

8085 Assembly Language Programming

Unit - III

Topics to be covered...

- Instruction Format, Opcodes & Operands
- Addressing Modes
- Machine Language Instruction Format
 - Instruction classification based on Sizes:
 - 1-Byte Instruction
 - 2-Byte Instruction
 - 3-Byte Instruction
 - Instruction classification based on Operation:
 - Data Transfer Instructions
 - Arithmetic Instructions
 - Logical Instructions
 - Branching & Looping Instructions
 - Stack Instructions
 - I/O and Machine Control Instructions
- Classification of Interrupts and Priorities
 - 8085 Vectored Interrupts and RST Instruction
 - 8085 Non-Vectored Interrupt: INTR

Instruction Format, Opcodes & Operands

- To perform any task on microprocessor, set of command needs to be given to 8085 in the form of assembly language program.
- Assembly language is a lower level language.
- Each line of assembly language program contains an instruction.
- Instruction contains 2 parts:
 - Opcode: Task to be completed.
 - Operand(s): Data to be operated upon – Source & Destination.
 - Example: A + B: Here A & B are considered as operands and + is considered as an operation.
- Each processor has its own set of instructions.
- 8085 word size is 8 – bits, So it can only handle 8-bits of data in a single cycle.
- Due to word size we can have 28 (256) instructions.
 - However 8085 uses only 246 combinations that represents total of 74 instructions.

8085 Instruction Sizes

✓ It can be classified into 3 types according to size

- 1 – Byte Instructions
- 2 – Byte Instructions
- 3 – Byte Instructions

✓ 1 – Byte Instructions

- The size of this instruction is 8 – bits.
- Format: <opcode>
- Only opcode which contains all the information.
- Example: **CMA**, **SPHL**, **HLT**

| Mem | Instruction | Opcode | Comment |
|-------|-------------|--------|---------|
| 2050H | CMA | 2FH | Opcode |

8085 Instruction Sizes

✓ 2 – Byte Instructions:

Size of this instruction is 16 – bits.

Format: **<opcode> <data8/add8>**

Opcode specifies only part of whole instruction, another part is stored in second byte.

Example: **MVI A, 42H** , **ADI 07H**

✓ 3 – Byte Instructions:

Size of instruction is 24 – bits.

Format: **<opcode> <data16/add16>**









Opcode contains only part of whole instruction, another part is stored in 2 consecutive locations after the first memory location.

Example: **LDA 5250H** , **STA 2500H**

| Mem | Instruction | Opcode | Comment |
|-------|-------------|--------|--------------|
| 2050H | MVI A, | 3EH | Opcode |
| 2051H | 42H | 42H | Operand/Data |

| Mem | Instruction | Opcode | Comment |
|-------|-------------|--------|-------------|
| 2050H | LDA | 3AH | Opcode |
| 2051H | 50H | 50H | Lower Byte |
| 2052H | 52H | 52H | Higher Byte |

8085 Addressing Modes

-  The method in which the address of source or destination data is provided in instruction is known as Addressing Mode.
-  The way in which the operands are specified.
-  Classification of Instructions of 8085 processor is done in following 5 types:
 -  Implicit Addressing Mode
 -  Immediate Addressing Mode
 -  Register Addressing Mode
 -  Direct Addressing Mode
 -  Register Indirect Addressing Mode

8085 Addressing Modes

Implicit Addressing Mode:

- If address of source and destination data is fixed. So no need to specify any addresses in the instruction.
- Example: **CMA**
- Here CMA is the operation.
- A is the (fixed) source and destination.

Immediate Address Mode:

- Operand is specified directly within the instruction.
- Example: **MVI A, 56H**
- Here MVI is the operation
- 56H is the operand (immediate data).

| | | |
|-----|-------|---------|
| 3EH | 1000H | Opcode |
| 56H | 1001H | Operand |

8085 Addressing Modes

❏ Register Addressing Mode: Register is Source and Destination

- ❏ Operand is stored in one of the General purpose register / Register Pair.
- ❏ Example: **MOV A, B**
- ❏ Here MOV is the Operation
- ❏ B register is the source of data.

❏ Direct Addressing Mode:

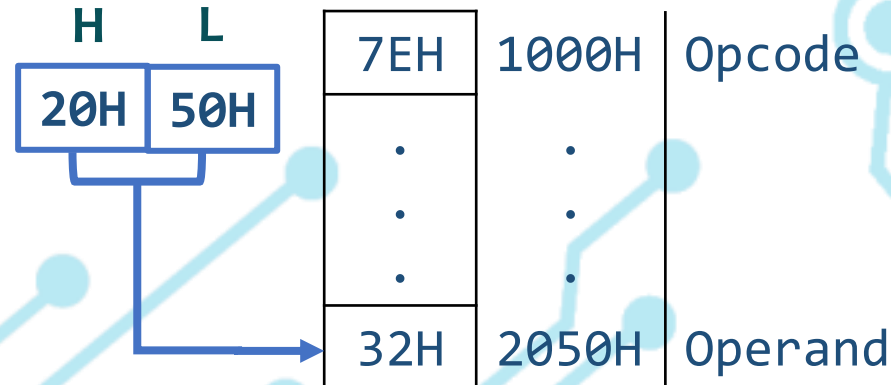
- ❏ Address of the operand is directly given in the instruction.
- ❏ Example: **LDA 2050H**
- ❏ Here LDA is the operation
- ❏ 2050H is the address where the operand is stored.

| | | |
|-----|-------|-------------|
| 3AH | 1000H | Opcode |
| 50H | 1001H | Lower Byte |
| 20H | 1002H | Higher Byte |
| . | . | |
| . | . | |
| . | . | |
| 32H | 2050H | Operand |

8085 Addressing Modes

❏ Register Indirect Addressing Mode:

- ❏ Address of the operand is stored in a register pair specified in the instruction.
- ❏ Example: **MOV A, M**
- ❏ Here MOV is the operation
- ❏ M is memory location defined by HL register pair.



Instruction Classification based on Operation

🖨 As per the function that instruction performs, we can classify 8085 instruction set in following types:

- 🖨 Data Transfer Instructions Eg: MOV, MVI, LDA, STA, LHLD, etc
- 🖨 Arithmetic Instructions Eg: ADD, ADI, SUB, INR, DCR, etc
- 🖨 Logical Instructions Eg: CMA, ANA, ORA, etc
- 🖨 Branching Instructions Eg: JMP, CALL, RET, JZ
- 🖨 Interrupt Instructions Eg: RST, RIM, SIM
- 🖨 Machine Control Instructions Eg: EI, DI, HLT

Data Transfer Instructions

- These instructions move data between registers, between memory and registers or Registers and I/O devices.
- Content is not modified while transferring.
- Data Transfer Instructions can be further Classified into:
 - Move Instructions
 - Load and Store Instructions
 - Load Register Pair
 - Exchange Instructions
 - Stack Instructions
 - PCHL

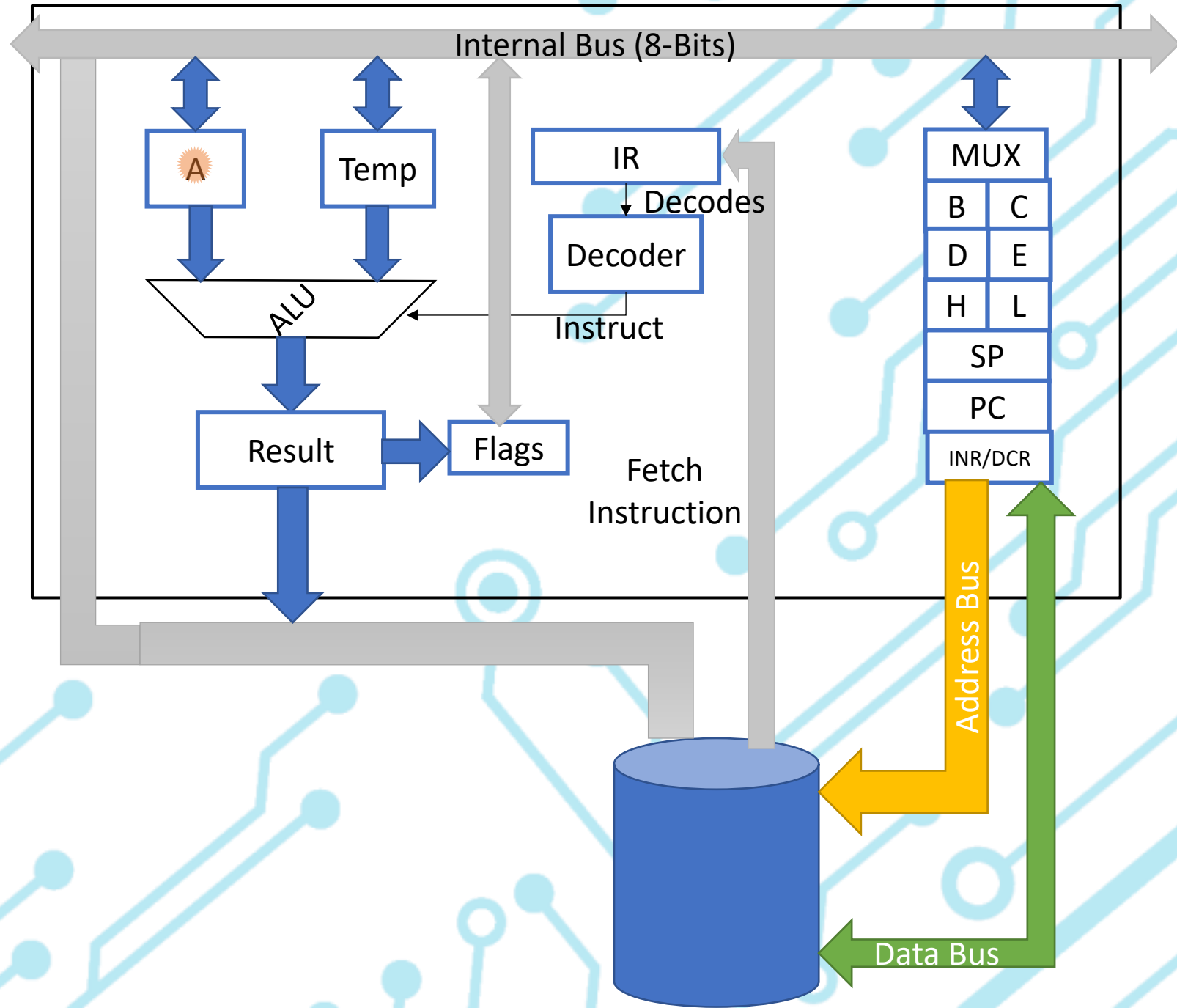
Move Instructions

| Opcode | Operands | Descriptions |
|--------|--------------------------|---|
| MOV | Rd, Rs M, Rs Rd, M | Copies the contents from source to Destination. |

- ❏ This instruction simply copies contents from source to destination.
- ❏ If one of the operands is M then it is location specified by HL register pair.
- ❏ You can only specify 1 memory location at max because 8085 does not support memory-memory data transfer.
- ❏ 1-byte instruction & It does not affect any Flags.
- ❏ Example: **MOV A, D** or **MOV A, M**

Execution

MOV B, A



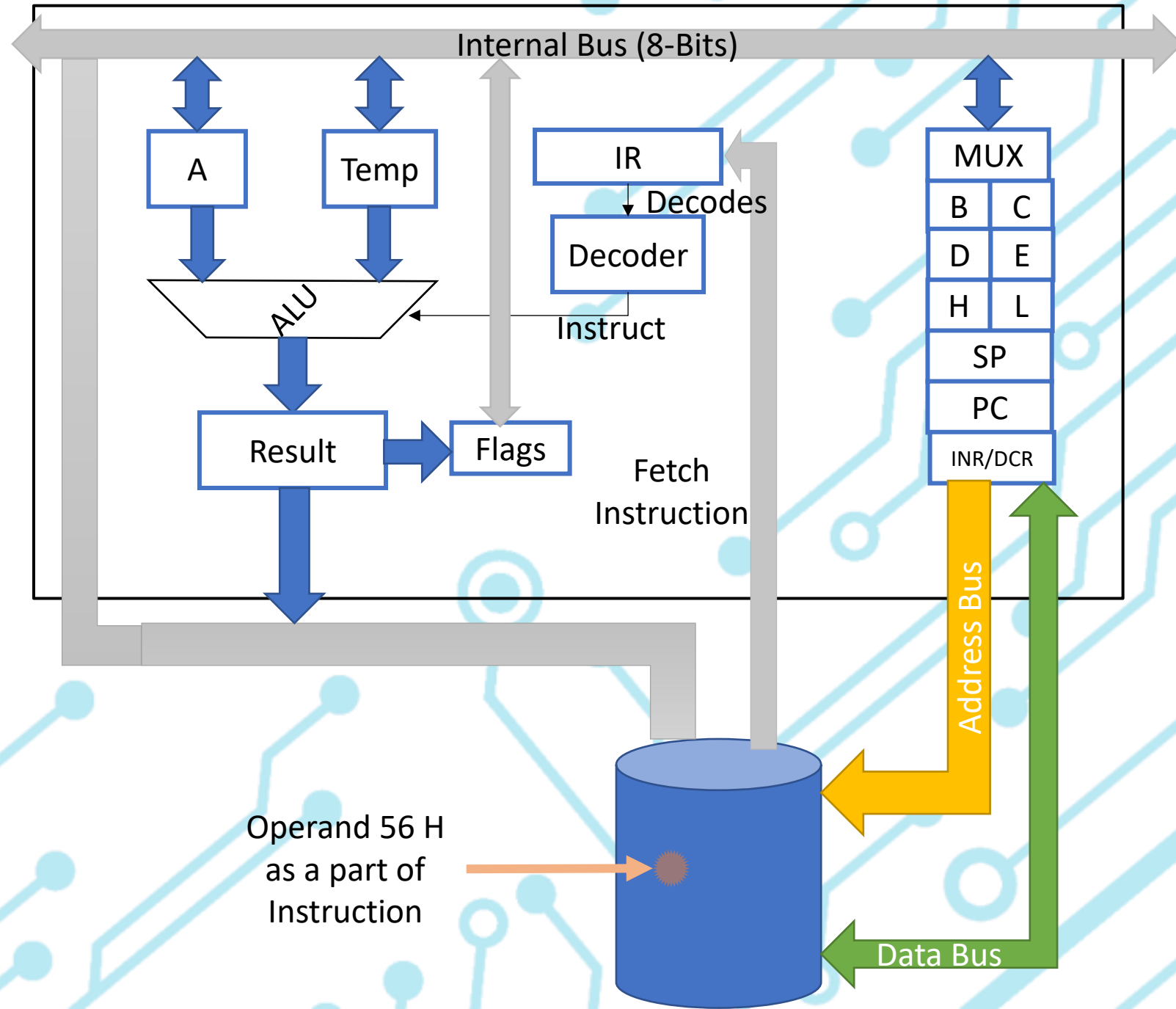
Move Instructions

| Opcode | Operands | Descriptions |
|--------|-----------------------|---|
| MVI | Rd, data8 M, data8 | Copies 8-bit immediate data to given register/memory. |

- ❏ The 8-bit data can be stored in register or memory.
- ❏ If first operand is M then it is location specified by HL register pair.
- ❏ 2-byte instruction & It does not affect any flags.
- ❏ Example: **MVI C, 56H**

Execution

🖨️ MVI C, 56H



Load & Store Instructions

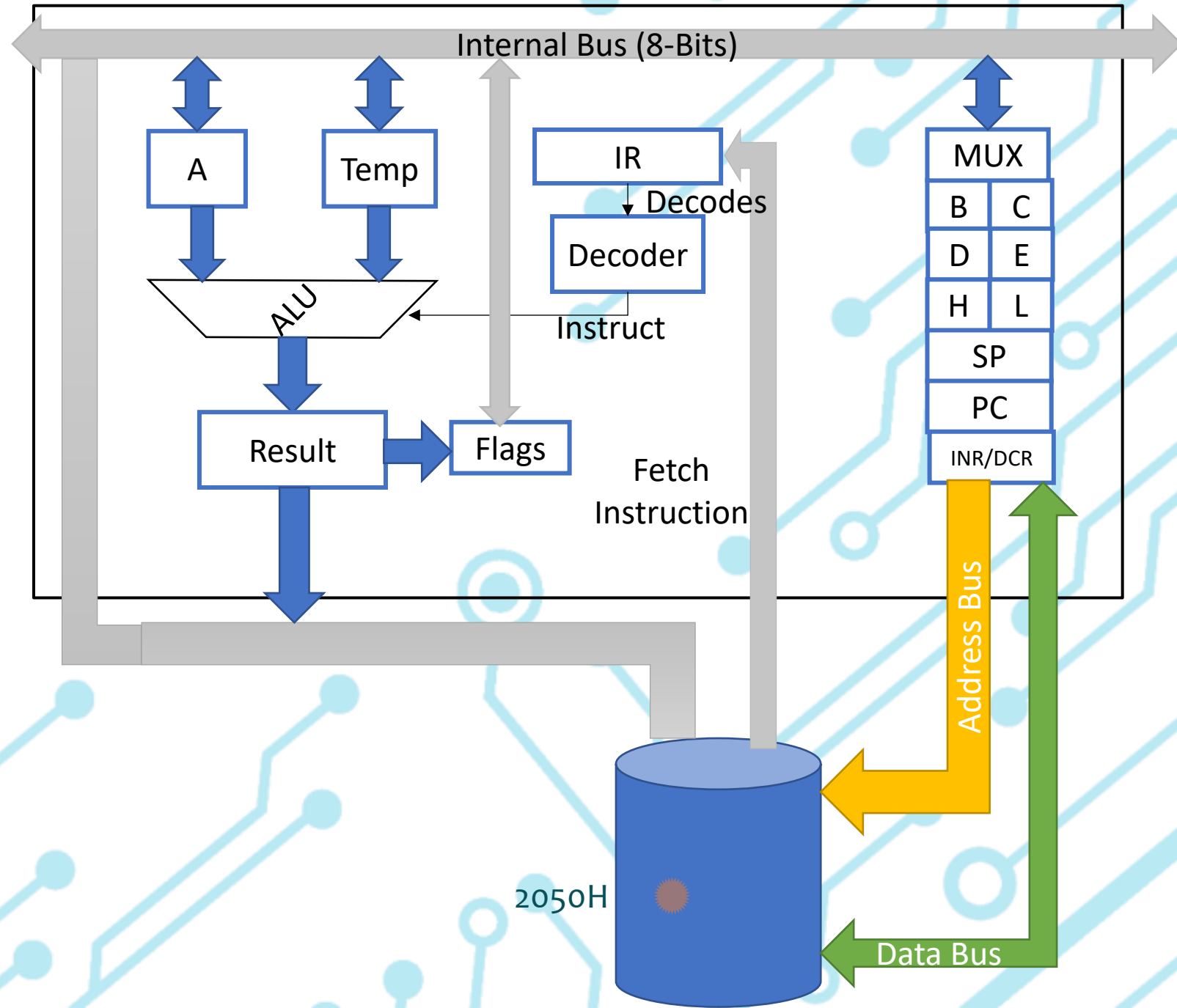
| Opcode | Operands | Descriptions |
|--------|----------|------------------|
| LDA | add16 | Load Accumulator |

- 🖨️ Loads accumulator with the data stored at the address specified within the instruction.
- 🖨️ 3-byte instruction & does not affect any flags.
- 🖨️ Example: **LDA 2050H**

Load data from memory address to register A

Execution

🖨️ LDA 2050H



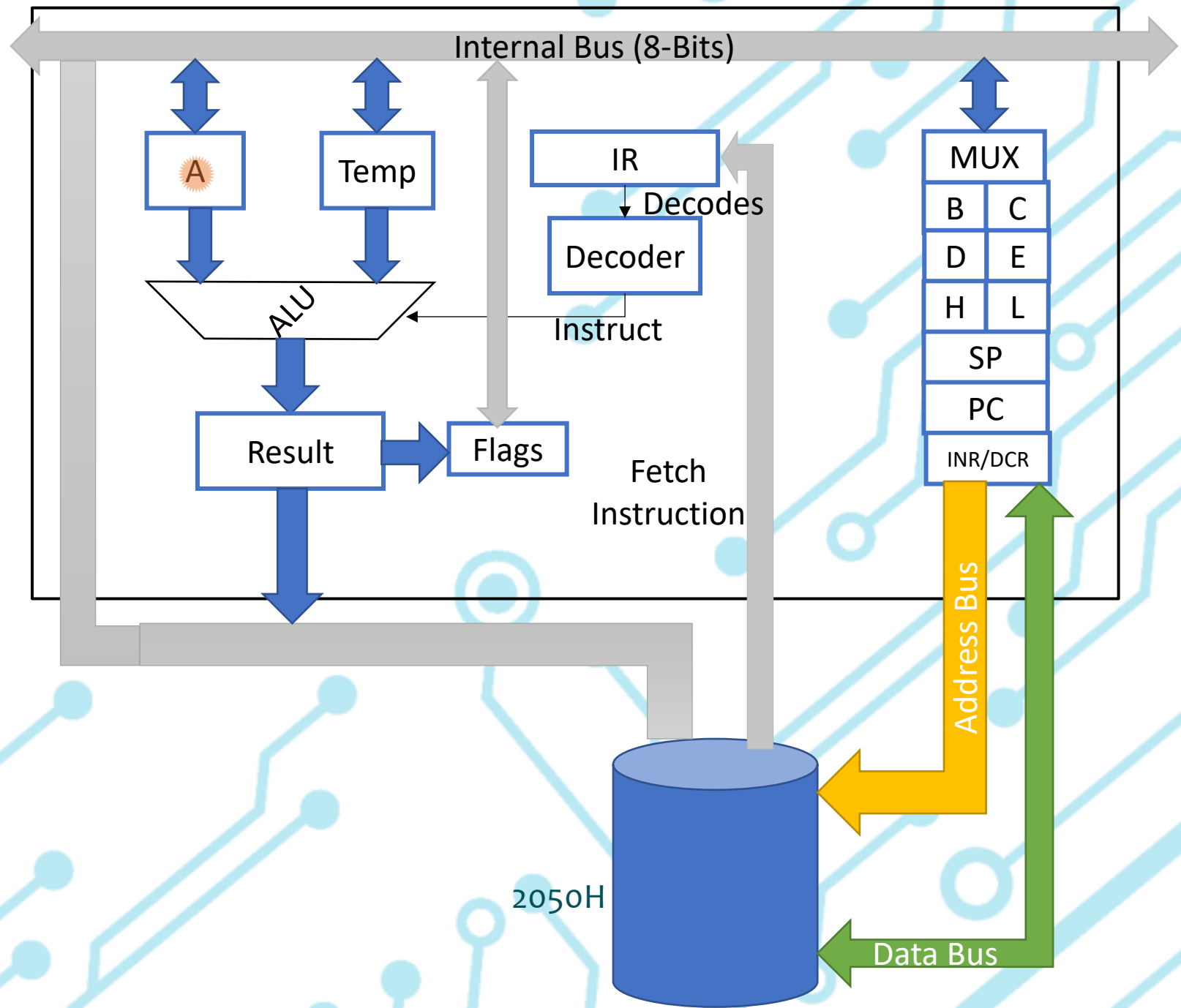
Load & Store Instructions

| Opcode | Operands | Descriptions |
|--------|----------|-------------------|
| STA | add16 | Store Accumulator |

- 🖨 Stores accumulator data at the address specified within the instruction.
- 🖨 3-byte instruction & does not affect any flags.
- 🖨 Example: **STA 2050H**

Load data from register and store it in memory address (A)

STA 2050H



Load & Store Instructions

| Opcode | Operands | Descriptions |
|--------|-------------------|---------------------------|
| LDAX | B/D Register pair | Load Accumulator Indirect |

Data from the address in BC/DE pair is stored in register A

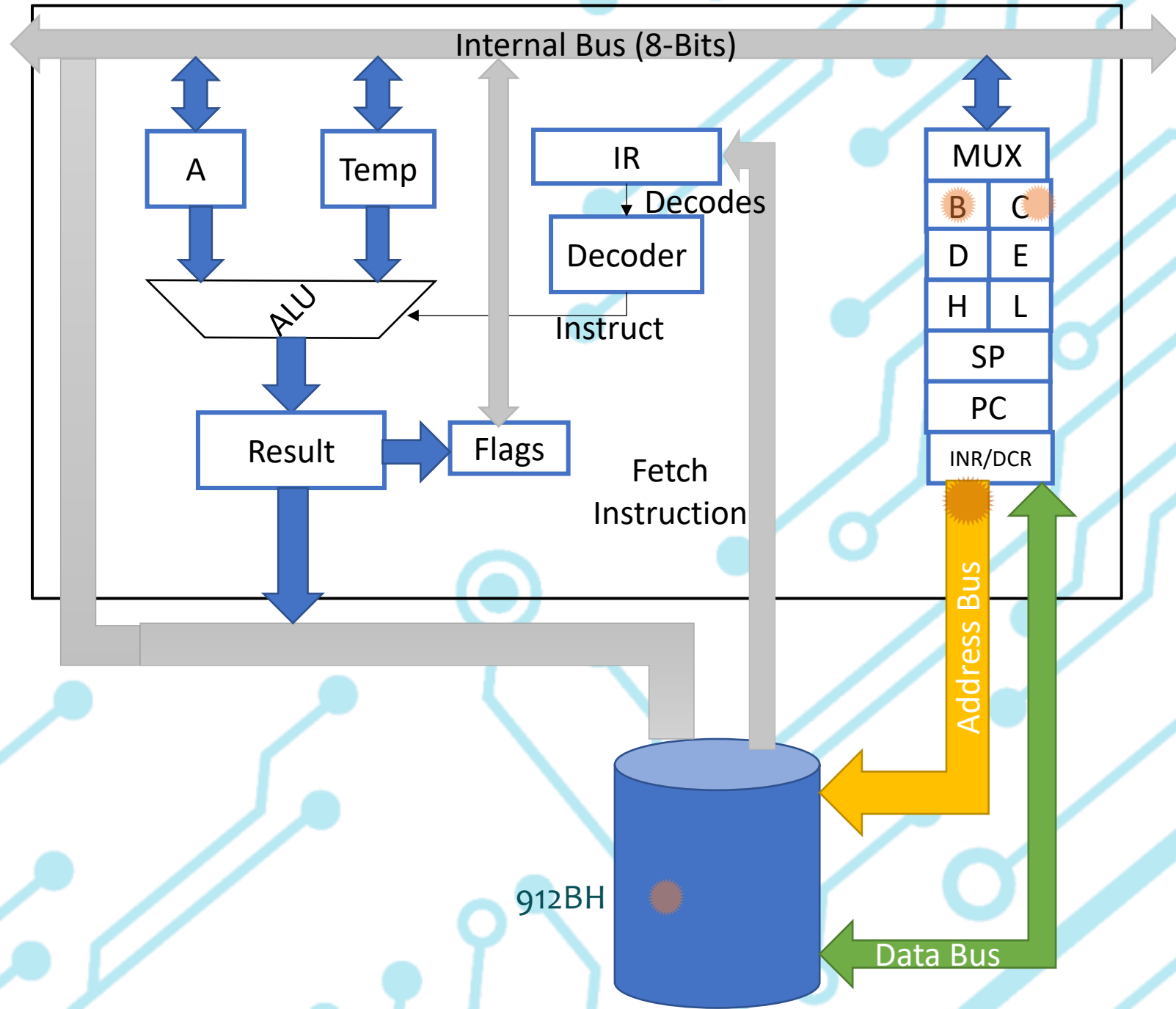
- ❏ Loads accumulator with the data stored at address specified by BC or DE register pair specified within the instruction.
- ❏ First the register pair is accessed to find the address of data and then the data from that address is loaded in accumulator, hence the indirect addressing mode.
- ❏ This instruction does not work with HL register pair.
- ❏ 1-byte instruction & does not affect any flags.
- ❏ Example: **LDAX B**

LDAX B

Suppose Data of

B : 91H

C : 2BH



Load & Store Instructions

| Opcode | Operands | Descriptions |
|--------|-------------------|----------------------------|
| STAX | B/D Register pair | Store Accumulator Indirect |

Data from register A is stored at address in BC/DE pair

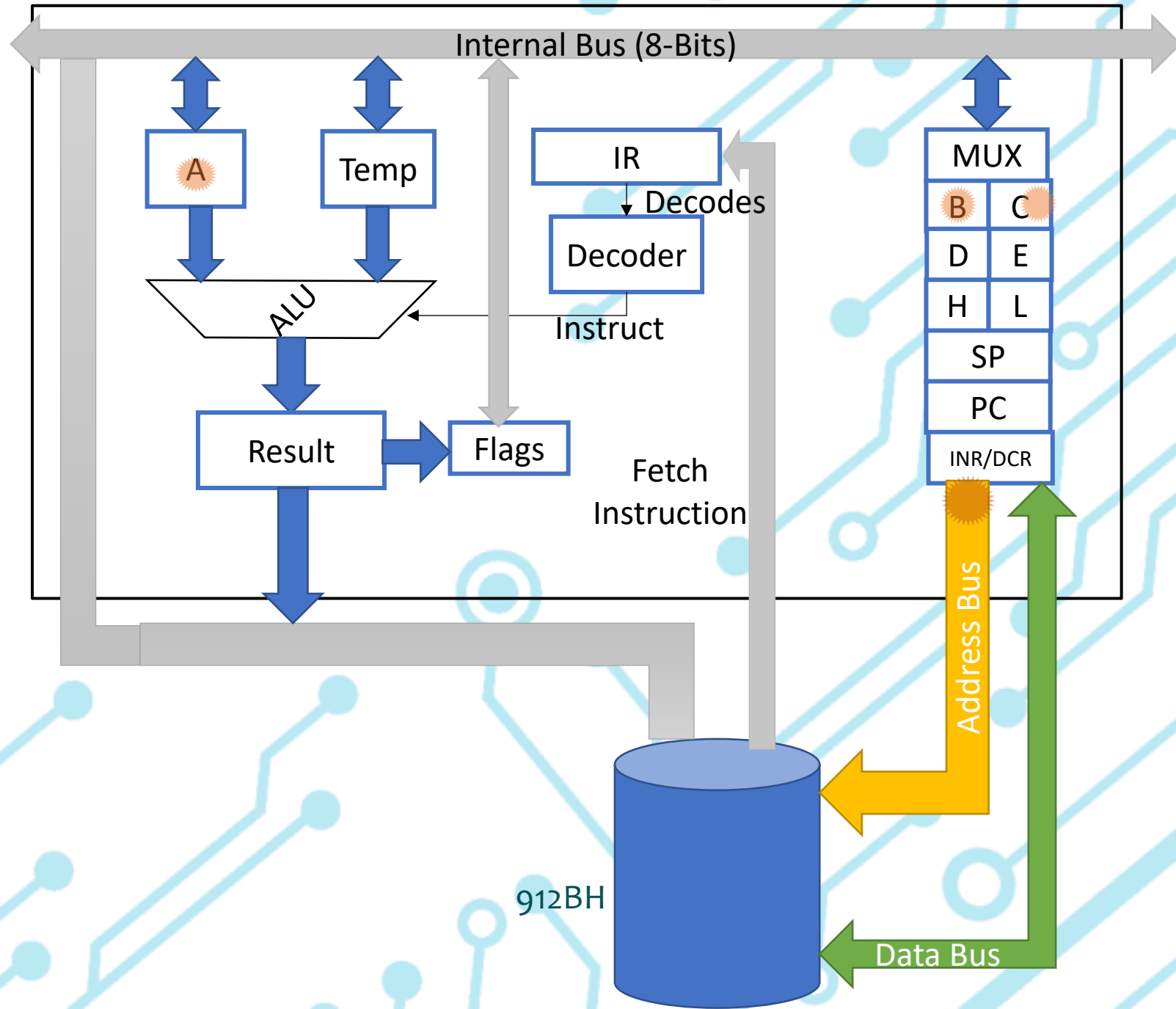
- ❏ Stores accumulator data at address specified by BC or DE register pair specified within the instruction.
- ❏ First the register pair is accessed to find the destination address and then the data from accumulator is stored at that address, hence the indirect addressing mode.
- ❏ This instruction does not work with HL register pair.
- ❏ 1-byte instruction & does not affect any flags.
- ❏ Example: **STAX B**

STAX B

Suppose Data of

B : 91H

C : 2BH



Load & Store Instructions

| Opcode | Operands | Descriptions |
|--------|----------|------------------------------|
| LHLD | add16 | Load HL register pair direct |

- 🖨 Loads HL register pair directly with contents of the memory location specified in the instruction.
- 🖨 3-byte instruction & does not affect any flags.
- 🖨 Example: **LHLD 2050H**
 - 🖨 Loads data of memory location **2050H** into **L** register.
 - 🖨 Loads data of memory location **2051H** into **H** register.

Load & Store Instructions

| Opcode | Operands | Descriptions |
|--------|----------|-------------------------------|
| SHLD | add16 | Store HL register pair Direct |

- ❏ Stores HL register pair directly at the memory location specified in the instruction.
- ❏ 3-byte instruction & does not affect any flags.
- ❏ Example: **SHLD 2050H**
 - ❏ Stores data on memory location **2050H** from **L** register.
 - ❏ Stores data on memory location **2051H** from **H** register.

Load Register Pair

| Opcode | Operands | Descriptions |
|--------|------------|------------------------------|
| LXI | Rp, data16 | Load Register Pair Immediate |

- ❏ Loads the 16-bit immediate data given inside the instruction as byte3 & byte 2 in the register pair given
- ❏ 3-byte instruction & does not affect any flags.
- ❏ Example: **LXI H, 3540H**
 - ❏ Loads data **40H** into **L** register.
 - ❏ Loads data **35H** into **H** register.

Exchange Instructions

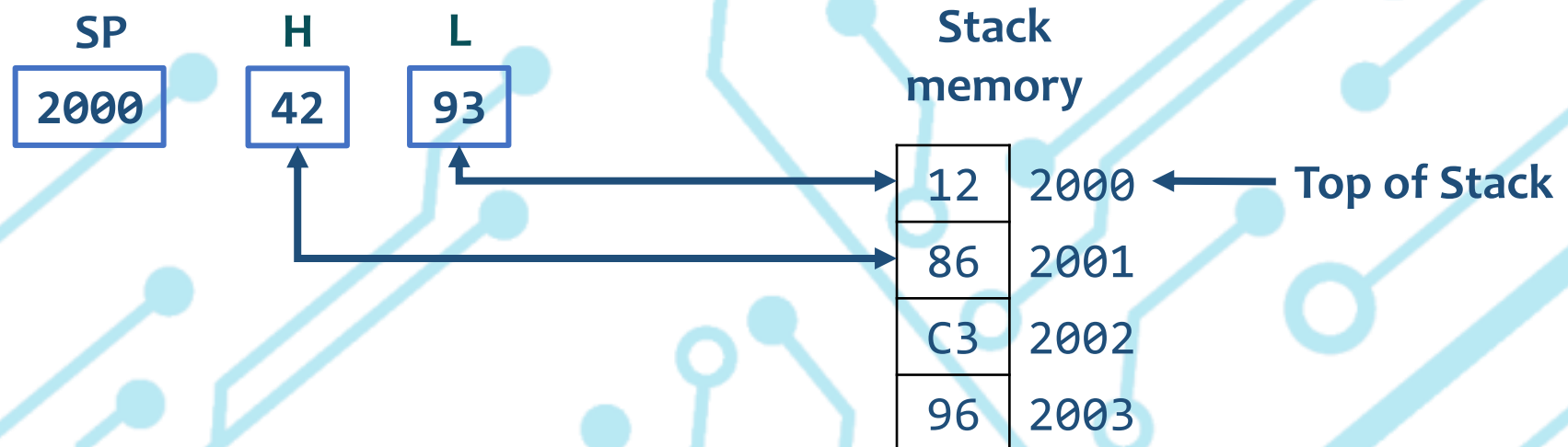
| Opcode | Operands | Descriptions |
|--------|----------|---|
| XCHG | None | Exchange the contents of DE with HL register pair |

- 🖨 Contents of register H is exchanged with contents of register D.
- 🖨 Contents of register L is exchanged with contents of register E.
- 🖨 1-byte instruction & no flags are affected.
- 🖨 Example: **XCHG**

Exchange Instructions

| Opcode | Operands | Descriptions |
|--------|----------|---|
| XTHL | None | Exchange TOS (Top of Stack) with contents of HL Register pair |

- ❏ This instruction exchanges the data between TOS and register L and another exchange of data is done between TOS + 1 & register H.
- ❏ 1-byte instruction & does not affect any flags.
- ❏ Example: **XTHL**



Stack Instructions

| Opcode | Operands | Descriptions |
|--------|----------|---|
| SPHL | None | Copies the contents of HL register pair to SP (Stack Pointer) |

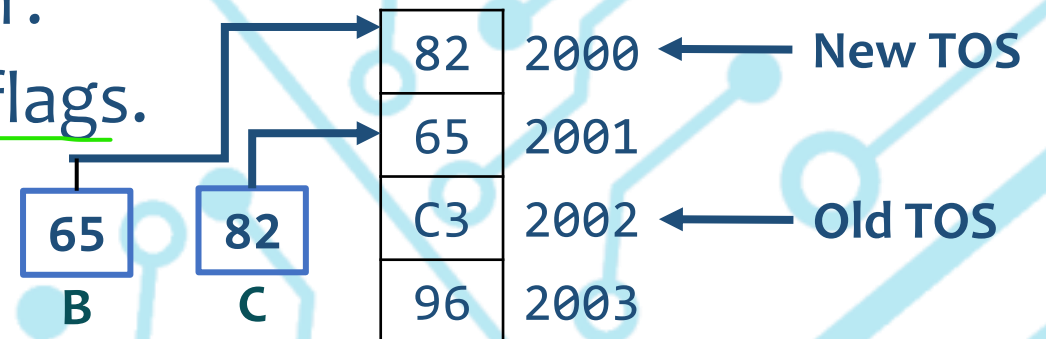
- ❏ This instruction is used to initialise or relocate stack.
- ❏ It copies the contents of HL register pair to SP.
- ❏ 1-byte instruction & does not affect any flags.
- ❏ Example: **SPHL**

| | | | |
|---------------------|------|----|----|
| Before Execution | SP | H | L |
| | 2000 | 42 | 93 |
| After Execution | SP | H | L |
| | 4293 | 42 | 93 |

Stack Instructions

| Opcode | Operands | Descriptions |
|--------|---------------|--|
| PUSH | Register pair | Push 16-bit data of register pair onto stack |

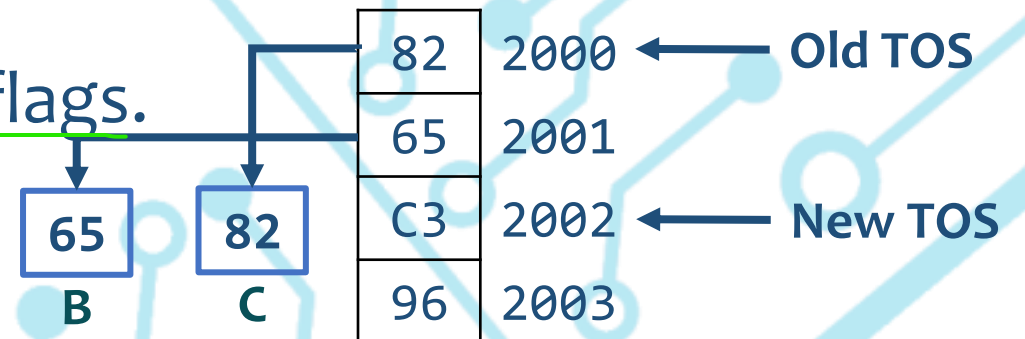
- ❏ Contents of Register pair is copied onto the stack.
- ❏ SP is decremented by 1 and the contents of higher order register (A, B, D, H) is copied into stack.
- ❏ SP is again decremented by 1 and the contents of lower register (Flags, C, E, L) is copied into stack.
- ❏ Here PUSH PSW specifies AF register pair.
- ❏ 1-byte instruction & does not affect any flags.
- ❏ Example: **PUSH B**



Stack Instructions

| Opcode | Operands | Descriptions |
|--------|---------------|--------------------------------|
| POP | Register pair | Pop stack into a register pair |

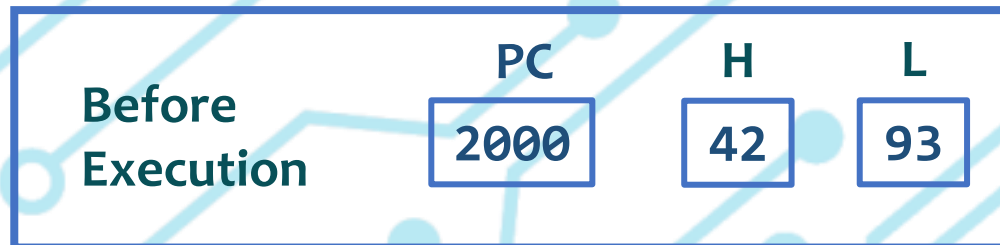
- ❏ Contents of the stack is copied into Register pair.
- ❏ The contents of SP is copied to higher order register (Flags, C, E, L) and SP is incremented by 1.
- ❏ The contents of SP is copied to lower order register (A, B, D, H) and SP is again incremented by 1.
- ❏ Here POP PSW specifies AF register pair.
- ❏ 1-byte instruction & does not affect any flags.
- ❏ Example: **POP B**



PCHL

| Opcode | Operands | Descriptions |
|--------|----------|---|
| PCHL | None | Copy the contents of HL register pair to PC |

- ❏ Copies the data of HL register pair into PC (Program Counter).
- ❏ 1-byte instruction & does not affect any flags.
- ❏ Example: **PCHL**
 - ❏ Execution of this instruction will change the contents of PC performing a jump in the program.



Arithmetic Instructions

- These instructions perform arithmetic operations on data.
- According to architecture of 8085 one operand is fixed which is accumulator and the destination register also will be fixed as accumulator.
- There are some exceptions in which other registers can be used like increment/decrement instructions.
- Arithmetic Instructions are further classified into:
 - Add Instructions
 - Subtract Instructions
 - Increment/Decrement Instructions
 - Decimal Adjust Accumulator

Add instructions

| Opcode | Operands | Descriptions |
|--------|----------|---|
| ADD | R M | Add register or memory with Accumulator |

- The contents of register or the memory location specified are added with the contents of Accumulator.
- The result of addition is stored back into the accumulator and residual carry is stored in CY flag
- 1-byte instruction & It affects all Flags.
- Example: **ADD C**
 - Equivalent to $[A] \leftarrow [A] + [C]$

Add Instructions

| Opcode | Operands | Descriptions |
|--------|----------|--|
| ADI | data8 | Add 8-bit of data with the contents of Accumulator |

- Immediate 8-bit data provided in next byte of instruction is added with the contents of accumulator.
- The result of addition is stored back into the accumulator and residual carry is stored in CY flag
- 2-byte instruction & It affects all Flags.
- Example: **ADI 56H**
 - Equivalent to $[A] \leftarrow [A] + 56H$

Add Instructions

| Opcode | Operands | Descriptions |
|--------|----------|--|
| ADC | R M | Add register or memory with the contents of Accumulator with carry flag. |

- ❏ The content of accumulator is added with the contents of register or memory specified and also CY flag is added.
- ❏ The result of addition is stored back into the accumulator and residual carry is stored in CY flag
- ❏ 1-byte instruction & It affects all Flags.
- ❏ Example: **ADC M**
 - ❏ Equivalent to $[A] \leftarrow [A] + [[H][L]] + [CY]$

Add Instructions

| Opcode | Operands | Descriptions |
|--------|----------|--|
| ACI | data8 | Adds 8-bit data with the contents of Accumulator and carry flag. |

- Immediate 8-bit data provided in next byte of instruction is added with the contents of accumulator and CY flag.
- The result of addition is stored back into the accumulator and residual carry is stored in CY flag
- 2-byte instruction & It affects all Flags.
- Example: **ACI 56H**
 - Equivalent to $[A] \leftarrow [A] + 56H + [CY]$

Add Instructions

| Opcode | Operands | Descriptions |
|--------|----------|--|
| ACI | data8 | Adds 8-bit data with the contents of Accumulator and carry flag. |

- Immediate 8-bit data provided in next byte of instruction is added with the contents of accumulator and CY flag.
- The result of addition is stored back into the accumulator and residual carry is stored in CY flag
- 2-byte instruction & It affects all Flags.
- Example: **ACI 56H**
 - Equivalent to $[A] \leftarrow [A] + 56H + [CY]$

Add Instructions

| Opcode | Operands | Descriptions |
|--------|----------|---|
| DAD | Rp | Adds HL register pair with the given register pair. |

- It performs 16-bit addition of the contents stored in HL register pair with the contents of given register pair.
- The result of addition is stored back into the HL Pair and residual carry is stored in CY flag
- 1-byte instruction & It affects CY flag only.
- Example: **DAD B**
 - Equivalent to $[H][L] \leftarrow [H][L] + [Rh][RI]$

Subtract Instructions

| Opcode | Operands | Descriptions |
|--------|----------|---|
| SUB | R M | Subtracts register or memory from the Accumulator |

- It subtracts the content of register or memory location specified from the contents of Accumulator.
- The result of subtraction is stored back into the Accumulator and residual borrow is held by CY flag.
- 1-byte instruction & It affects all flags.
- Example: **SUB C**
 - Equivalent to $[A] \leftarrow [A] - [C]$

Subtract Instructions

| Opcode | Operands | Descriptions |
|--------|----------|---|
| SUI | data8 | Subtract 8-bit of data from the contents of Accumulator |

- Immediate 8-bit data provided in next byte of instruction is subtracted from the contents of accumulator.
- The result of subtraction is stored back into the accumulator and residual borrow is held by CY flag.
- 2-byte instruction & It affects all Flags.
- Example: **SUI 56H**
 - Equivalent to $[A] \leftarrow [A] - 56H$

Subtract Instructions

| Opcode | Operands | Descriptions |
|--------|----------|---|
| SBB | R M | Subtract register or memory from the contents of Accumulator with carry flag. |

- ❏ The content of accumulator is subtracted with the contents of register or memory specified and also CY flag is added.
- ❏ The result of subtraction is stored back into the accumulator and residual borrow is held by CY flag.
- ❏ 1-byte instruction & It affects all Flags.
- ❏ Example: **SBB M**
 - ❏ Equivalent to $[A] \leftarrow [A] - [[H][L]] - [CY]$

Subtract Instructions

| Opcode | Operands | Descriptions |
|--------|----------|--|
| SBI | data8 | Subtract 8-bit data and carry flag from the contents of Accumulator. |

- Immediate 8-bit data provided in next byte of instruction and carry flag is subtracted with the contents of accumulator.
- The result of subtraction is stored back into the accumulator and residual borrow is held by CY flag.
- 2-byte instruction & It affects all Flags.
- Example: **SBI 56H**
 - Equivalent to $[A] \leftarrow [A] - 56H - [CY]$

Increment/Decrement Instructions

| Opcode | Operands | Descriptions |
|--------|----------|---|
| INR | R M | Increments the register or memory by 1. |

- 🖨 Increments the register or memory specified by 1.
- 🖨 It stores the result back to the location specified as source.
- 🖨 1-byte instruction & It affects all flags except CY.
- 🖨 Example: **INR B**
 - 🖨 Equivalent to $[B] \leftarrow [B] + 1$

Increment/Decrement Instructions

| Opcode | Operands | Descriptions |
|--------|----------|---|
| DCR | R M | Decrements the register or memory by 1. |

- ❏ Decrements the register or memory specified by 1.
- ❏ It stores the result back to the location specified as source.
- ❏ 1-byte instruction & It affects all flags except CY.
- ❏ Example: **DCR M**
 - ❏ Equivalent to $[[H][L]] \leftarrow [[H][L]] - 1$

Increment/Decrement Instructions

| Opcode | Operands | Descriptions |
|--------|----------|------------------------------------|
| INX | Rp | Increments the register pair by 1. |

- 🖨 Increments the register pair (16-bits) by 1.
- 🖨 It stores the result back to the location specified as source.
- 🖨 1-byte instruction & It does not affect any flags.
- 🖨 Example: **INX B**
 - 🖨 Equivalent to $[B][C] \leftarrow [B][C] + 1$

Increment/Decrement Instructions

| Opcode | Operands | Descriptions |
|--------|----------|------------------------------------|
| DCX | Rp | Decrements the register pair by 1. |

- ❏ Decrements the register pair (16-bits) by 1.
- ❏ It stores the result back to the location specified as source.
- ❏ 1-byte instruction & It does not affect any flags.
- ❏ Example: **DCX B**
 - ❏ Equivalent to $[B][C] \leftarrow [B][C] - 1$

Decimal Adjust Accumulator

| Opcode | Operands | Descriptions |
|--------|----------|--|
| DAA | None | Adjust value of Accumulator for BCD operations |

■ If we save 2 BCD data and Add them using one of the Addition instruction then it performs binary addition so result in accumulator might contain some invalid character for BCD. DAA will simple add 6 to that invalid digit and make it a legit BCD number (i.e. between 0 to 9).

■ 1-byte instruction & It affects all flags.

■ Example:

■ If BCD nos 56 and 16 is added then AC will contain 6C. If we write DAA after addition instruction then it will add 6 to invalid character C

$$\begin{array}{r} 1 \leftarrow \text{Auxilliary Carry} \\ 6 \quad C \\ + 0 \quad 6 \\ \hline 7 \quad 2 \leftarrow \text{BCD Answer of } 56+16 \end{array}$$

Logical Instructions

- These instructions perform logical operations on data.
- According to architecture of 8085 one operand is fixed which is accumulator and the destination register also will be fixed as accumulator.
- There are some exceptions in which other registers can be used like some complement or compare instructions.
- Logical Instructions can be further classified into:
 - AND Instructions
 - OR Instructions
 - EX-OR Instructions
 - Set/Complement Instructions
 - Compare Instructions
 - Rotate Instructions

AND instructions

| Opcode | Operands | Descriptions |
|--------|----------|---|
| ANA | R M | Performs logical AND between Accumulator and register or memory |

- Logical AND/Bitwise AND operation is performed between contents of Accumulator and contents of register or the memory location specified.
- The result of AND is stored back into the accumulator.
- 1-byte instruction & It affects all Flags.
- Example: **ANA B**
 - Equivalent to $[A] \leftarrow [A] \wedge [B]$

AND instructions

| Opcode | Operands | Descriptions |
|--------|----------|---|
| ANI | data8 | Performs logical AND between Accumulator and 8-bit data specified |

- ❏ Logical AND/Bitwise AND operation is performed between contents of Accumulator and contents specified in the next byte of instruction as immediate data.
- ❏ The result of AND is stored back into the accumulator.
- ❏ 2-byte instruction & It affects all Flags.
- ❏ Example: **ANI 32H**
 - ❏ Equivalent to $[A] \leftarrow [A] \wedge 32H$

OR instructions

| Opcode | Operands | Descriptions |
|--------|----------|--|
| ORA | R M | Performs logical OR between Accumulator and register or memory |

- ❏ Logical OR/Bitwise OR operation is performed between contents of Accumulator and contents of register or the memory location specified.
- ❏ The result of OR is stored back into the accumulator.
- ❏ 1-byte instruction & It affects all Flags.
- ❏ Example: **ORA B**
 - ❏ Equivalent to $[A] \leftarrow [A] \vee [B]$

OR instructions

| Opcode | Operands | Descriptions |
|--------|----------|--|
| ORI | data8 | Performs logical OR between Accumulator and 8-bit data specified |

- ❏ Logical OR/Bitwise OR operation is performed between contents of Accumulator and contents specified in the next byte of instruction as immediate data.
- ❏ The result of OR is stored back into the accumulator.
- ❏ 2-byte instruction & It affects all Flags.
- ❏ Example: **ORI 32H**
 - ❏ Equivalent to $[A] \leftarrow [A] \vee 32H$

EX-OR instructions

| Opcode | Operands | Descriptions |
|--------|----------|---|
| XRA | R M | Performs logical Ex-OR between Accumulator and register or memory |

- ❏ Logical Ex-OR/Bitwise Ex-OR operation is performed between contents of Accumulator and contents of register or the memory location specified.
- ❏ The result of Ex-OR is stored back into the accumulator.
- ❏ 1-byte instruction & It affects all Flags.
- ❏ Example: **XRA B**
 - ❏ Equivalent to $[A] \leftarrow [A] \vee [B]$

Ex-OR instructions

| Opcode | Operands | Descriptions |
|--------|----------|---|
| XRI | data8 | Performs logical Ex-OR between Accumulator and 8-bit data specified |

- ❏ Logical Ex-OR/Bitwise Ex-OR operation is performed between contents of Accumulator and contents specified in the next byte of instruction as immediate data.
- ❏ The result of Ex-OR is stored back into the accumulator.
- ❏ 2-byte instruction & It affects all Flags.
- ❏ Example: **XRI 32H**
 - ❏ Equivalent to $[A] \leftarrow [A] \vee 32H$

Set/Complement Instructions

| Opcode | Operands | Descriptions |
|--------|----------|---|
| CMA | None | Complements the contents of Accumulator |

- ❏ Performs complement/Logical NOT operation on contents of Accumulator.
- ❏ Result of complement is stored back into the accumulator.
- ❏ 1-byte Instruction & It does not affect any flags.
- ❏ Example: **CMA**
 - ❏ Equivalent to $[A] \leftarrow \neg[A]$

Set/Complement Instructions

| Opcode | Operands | Descriptions |
|--------|----------|--------------------------------|
| CMC | None | Complements the contents of CY |

- ❏ Performs complement/Logical NOT operation on contents of Carry Flag.
- ❏ Result of complement is stored back into the CY Flag.
- ❏ 1-byte Instruction & It affects CY flag.
- ❏ Example: **CMC**
 - ❏ Equivalent to `[CY] <- ![CY]`

Set/Complement Instructions

| Opcode | Operands | Descriptions |
|--------|----------|------------------------------|
| STC | None | Sets the contents of CY to 1 |

- 🖨️ Set the value to Carry flag to 1 irrespective of its old value.
- 🖨️ Result of complement is stored back into the CY Flag.
- 🖨️ 1-byte Instruction & It affects CY flag.
- 🖨️ Example: **STC**
 - 🖨️ Equivalent to $[CY] \leftarrow 1$

Compare Instructions

| Opcode | Operands | Descriptions |
|--------|----------|---|
| CMP | R M | Compare the contents of accumulator with register or given memory |

- ❏ Performs comparison between contents of Accumulator and contents of specified register or memory. The output is calculated by subtracting register or memory from the Accumulator temporarily, the answer is not stored anywhere.
- ❏ 1-byte instruction & It affects all the flags.
- ❏ The answer is stored using flags in following manner.

| Condition | Operation | CY Flag | Z Flag |
|-----------|-------------|---------|--------|
| $A > B$ | $A - B > 0$ | 0 | 0 |
| $A = B$ | $A - B = 0$ | 0 | 1 |
| $A < B$ | $A - B < 0$ | 1 | 0 |

Compare Instructions

| Opcode | Operands | Descriptions |
|--------|----------|---|
| CPI | data8 | Compare the contents of accumulator with 8-bit immediate data |

- ❏ Performs comparison between contents of Accumulator and specified immediate 8-bit data. The output is calculated by subtracting register or memory from the Accumulator temporarily, the answer is not stored anywhere.
- ❏ 1-byte instruction & It affects all the flags.
- ❏ The answer is stored using flags in following manner.

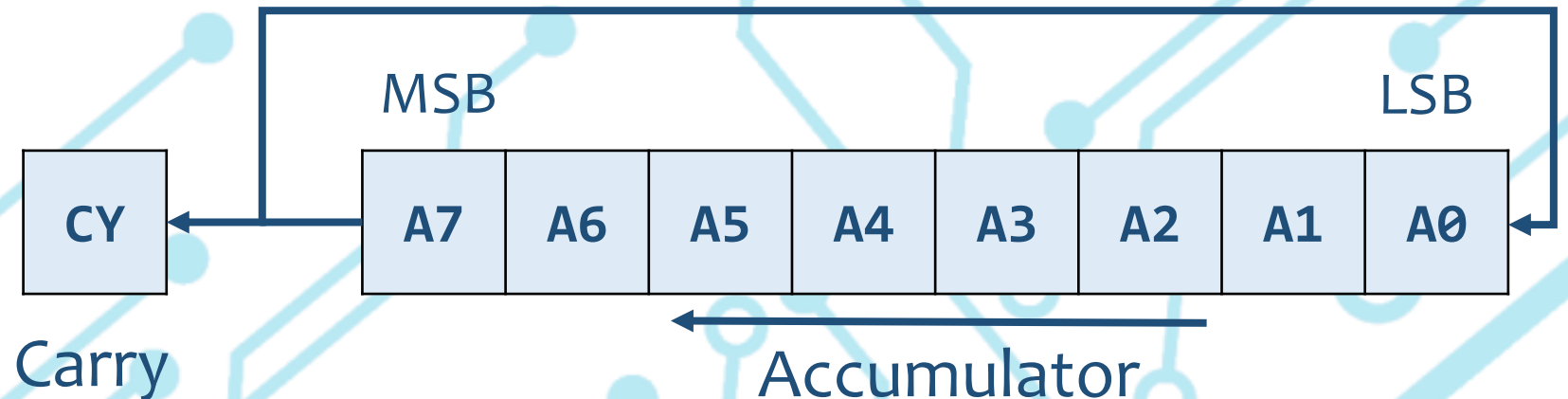
| Condition | Operation | CY Flag | Z Flag |
|--------------------|------------------------|---------|--------|
| $A > \text{data8}$ | $A - \text{data8} > 0$ | 0 | 0 |
| $A = \text{data8}$ | $A - \text{data8} = 0$ | 0 | 1 |
| $A < \text{data8}$ | $A - \text{data8} < 0$ | 1 | 0 |

Rotate Instructions

| Opcode | Operands | Descriptions |
|--------|----------|-------------------------|
| RLC | None | Rotate Accumulator Left |

- Rotates the data of Accumulator to left by 1 position.
- New rotated data is stored back in accumulator.
- 1-byte instruction & It affects CY flag.
- Example: **RLC**

- Equivalent to,
- $[A_{n+1}] \leftarrow [A_n]$,
- $[A_0] \leftarrow [A_7]$,
- $[CY] \leftarrow [A_7]$



Rotate Instructions

| Opcode | Operands | Descriptions |
|--------|----------|--------------------------|
| RRC | None | Rotate Accumulator Right |

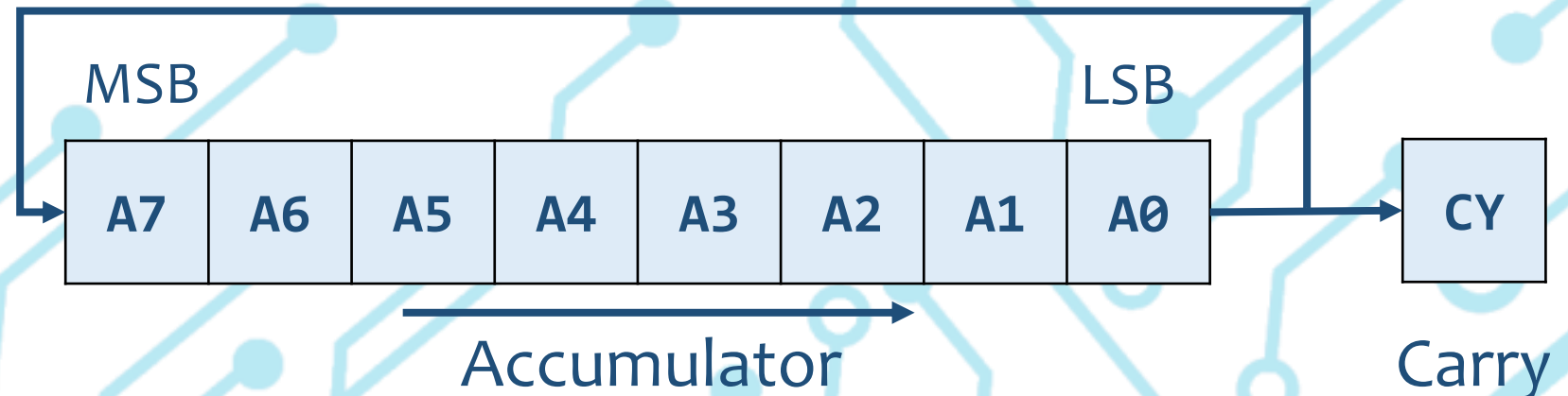
- Rotates the data of Accumulator to right by 1 position.
- New rotated data is stored back in accumulator.
- 1-byte instruction & It affects CY flag.
- Example: **RRC**

- Equivalent to,

- $[A_n] \leftarrow [A_{n+1}]$,

- $[A_7] \leftarrow [A_0]$,

- $[CY] \leftarrow [A_0]$



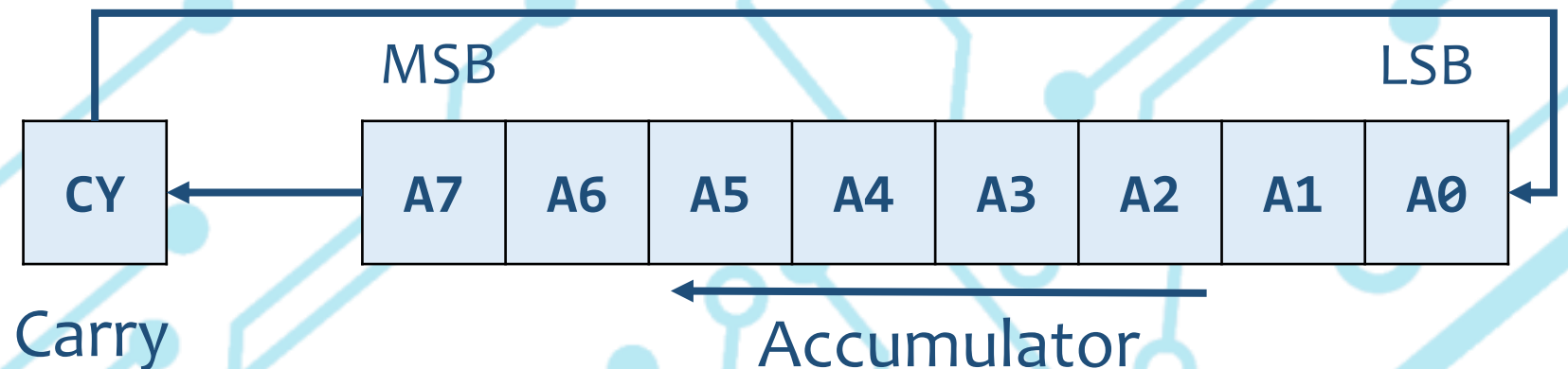
Rotate Instructions

| Opcode | Operands | Descriptions |
|--------|----------|---------------------------------------|
| RAL | None | Rotate Accumulator Left through Carry |

- Rotates the data of Accumulator and carry together (total 9-bits) to left by 1 position.
- New rotated data is stored back between Accumulator & CY.
- 1-byte instruction & It affects CY flag.

Example: **RAL**

- Equivalent to,
- $[A_{n+1}] \leftarrow [A_n]$,
- $[CY] \leftarrow [A_7]$,
- $[A_0] \leftarrow [CY]$



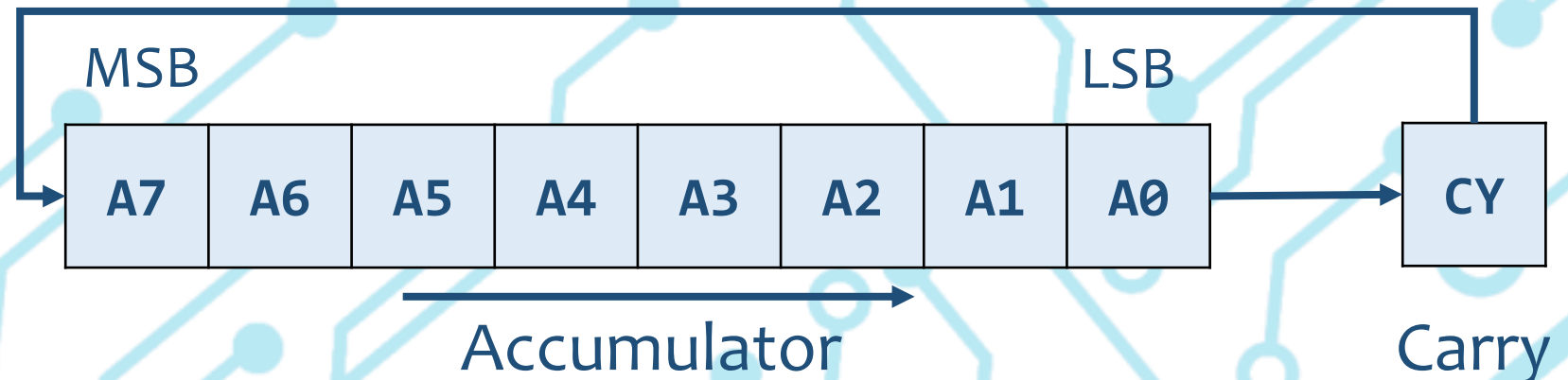
Rotate Instructions

| Opcode | Operands | Descriptions |
|--------|----------|--|
| RAR | None | Rotate Accumulator Right through Carry |

- Rotates the data of Accumulator and carry together (total 9-bits) to right by 1 position.
- New rotated data is stored back between Accumulator & CY.
- 1-byte instruction & It affects CY flag.

Example: **RAR**

- Equivalent to,
- $[A_n] \leftarrow [A_{n+1}]$,
- $[CY] \leftarrow [A_0]$,
- $[A_7] \leftarrow [CY]$



I/O Instructions

- These instructions move data between Accumulator and I/O devices.
- Content is not modified while transferring.
- I/O Instructions can be further Classified into:
 - IN Instruction
 - OUT Instruction

I/O Instructions

| Opcode | Operands | Descriptions |
|--------|----------|---|
| IN | port8 | Copy data to accumulator from the 8-bit port number specified |

- 🖨 Data of 8-bits from Port number specified within the instruction is copied into the accumulator.
- 🖨 2-byte instruction & does not affect any flags
- 🖨 Example: **IN 32H**

I/O Instructions

| Opcode | Operands | Descriptions |
|--------|----------|---|
| OUT | port8 | Copy data from accumulator to the 8-bit port number specified |

- 🖨 Data of 8-bits from Accumulator is copied into Port number specified within the instruction.
- 🖨 2-byte instruction & does not affect any flags
- 🖨 Example: **OUT 56H**

Machine Control Instructions

- Some Instructions are used to control program execution or to generate some delay to cope up with other devices.
- Those are known as Machine Control Instructions

Machine Control instructions

| Opcode | Operands | Descriptions |
|--------|----------|--------------|
| NOP | None | No Operation |

- 🖨 This instruction perform no operations at all.
- 🖨 It is used when the processor needs to wait for some data or event to occur.
- 🖨 It just wastes CPU cycles performing nothing.
- 🖨 1-byte instruction & does not affect any flags
- 🖨 Example: **NOP**

Machine Control instructions

| Opcode | Operands | Descriptions |
|--------|----------|---------------|
| HLT | None | Halt computer |

- 🖨 This instruction will stop the processor from executing program further.
- 🖨 Usually written as the last line of the program.
- 🖨 1-byte instruction. & does not affect any flags.
- 🖨 Example: **HLT**

Interrupt Instructions

- These instructions perform interrupts.
- An interrupt is a condition that causes the microprocessor to temporarily work on a different task, and then later return to its previous task.
- Upon execution of one of these instructions the program will perform jump to the subroutine for interrupt which is known as Interrupt Service Routine (ISR).
- After the execution of ISR is complete the main routine is resumed from where its execution was paused.
- So in execution it is similar to CALL. (Will be explained further in branching & looping instructions).

Restart instructions

| Opcode | Operands | Descriptions |
|--------|--|---|
| RST | digit here $0 \leq \text{digit} \leq 7$ | Performs unconditional jump of program to specified memory location |

- ❏ Performs interrupt call upon executing it.
- ❏ The jump address is fixed which can be calculated by doing $\text{digit} * 8$.
- ❏ Operations in RST digit:
 - ❏ $((\text{SP}) - 1) \leftarrow (\text{PCH})$
 - ❏ $((\text{SP}) - 2) \leftarrow (\text{PCL})$
 - ❏ $(\text{SP}) \leftarrow (\text{SP}) - 2$
 - ❏ $(\text{PC}) \leftarrow 8 * \text{digit}$

Restart Instructions

| Instruction | Calculation | Jump Location |
|-------------|--------------------|---|
| RST 0 | $0 * 8 = 0 = 00H$ | IF (Z = 0) then <i>Perform Jump</i> else <u>NOP</u> |
| RST 1 | $1 * 8 = 8 = 08H$ | IF (Z = 1) then <i>Perform Jump</i> else <u>NOP</u> |
| RST 2 | $2 * 8 = 16 = 10H$ | IF (CY = 0) then <i>Perform Jump</i> else <u>NOP</u> |
| RST 3 | $3 * 8 = 24 = 18H$ | IF (CY = 1) then <i>Perform Jump</i> else <u>NOP</u> |
| RST 4 | $4 * 8 = 32 = 20H$ | IF (P = 0) then <i>Perform Jump</i> else <u>NOP</u> |
| RST 5 | $5 * 8 = 40 = 28H$ | IF (P = 1) then <i>Perform Jump</i> else <u>NOP</u> |
| RST 6 | $6 * 8 = 48 = 30H$ | IF (S = 0) then <i>Perform Jump</i> else <u>NOP</u> |
| RST 7 | $7 * 8 = 56 = 38H$ | IF (S = 1) then <i>Perform Jump</i> else <u>NOP</u> |

Read & Set Interrupt Mask

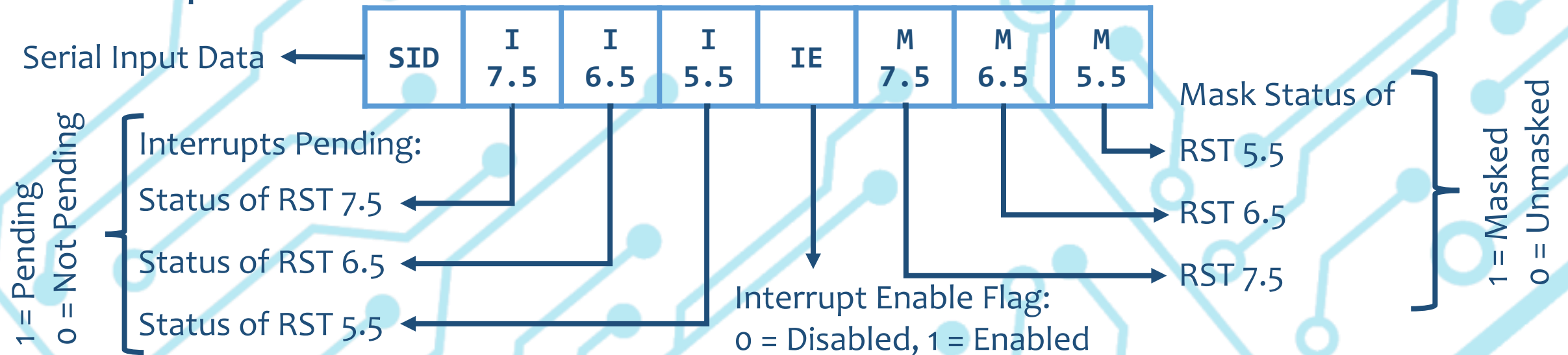
- ❏ Read / Write Serial Input / Output Data.
- ❏ The 8085 microprocessor allows user to enable/disable interrupts as a whole.
- ❏ 8085 has some hardware interrupts available, accessed using its pins. (TRAP, RST 7.5, RST 6.5, RST 5.5, INTR).
- ❏ 8085 allows user to mask some of them individually using its mask bits.
- ❏ Also RST 7.5 has memory so it can remember old pending requests.
- ❏ To read, set or clear these things we need to use RIM and SIM instructions.

Read & Set Interrupt Mask

| Opcode | Operands | Descriptions |
|--------|----------|---------------------|
| RIM | None | Read Interrupt Mask |

🖨️ This will read the interrupt mask and saves its 8-bit pattern in the accumulator.

🖨️ The pattern is as follows:

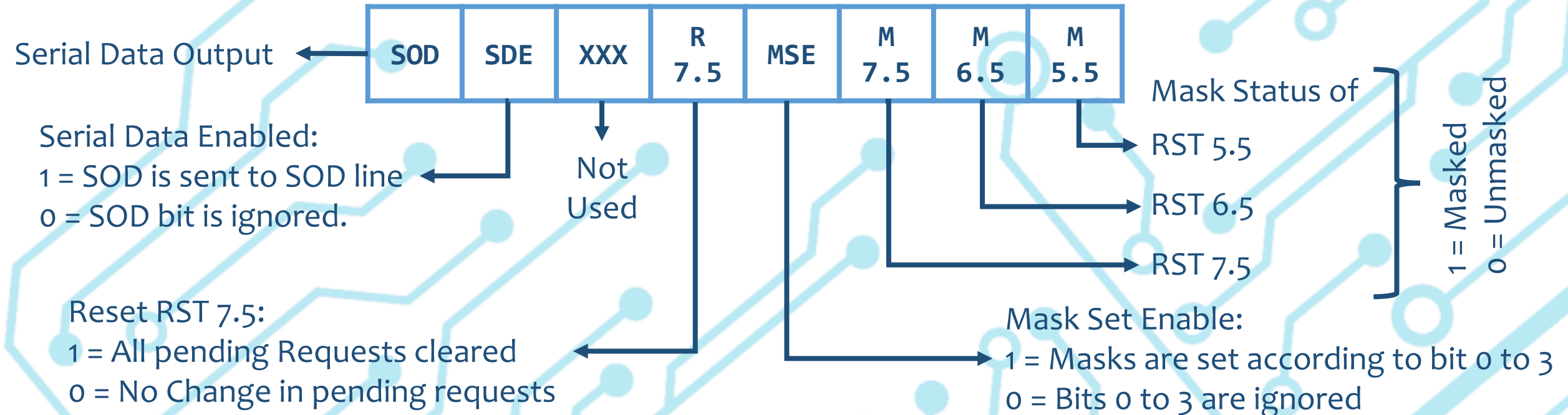


Read & Set Interrupt Mask

| Opcode | Operands | Descriptions |
|--------|----------|--------------------|
| SIM | None | Set Interrupt Mask |

🔌 This will set the interrupt mask from the 8-bit pattern saved in Accumulator.

🔌 The Pattern is as follows:



Branching Instructions

- These instructions perform branching i.e. it performs jumps in the program from one location to another, executing the program in and out of order fashion.
- Labels are used to perform jump.
- Label specifies an address of memory i.e. label is an address of 16-bits.
- Branching & Looping instructions are further classified into:
 - Jump Instructions
 - Call & Return Instructions

Jump instructions

| Opcode | Operands | Descriptions |
|--------|----------|---|
| JMP | add16 | Performs unconditional jump of program to specified memory location |

- ❏ Unconditional jump is performed in the program.
- ❏ It simply copies the address specified in next 2-bytes of instruction to the PC (Program Counter).
- ❏ 3-byte instruction & It does not affect any flags.
- ❏ Example: **JMP loop ;Where loop is a label specifying a location in memory.**
 - ❏ Equivalent to $[PC] \leftarrow [byte3][byte2]$

Jump Instructions

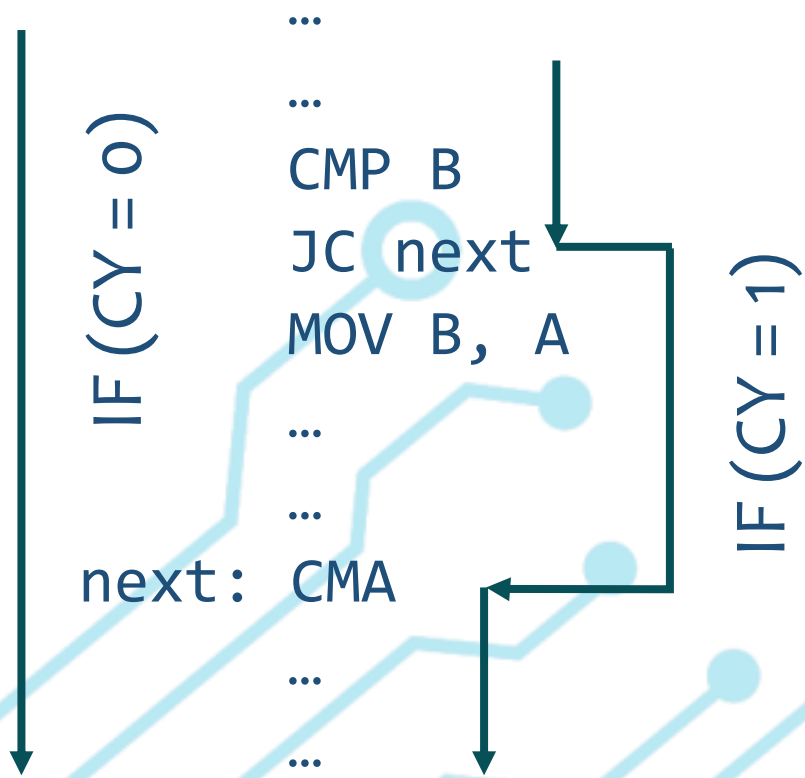
| Opcode | Condition | Descriptions |
|--------|-----------|---|
| JNZ | $Z = 0$ | IF ($Z = 0$) then <u>Perform Jump</u> else <u>NOP</u> |
| JZ | $Z = 1$ | IF ($Z = 1$) then <u>Perform Jump</u> else <u>NOP</u> |
| JNC | $CY = 0$ | IF ($CY = 0$) then <u>Perform Jump</u> else <u>NOP</u> |
| JC | $CY = 1$ | IF ($CY = 1$) then <u>Perform Jump</u> else <u>NOP</u> |
| JPO | $P = 0$ | IF ($P = 0$) then <u>Perform Jump</u> else <u>NOP</u> |
| JPE | $P = 1$ | IF ($P = 1$) then <u>Perform Jump</u> else <u>NOP</u> |
| JP | $S = 0$ | IF ($S = 0$) then <u>Perform Jump</u> else <u>NOP</u> |
| JM | $S = 1$ | IF ($S = 1$) then <u>Perform Jump</u> else <u>NOP</u> |

🖨 In generalized form we can write:

🖨 Jcondition add16/label

Jump Instructions

Example:



- In this example the jump is performed ahead of the program, So this jump is known as Forward Jump.
- If the jump is performed towards the previously executed lines of the program then the jump is known as Backward Jump.

Call & Return Instructions

- These instructions are used to perform a function/subroutine call. So these will also perform jump but in addition to that it will also store the address of the next line from where the call is performed. This address is known as return address.
- The return address is pushed into the stack.
- Operations in **CALL add16**:
 - $((SP) - 1) \leftarrow (PCH)$
 - $((SP) - 2) \leftarrow (PCL)$
 - $(SP) \leftarrow (SP) - 2$
 - $(PC) \leftarrow (byte3)(byte2)$

Call & Return Instructions

- The CALL instruction is generally used in conjunction with an appropriate RET (Return) instruction.
- RET instruction will perform a jump from the called subroutine to the main routine.
- The return address saved while performing CALL will be popped from the stack to the PC returning to the main routine from where it left.
- Operations in **RET**:
 - $(PCL) \leftarrow ((SP))$
 - $(PCH) \leftarrow ((SP) + 1)$
 - $(SP) \leftarrow (SP) + 2$

Call & Return Instructions

| Opcode | Condition | Descriptions |
|--------|-----------|---|
| CNZ | $Z = 0$ | IF ($Z = 0$) then <u>Perform Call</u> else <u>NOP</u> |
| CZ | $Z = 1$ | IF ($Z = 1$) then <u>Perform Call</u> else <u>NOP</u> |
| CNC | $CY = 0$ | IF ($CY = 0$) then <u>Perform Call</u> else <u>NOP</u> |
| CC | $CY = 1$ | IF ($CY = 1$) then <u>Perform Call</u> else <u>NOP</u> |
| CPO | $P = 0$ | IF ($P = 0$) then <u>Perform Call</u> else <u>NOP</u> |
| CPE | $P = 1$ | IF ($P = 1$) then <u>Perform Call</u> else <u>NOP</u> |
| CP | $S = 0$ | IF ($S = 0$) then <u>Perform Call</u> else <u>NOP</u> |
| CM | $S = 1$ | IF ($S = 1$) then <u>Perform Call</u> else <u>NOP</u> |

🖨 In generalized form we can write:

🖨 Ccondition add16/label

Call & Return Instructions

| Opcode | Condition | Descriptions |
|--------|-----------|---|
| RNZ | $Z = 0$ | IF ($Z = 0$) then <u>Perform Return</u> else <u>NOP</u> |
| RZ | $Z = 1$ | IF ($Z = 1$) then <u>Perform Return</u> else <u>NOP</u> |
| RNC | $CY = 0$ | IF ($CY = 0$) then <u>Perform Return</u> else <u>NOP</u> |
| RC | $CY = 1$ | IF ($CY = 1$) then <u>Perform Return</u> else <u>NOP</u> |
| RPO | $P = 0$ | IF ($P = 0$) then <u>Perform Return</u> else <u>NOP</u> |
| RPE | $P = 1$ | IF ($P = 1$) then <u>Perform Return</u> else <u>NOP</u> |
| RP | $S = 0$ | IF ($S = 0$) then <u>Perform Return</u> else <u>NOP</u> |
| RM | $S = 1$ | IF ($S = 1$) then <u>Perform Return</u> else <u>NOP</u> |

🖨 In generalized form we can write:

🖨 Rcondition add16/label

Call & Return Instructions

🖨 Example:

```
main: ...
```

```
...
```

```
CMP B
```

```
CALL sub
```

```
MOV B, A
```

```
...
```

```
...
```

```
sub: CMA
```

```
...
```

```
...
```

```
RET
```

