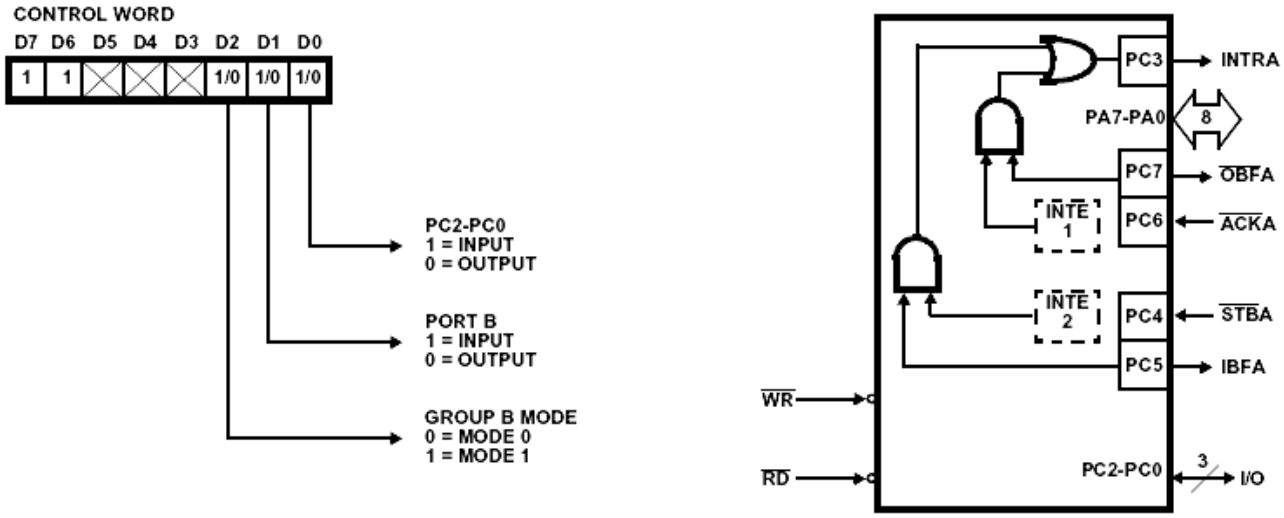
Working

- **Each port** uses **3 lines** of **Port C** for the following signals:
  OBF (Output Buffer Full), **ACK** (Acknowledgement) ➔ Handhsake signals
  INTR (interrupt) ➔ Interrupt signal
- Additionally the **WR** signal of 8255 is also used.

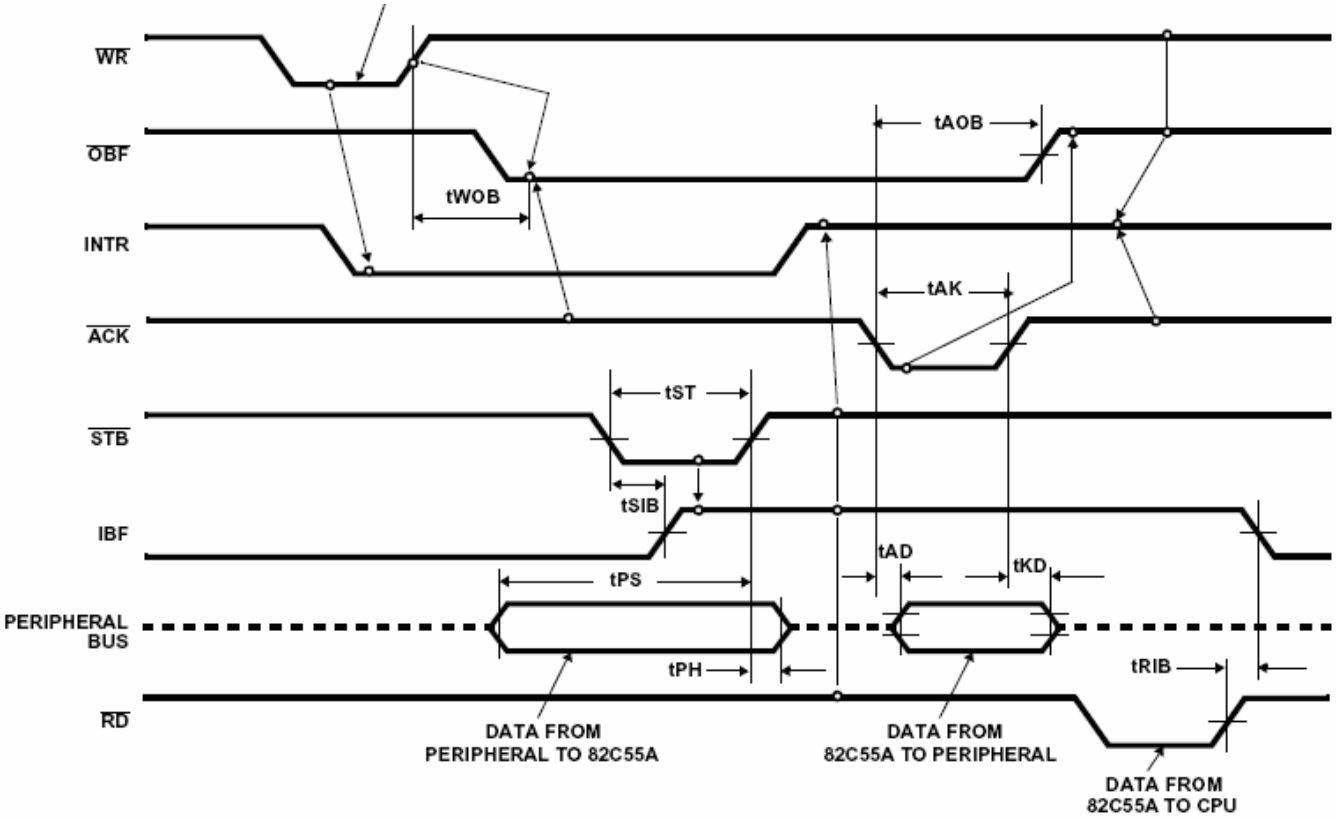
Handshaking takes place in the following manner:

1) When the output port is empty (indicated by a high on the INTR line), the **µP writes data** on the output port by giving the **WR** signal.
2) As soon as the WR operation is complete, the **8255 makes the INTR low**, indicating that the µP should **wait**.
   This **prevents** the µP from **sending more data** to the 8255 and **hence data loss** is prevented.
3) **8255** also **makes** the **OBF low** to indicate to the output peripheral that **data** is **available** on the data bus.
4) The **peripheral accepts** the **data** and sends an acknowledgement by making the **ACK low**.
   The **data byte** is **thus transferred** to the peripheral.
5) Now, the **OBF** and **ACK** lines **go high**.
6) The **INTR** line **becomes high** to **inform** the **µP** that **another byte** can be **sent.** i.e. the output port is empty.
   This process is repeated for further bytes.

❖ Mode 2 (Bi-directional Handshake I/O)



Timing Diagram for Mode 2 Bi-Directional Transfer

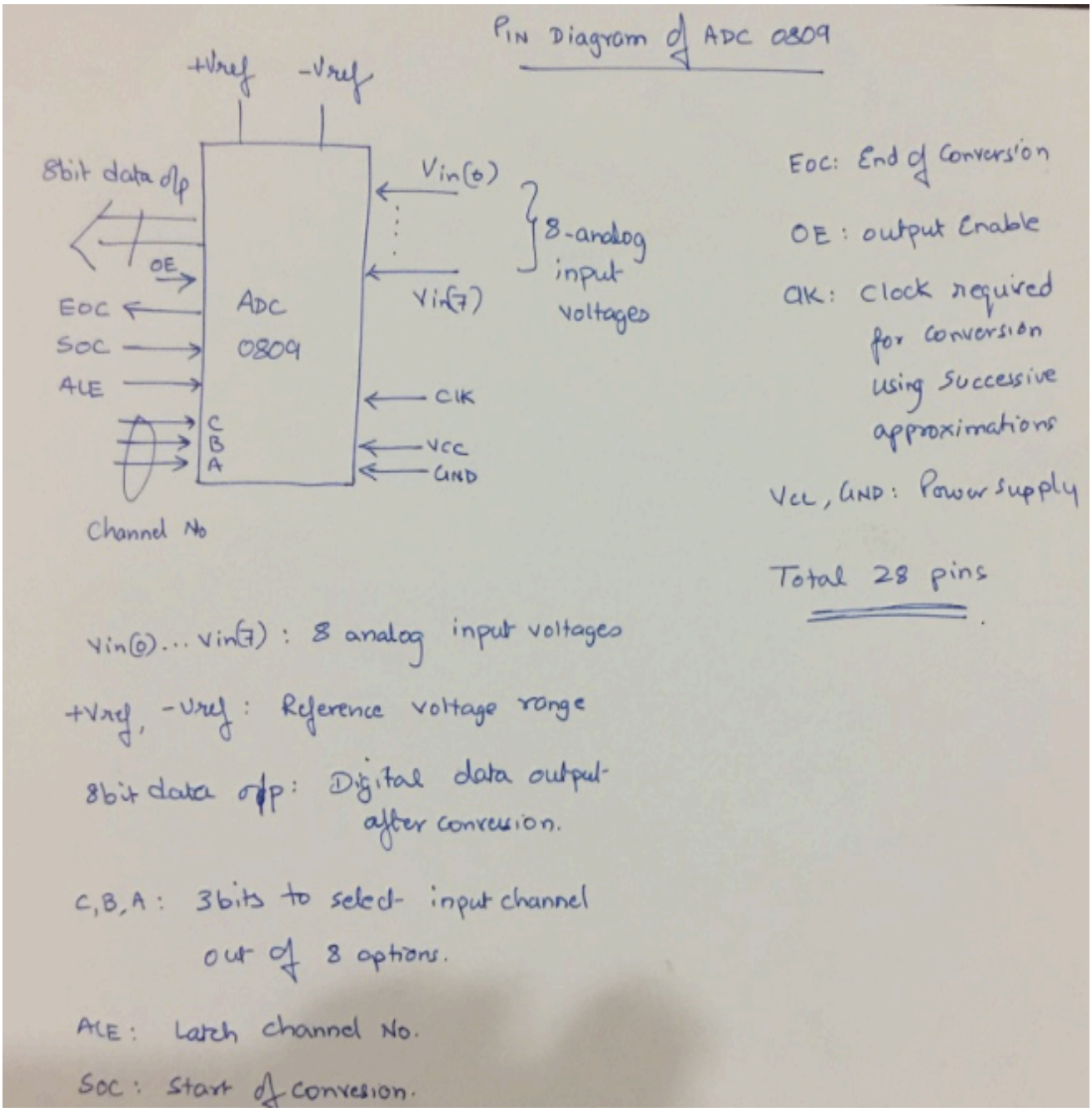


Working:

- In this mode, **Port A** is **used** as an **8-bit bi-directional Handshake I/O Port**.
- **Port A** requires **5 signals from Port C** for doing Bi-directional handshake.
- **Port B** has the following **options**:
  1) **Use** the **remaining 3 lines** of **Port C** for handshaking so that **Port B is in Mode 1**.
     Here **Port C** lines will be **completely used for handshaking** (5 by Port A and 3 by Port B).
     OR
  2) **Port B** works in **Mode 0** as simple I/O.
     In this case the **remaining 3 lines** of **Port C** can be used for **data transfer**.

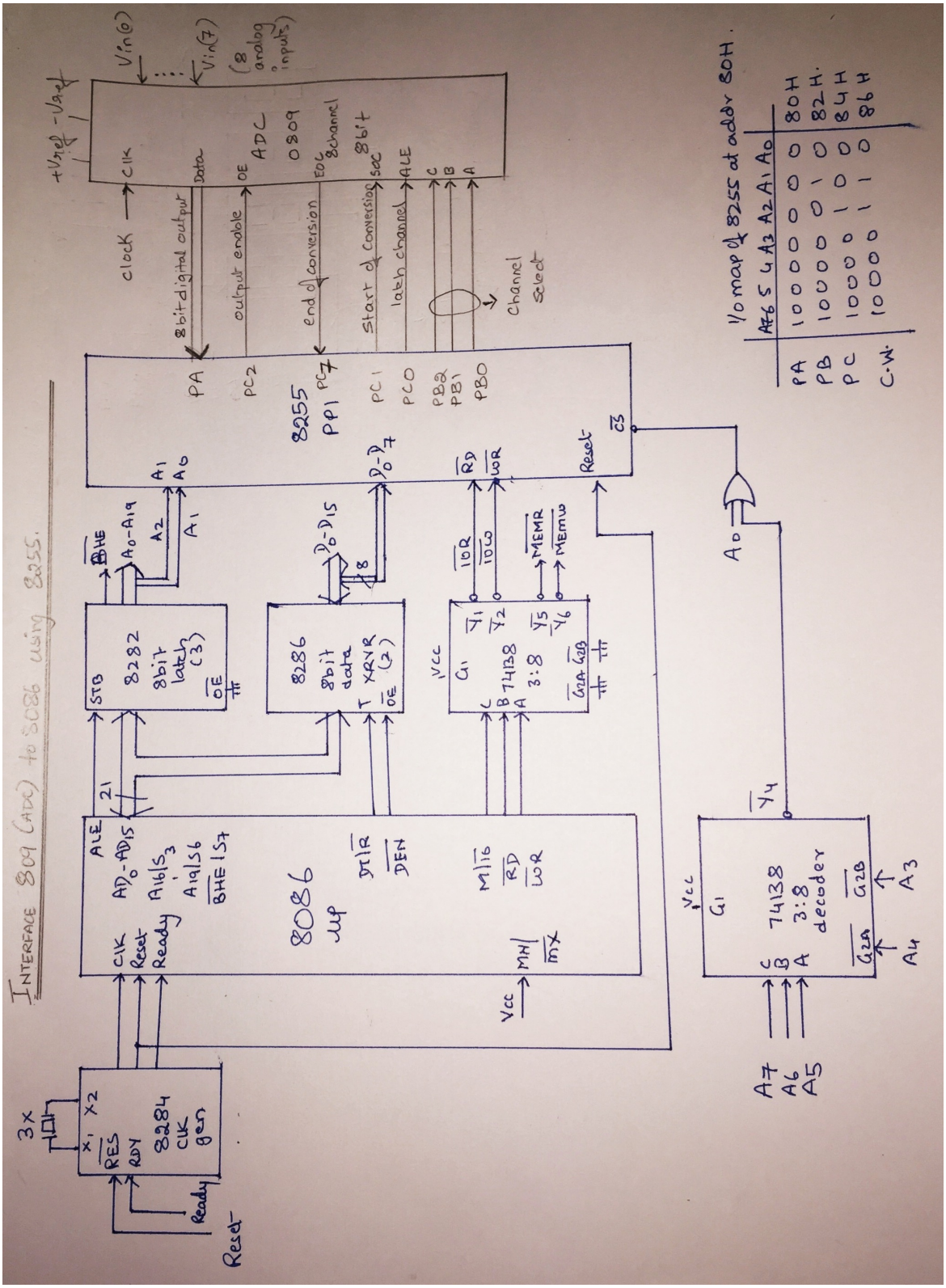
- Port A can be used for data transfer between two computers as shown.
- The high-speed computer is known as the master and the dedicated computer is known as the slave.
- Handshaking process is similar to Mode 1.
- For **Input**:
  - **STB** and **IBF** → handshaking signals
  - **INTR** → Interrupt signal.
- For **Output**:
  - **OBF** and **ACK** → handshaking signals
  - **INTR** → Interrupt signal.
- Thus the 5 signals used from Port C are:
  STB, IBF, INTR, OBF and **ACK**.

ADC 0809


Pin Diagram of ADC 0809
+Vref
-Vref
8bit data o/p
OE
EOC
SOC
ALE
C
B
A
ADC
0809
Vin(0)
Vin(7)
8-analog input voltages
CLK
Vcc
GND
Channel No
EOC: End of Conversion
OE: output Enable
CLK: Clock required for conversion using successive approximations
Vcc, GND: Power supply
Total 28 pins
Vin(0)... Vin(7): 8 analog input voltages
+Vref, -Vref: Reference voltage range
8bit data o/p: Digital data output after conversion.
C,B,A: 3bits to select input channel out of 8 options.
ALE: Latch channel No.
SOC: Start of conversion.

Interface ADC 0809 to 8086 using 8255

1) ADC 0809 is an **8 channel, 8 bit ADC**.
2) It can **convert** an analog **voltage** input **into** an 8 bit digital **data** output.
3) To select an input out of 8 options, there are **three select lines** (C, B and A).
4) We **put a channel number** on these lines (0…7) and latch it using ALE.
5) Now we **give SOC** indicating start of conversion.
6) The channel voltage is internally **sampled** and held into a capacitor.
7) Conversion takes place internally using "**Successive Approximations** Algorithm".
8) **Reference voltage** for conversion is provided using **+Vref and –Vref**.
9) The **clock supply** needed for conversion is given through **CLK** (typically ~ 1MHz).
10) The **end of conversion** is indicated by the ADC using **EOC signal**.
11) Now we **give the OE** signal enabling 8-bit data output from the **ADC to 8255**.
12) This data from 8255 is now **transferred to the microprocessor**.
13) The process is repeated for **subsequent channels**, by changing the channel number.
14) The ADC could also be connected directly to 8086 but **using an 8255 just makes it easier** as the **port lines of 8255 can control various functions of the ADC.**
15) ADCs have a **vast use** in the modern electronic world for **Data Acquisition Systems**.
16) They can be used for **temperature sensing, voice recording, speed sensing** etc.

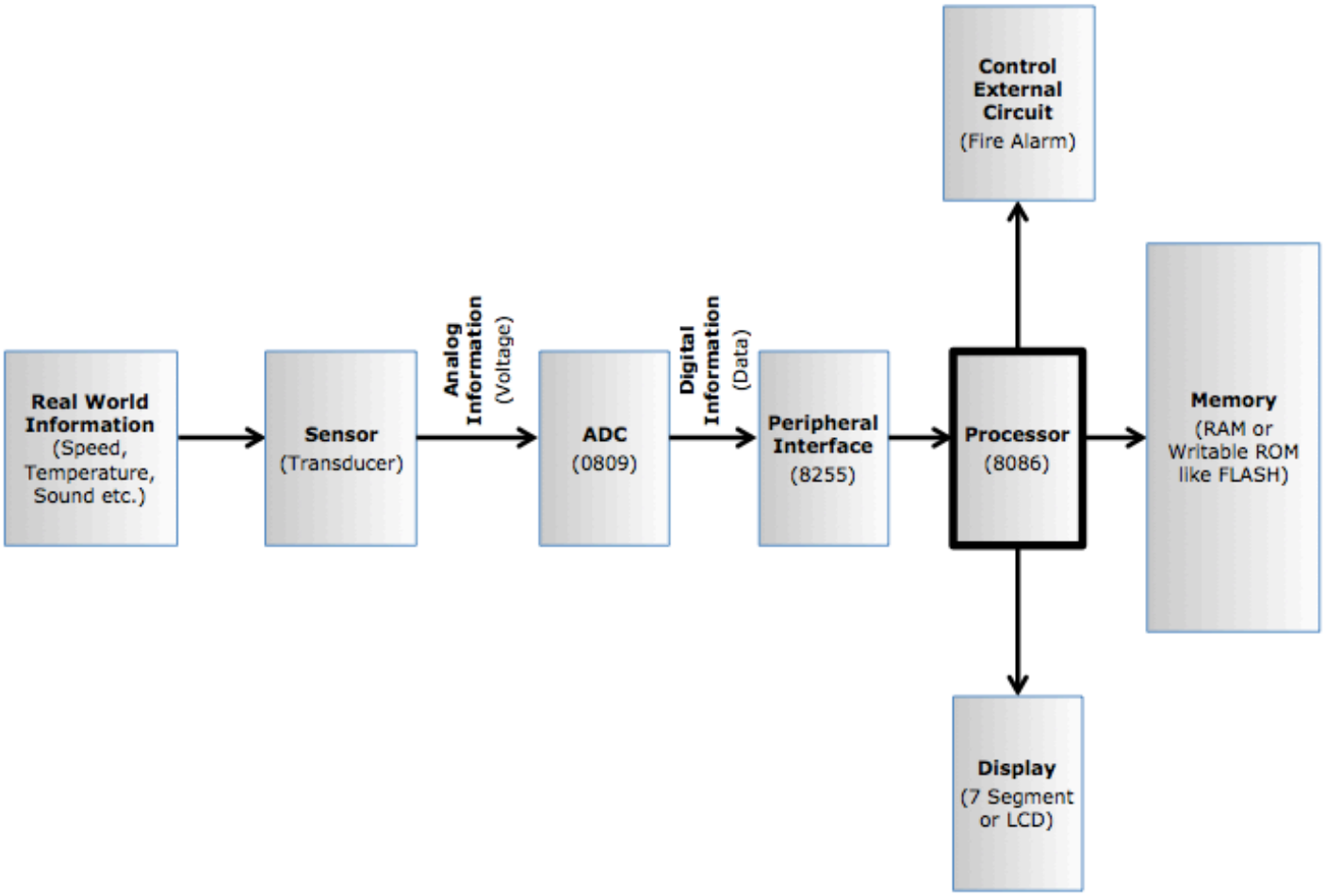

INTERFACE 809 (ADC) to 8086 using 8255.
+Vref -Vref
Vin(0)
Vin(7)
(8 analog inputs)
ADC 0809
clock
8 bit digital output
output enable
end of conversion
Start of Conversion
latch channel
Channel select
8255 PPI
PA
PB
PC
CW
I/O map of 8255 at addr 80H.
A7 A6 A5 A4 A3 A2 A1 A0
PA 10000000 80H
PB 10000010 82H
PC 10000100 84H
C.W. 10000110 86H
8086 up
8282 8bit latch (3)
8286 8bit data T XRVR (2)
74138 3:8 decoder
8284 CLK gen
3x
Reset
Ready

DATA ACQUISITION SYSTEM

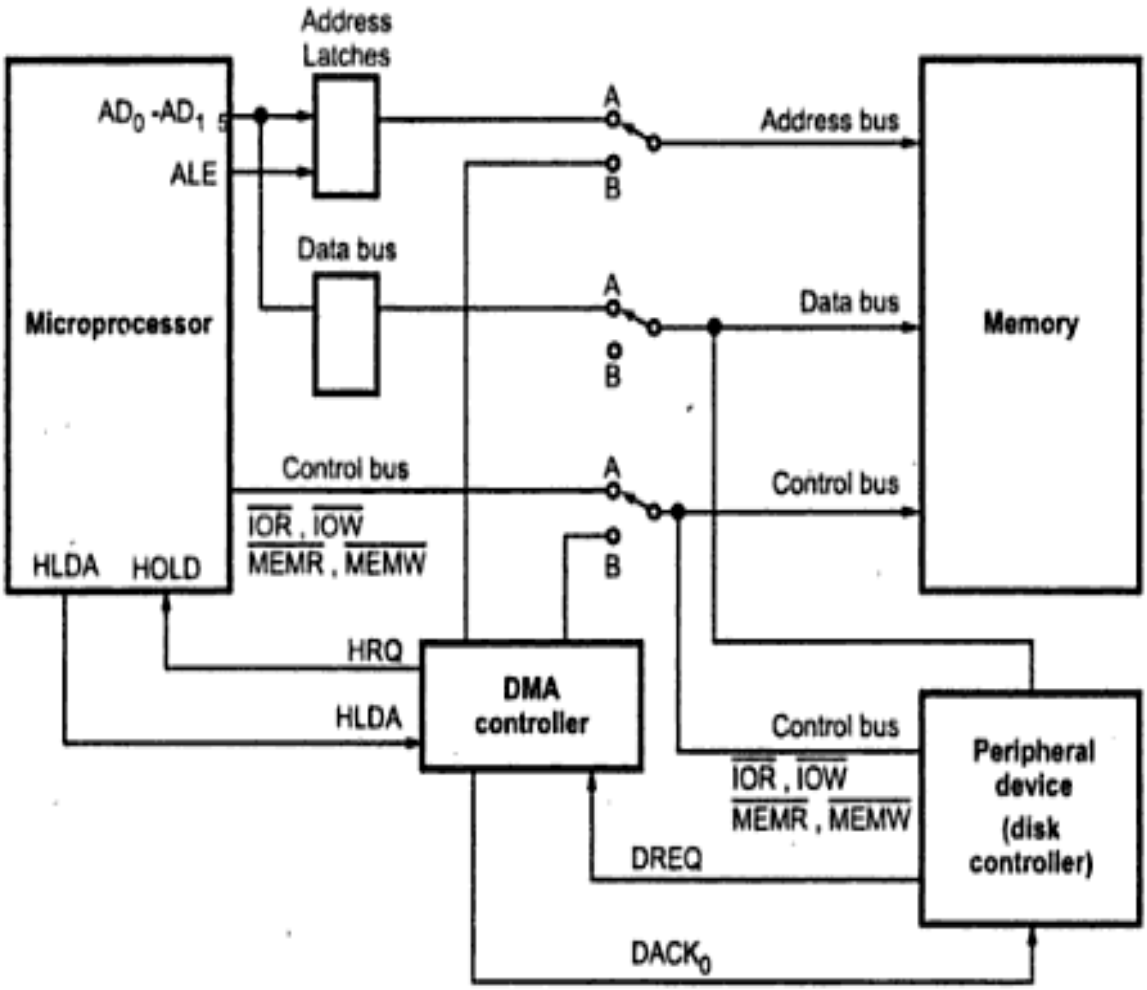
Explain a Data Acquisition System using 8086

1) A data acquisition system is required whenever we need to **obtain real world data such as speed, temperature, sound** etc.
2) Real World data is first **converted into electrical voltage pulses** by a sensor like a transducer, a microphone etc.
3) This is now **Analog information**.
4) This is **fed into an ADC** to convert it into **Digital information**, that's data.

   E.g.:: An **ADC 0809** will convert every Analog sample into 8-bit data.
5) Such data is passed on to a **peripheral interface** device like **8255**.
6) From 8255, it is collected by the **microprocessor 8086**.
7) The ADC could also be connected directly to 8086 but using an 8255 **just makes it easier** as the **port lines of 8255 can control various functions of the ADC.**
8) This data is **stored by the microprocessor into the system memory**.
9) Further on, it **can be processed in various ways**.
10) If it is **Audio**, it can be **stored as an mp3 file**.
11) If it is **temperature or speed** it can be displayed on a **seven segment display**.
12) **Applications: Temperature sensing** in Fire Detection systems**, Speed sensing** in Speed Limiting systems**, Audio recording and playback** (Remember Talking Tomcat app ;-))


Real World Information (Speed, Temperature, Sound etc.)
Sensor (Transducer)
Analog Information (Voltage)
ADC (0809)
Digital Information (Data)
Peripheral Interface (8255)
Processor (8086)
Control External Circuit (Fire Alarm)
Memory (RAM or Writable ROM like FLASH)
Display (7 Segment or LCD)

CONCEPT OF DMA



DMA Transfer is a **hardware controlled I/O** Transfer technique.
It is mainly used for **high-speed data transfer between I/O and Memory** where the speed of the peripheral is generally faster than the µP. **For doubts contact Bharat Sir on 98204 08217**
In Program Controlled I/O, Status or interrupt driven I/O the speed of transfer is **slow** mainly **because instructions** need to be **decoded** and **then executed** for the transfer.
DMA transfer is **software independent** and **hence much faster**.
A device known as the DMA Controller (DMAC) is responsible for the DMA transfer.

The **sequence of DMA transfer** is as follows:

1) µP initializes the DMAC by giving the starting address and the number of bytes to be transferred.
2) An **I/O device requests** the **DMAC**, to perform DMA transfer, **through the DREQ line**.
3) The **DMAC** in turn sends a **request signal to** the **µP**, through the **HOLD line**.
4) The **µP finishes the current machine cycle** and **releases** the **system bus** (gets disconnected from it).
   It also **acknowledges** receiving the HOLD signal through the **HLDA line**.
5) The **DMAC acquires control of the system bus.**
   The **DMAC sends** the **DACK signal** to the I/O peripheral and the **DMA transfer begins**.
6) After every byte is transferred, the Address Reg is incremented (or decremented) and the Count Reg is decremented.
7) This continues till the Count reaches zero (Terminal Count). Now the DMA transfer is completed.
8) **At the end** of the transfer, **the system bus is released by the DMAC** by making HOLD = 0.
   Thus **µP takes control of the system bus** and continues its operation.

The DMA Controller (DMAC) does DMA transfer through its channels.
The minimum requirements of each channel are:

i. **Address Register** (to store the starting address for the transfer).
ii. **Count Register** (to store the number of bytes to be transferred).
iii. **A DREQ signal** from the IO device**.**
iv. **A DACK signal** to the IO device.

Operation Cycles of DMAC

There are mainly two operation cycles of a DMAC.

i. Idle Cycle

After Reset, the DMAC is in idle state (idle cycle).
During idle state, **no DMA operation is taking place**.
No DMA requests are **active**.
The **initialization** of the DMAC takes place in the idle mode.

ii. Active Cycle

Once **DMA operation begins,** the **DMAC** is said to be **in active mode**.
Now the **DMAC controls** the **system bus**.
There are **three types of ACTIVE DMA Cycles** while performing DMA transfer:

1) DMA Read

The DMAC **reads** data **from** the **memory** and **writes into** to the **I/O device**.
Thus, **MEMR** and **IOW** signals are used.

2) DMA Write

The DMAC **reads** data **from** the **I/O device** and **writes into** to the **memory**.
Thus, **IOR** and **MEMW** signals are used.

3) DMA Verify

In this cycle, 8237 does not generate any control signals.
Hence, no data transfer takes place.
During this time, the peripheral and the DMAC verify the correctness of the data transferred, using some error detection method.

Transfer Modes of 8237

8237 has **four modes of data transfer**:

1) Single Byte Transfer Mode/ Cycle Stealing.

Once the DMAC becomes the bus master, it will transfer only **ONE BYTE** and return the bus back to the microprocessor. As soon as the microprocessor performs one bus cycle, DMAC will once again take the bus back from the microprocessor.
Hence both **DMAC and microprocessor** are **constantly stealing bus cycles** from each other.
It is the **most popular** method of DMA, because it keeps the **microprocessor active in the background**.
After a byte is transferred, the **CAR** and **CWCR** are **adjusted** accordingly.
The **system bus is returned** to the **µP.**
For further bytes to be transferred, **the DREQ line must go active again**, and then the entire operation is repeated.

2) Block Transfer Mode.

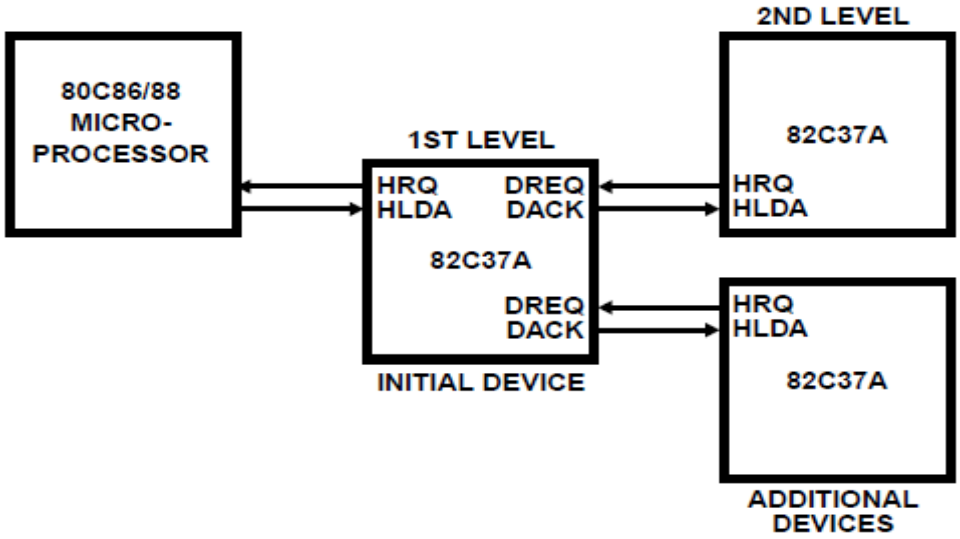
In this mode, the DMAC is programmed to **transfer ALL THE BYTES** in one complete DMA operation.
After a byte is transferred, the **CAR and CWCR are adjusted** accordingly.
The **system bus** is **returned** to the **µP, ONLY after all the bytes are transferred.** I.e. **TC** is **reached or EOP** signal is issued.
It is the **fastest** form of DMA but keeps the **microprocessor inactive** for a long time.
The **DREQ** signal **needs to be active only in the beginning** for requesting the DMA service initially.
Thereafter **DREQ can become low during the transfer**.

3) Demand Transfer Mode.

It is very **similar to Block Transfer**, **except** that the **DREQ must active throughout the DMA operation**.
If during the operation **DREQ goes low**, the **DMA operation is stopped** and the **busses** are **returned** to the **µP.** *#Please refer Bharat Sir's Lecture Notes for this ...*
In the meantime, the **µP** can **continue** with its own operations. **Once DREQ goes high again**, the **DMA operation continues** from where it had stopped.

4) Cascade Transfer Mode. Specific for 8237

In this mode, **more than one DMACs** are cascaded together.
It is used to **increase** the **number of devices interfaced** to the **µP**.
Here we have **one Master DMAC**, to which **one or more Slave DMACs** are connected. The **Slave gives HRQ** to the Master **on the DREQ of the Master**, and the **Master gives HRQ** to the **µP on the HOLD of the µP.**



BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

8085

FULL SYSTEM DESIGNING

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

Important Rules for designing 8085 based systems:

1> **Always map EPROM from location 0000H onwards.** This is the reset vector address of 8085. On reset PC becomes 0000H. Now 8085 executes the "Monitor program" from this address hence we should have permanent (non-volatile) memory at this address.

2> **RAM should start from the next address after EPROM ends**, unless specified otherwise.

3> **I/O chips such as 8255, 8259 etc should be mapped using I/O mapped I/O by default** unless the questions says to use memory mapped I/O.

4> If memory chip number is given instead of chip size then the chip size can be calculated as follows:

1> Chip No: 2764 Chip Size: 27 $\frac{64}{8}$ = 8 KB

2> Chip No: 17128 Chip Size: 17 $\frac{128}{8}$ = 16 KB

3> Chip No: 62256 Chip Size: 62 $\frac{256}{8}$ = 32 KB

5> If the Chip Size is mentioned as (16K x 8) then it means it is a chip of 16K locations and each location has 8-bits making it a 16 KB chip.
Therefore …

1> (16K x 8) Chip means 16 KB.
2> (8K x 8) Chip means 8 KB.
3> (4K x 8) Chip means 4 KB.

6> EPROM is also called Monitor Program Memory or Firmware Memory. RAM is also called Data Storage Memory.

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

Q1) Design an 8085 based system working at 3 MHZ having the following…
16 KB EPROM using 8 KB Chips
32 KB RAM using 16 KB Chips
One 8259 in Memory Mapped I/O.
One 8255 in I/O Mapped I/O. {12/15/20 marks}

Soln: As 8085 is working at 3 MHz, we must connect a crystal of 6 MHz (2 x desired frequency)

Memory Calculations

EPROM:

Required = 16 KB
Available = 8 KB
No of Chips = 2 chips.

Size of a **"Single"** EPROM chip = 8 KB
= 8 x 1 KB
= $2^3 \times 2^{10}$
= 2^{13}
∴ **Each EPROM chip requires 13 address lines (A12 – A0)**

RAM:

Required = 32 KB
Available = 16 KB
No of Chips = 2 chips.

Size of a **"Single"** RAM chip = 16 KB
= 16 x 1 KB
= $2^4 \times 2^{10}$
= 2^{14}
∴ **Each EPROM chip requires 14 address lines (A13 – A0)**

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

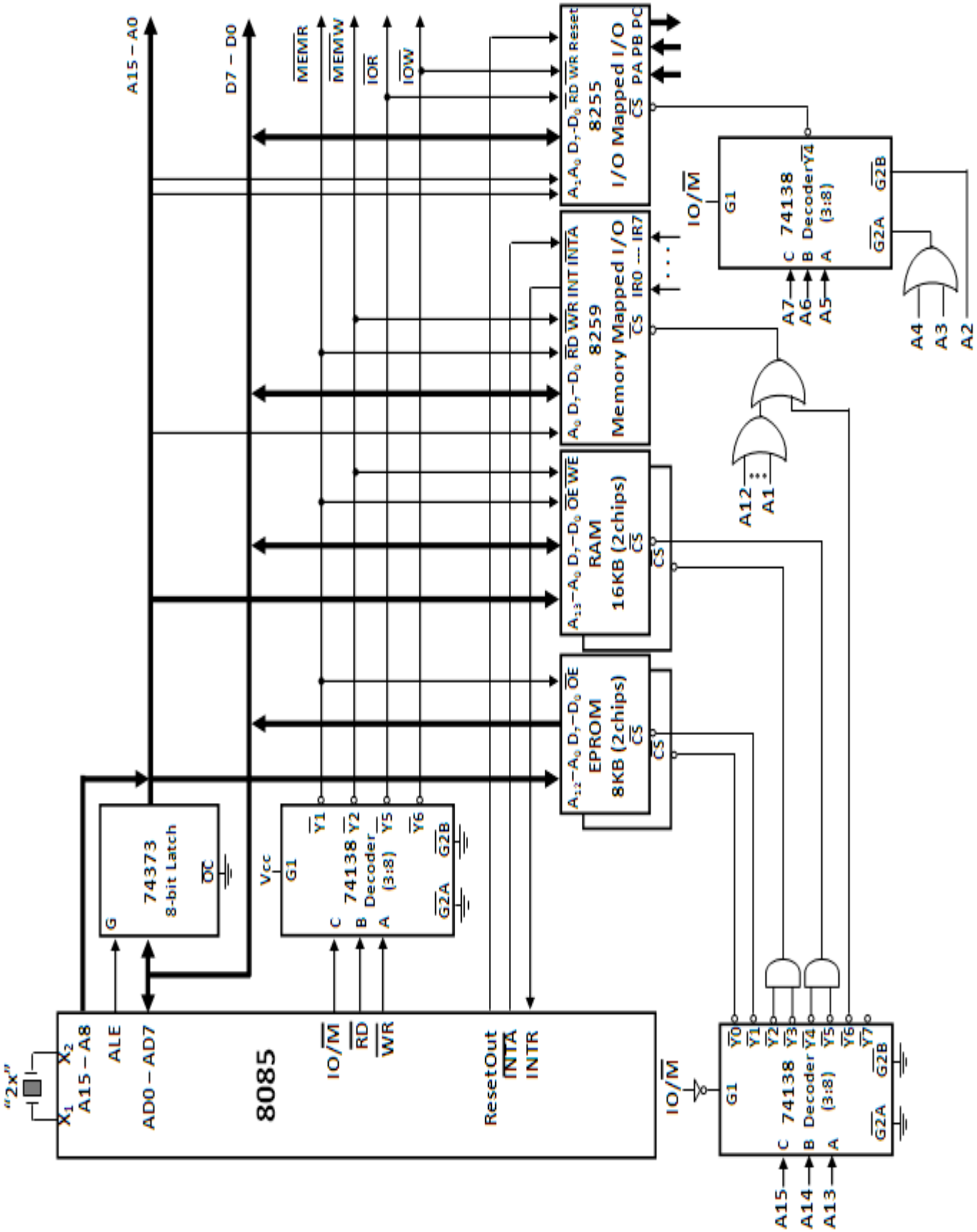
Memory Map

Memory Chip	Address Bus																Memory Address
	A15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	A0	
EPROM1 Begins	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000 H
EPROM1 Ends	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	1FFF H
EPROM2 Begins	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000 H
EPROM2 Ends	0	0	1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	3FFF H
RAM1 Begins	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000 H
RAM1 Ends	0	1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	7FFF H
RAM2 Begins	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000 H
RAM2 Ends	1	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	BFFF H
8259 ICW1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C000 H
8259 ICW2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	C001 H

I/O Map

I/O Chip	Address Bus								I/O port Address
	A7	A6	A5	A4	A3	A2	A1	A0	
8255 PA	1	0	0	0	0	0	**0**	**0**	**80 H**
8255 PB	1	0	0	0	0	0	**0**	**1**	**81 H**
8255 PC	1	0	0	0	0	0	**1**	**0**	**82 H**
8255 CW	1	0	0	0	0	0	**1**	**1**	**83 H**

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com



BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

Q2) Design an 8085 based system working at 3 MHZ having the following…
8 KB EPROM using 4 KB Chips
16 KB RAM using 8 KB Chips

Soln: As 8085 is working at 3 MHz, we must connect a crystal of 6 MHz (2 x desired frequency)

Memory Calculations

EPROM:

Required = 8 KB

Available = 4 KB

No of Chips = 2 chips.

Size of a **"Single"** EPROM chip = 4 KB

$= 4 \times 1 \text{ KB}$

$= 2^2 \times 2^{10}$

$= \mathbf{2^{12}}$

∴ **Each EPROM chip requires 12 address lines (A11 – A0)**

RAM:

Required = 16 KB

Available = 8 KB

No of Chips = 2 chips.

Size of a **"Single"** RAM chip = 8 KB

$= 8 \times 1 \text{ KB}$

$= 2^3 \times 2^{10}$

$= \mathbf{2^{13}}$

∴ **Each RAM chip requires 13 address lines (A12 – A0)**

BHARAT ACHARYA EDUCATION
Cell: 98204 08217
BharatSir@hotmail.com

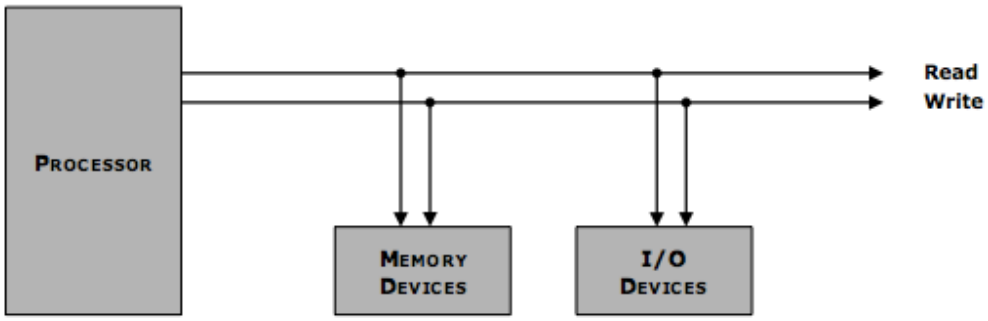
Memory Map

Memory Chip	Address Bus																Memory Address
	A15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	A0	
EPROM1 Begins	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000 H
EPROM1 Ends	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0FFF H
EPROM2 Begins	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1000 H
EPROM2 Ends	0	0	0	1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	1FFF H
RAM1 Begins	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000 H
RAM1 Ends	0	0	1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	3FFF H
RAM2 Begins	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000 H
RAM2 Ends	0	1	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	5FFF H

Solution method 1:

	MEMORY MAPPED I/O	I/O MAPPED I/O
1	**I/O devices are mapped into memory space.**	**I/O devices are mapped into I/O space.**
2	I/O devices are allotted **memory addresses**.	I/O devices are allotted **I/O addresses**.
3	Processor **does not differentiate** between memory and I/O. Treats I/O devices also like memory devices.	Processor **differentiates between I/O devices and memory**. It isolates I/O devices.
4	I/O addresses are as big as memory addresses. E.g.:: in 8085, **I/O addresses will be 16 bit** as memory addresses are also 16-bit.	I/O addresses are smaller than memory addresses. E.g.:: in 8085, **I/O addresses will be 8 bit** though memory addresses are 16-bit.
5	This allows us to increase the number of I/O devices. E.g.:: in 8085, we can access up to **2^{16} = 65536 I/O devices**.	This allows us to access limited number of I/O devices. E.g.:: in 8085, we can access only up to **2^{8} = 256 I/O devices**.
6	We can transfer data from I/O devices using **any instruction like MOV** etc.	We can transfer data from I/O device using **dedicated I/O instructions like IN and OUT ONLY**.
7	Data can be transferred using **any register** of the processor.	Data can be transferred only using a fixed register. E.g.:: in 8085 only **"A" register**.
8	We need **only two control** signals in the system: Read and Write.	We need **four control signals**: Memory Read, Memory Write and I/O Read and I/O Write
9	Memory addresses are big so **address decoding will be slower**.	I/O addresses are smaller so **address decoding will be faster**.
10	Address decoding will be **more complex and costly**.	Address decoding will be **simpler and cheaper**.

MEMORY MAPPED I/O



I/O MAPPED I/O