Notes on: 8085 Microprocessor_from_0

1.) 8085 Pin Diagram & Pin Functions

8085 Pin Diagram & Pin Functions

The 8085 Microprocessor is a 40-pin integrated circuit (IC) that forms the central processing unit (CPU) of a microcomputer system. The 'pin diagram' illustrates the physical layout of these 40 pins and their specific functions. Each pin serves as a connection point, allowing the 8085 to communicate and interact with various external components like memory, input/output (I/O) devices, and other support chips. Understanding these pin functions is fundamental to designing and troubleshooting 8085-based systems.

The 8085 operates on a single +5V power supply and provides all the necessary control and data signals to manage a complete system. Its pins are categorized into several functional groups to simplify understanding.

1. Power Supply and Clock Signals

These pins are essential for the microprocessor to power on and operate in a synchronized manner.

- VCC (Pin 40): This is the positive power supply pin, requiring a +5V DC supply for the 8085 to function.
- VSS (Pin 20): This is the ground reference pin for the power supply.
- X1 (Pin 1) and X2 (Pin 2): These pins are inputs for connecting an external crystal or RC network. An internal oscillator then generates the fundamental clock frequency for the 8085. For instance, if a 6 MHz crystal is connected, the internal operating frequency of the 8085 will be 3 MHz (crystal frequency divided by 2).
- CLK OUT (Pin 37): This pin provides an output clock signal, which is typically half the crystal frequency (e.g., 3 MHz if a 6 MHz crystal is used). This clock signal can be used to synchronize other peripheral devices in the system with the 8085's operations.

2. Address Bus

The address bus is used by the 8085 to identify memory locations or I/O ports. It is unidirectional, meaning information flows only from the 8085 to the external devices. The 8085 has a 16-bit address bus, enabling it to access 2^16 = 65,536 (64 KB) unique memory locations or 256 I/O ports.

- A8-A15 (Pins 21-28): These are the higher-order 8 bits of the address bus. They are unidirectional output pins, always carrying address information. When the 8085 needs to read from or write to a specific memory location or I/O port, it places the 16-bit address on the address bus. A8-A15 carry the most significant 8 bits of this address.

3. Multiplexed Address/Data Bus

To reduce the total number of pins on the IC, the 8085 uses multiplexing for its lower-order address and data lines. This means the same 8 pins carry both address and data information at different times.

- AD0-AD7 (Pins 12-19): These are the lower-order 8 bits of the address bus (A0-A7) and the 8-bit data bus (D0-D7).
- During the first clock cycle of a machine cycle, these pins function as the lower-order address bus (A0-A7), carrying the least significant 8 bits of the memory or I/O address.
- During subsequent clock cycles, after the address has been latched by an external latch (explained by ALE below), these pins function as the bidirectional data bus (D0-D7), carrying 8-bit data between the 8085 and memory/I/O devices.
- Analogy: Imagine a single lane on a bridge that is sometimes used by cars (address) and

sometimes by trucks (data). A traffic controller (ALE) signals when cars are allowed and when trucks are allowed. This shared use reduces the number of lanes needed for the bridge.

4. Control and Status Signals

These signals control the flow of data and provide status information about the 8085's current operation.

• ALE (Address Latch Enable) (Pin 30): This is an output signal that goes high during the first clock cycle of a machine cycle. It indicates that the information on AD0-AD7 is address information (A0-A7). This signal is crucial for **demultiplexing** the AD0-AD7 lines. An external latch (like an 8212 IC) uses ALE to capture and hold the lower-order address bits from AD0-AD7, separating them from the data that will appear later on the same lines.
• RD- (Read) (Pin 31): This is an active-low output signal. When RD- is low, it indicates that the 8085 is performing a read operation, meaning it is fetching data from memory or an I/O device.
• WR- (Write) (Pin 32): This is an active-low output signal. When WR- is low, it indicates that the 8085 is performing a write operation, meaning it is sending data to memory or an I/O device.
• IO/M- (I/O or Memory) (Pin 34): This is an output signal that distinguishes between memory and I/O operations. When IO/M- is high, the 8085 is accessing an I/O device. When it is low, the 8085 is accessing memory. This signal, along with RD- and WR-, helps external devices decode whether they are being addressed and for what purpose (read/write).
• S0 (Pin 33) and S1 (Pin 29): These are output status signals that, in combination, indicate the type of machine cycle currently being executed by the 8085. For example:
• S1=1, S0=1: Opcode Fetch cycle
• S1=1, S0=0: Memory Read cycle
• S1=0, S0=1: Memory Write cycle
• S1=0, S0=0: I/O Read or I/O Write cycle
• READY (Pin 35): This is an input signal. If a peripheral device is slow and cannot respond immediately to a read or write request, it can pull the READY line low. This forces the 8085 to enter wait states, pausing its operation until the peripheral asserts READY high again, indicating it is ready for data transfer. This ensures proper synchronization between the fast CPU and slower peripherals.
• HOLD (Pin 39): This is an input signal used by external devices (like a Direct Memory Access - DMA controller) to request control of the system bus. When HOLD is asserted high, the 8085 will relinquish control of its address and data buses as soon as its current machine cycle is complete. This allows the DMA controller to directly access memory without CPU intervention, speeding up data transfers.
• HLDA (Hold Acknowledge) (Pin 38): This is an output signal. When the 8085 receives a HOLD request, it asserts HLDA high to acknowledge that it has relinquished control of its buses.

5. Interrupt Signals

Interrupts are mechanisms that allow external devices to temporarily suspend the 8085's normal program execution and force it to execute a special routine (Interrupt Service Routine - ISR) to handle a particular event.

• TRAP (Pin 6): This is a non-maskable, highest priority interrupt input. It cannot be disabled by software. It is typically used for critical events like power failure or emergency shutdowns. When TRAP is activated, the 8085 vectors to a fixed memory location (0024H) to execute its ISR.
• RST 7.5 (Pin 7), RST 6.5 (Pin 8), RST 5.5 (Pin 9): These are maskable, vectored interrupt inputs. 'Maskable' means they can be enabled or disabled by software instructions. 'Vectored' means each interrupt automatically directs the CPU to a specific memory location to fetch the ISR address. RST 7.5 has the highest priority among these, followed by RST 6.5 and RST 5.5. Their respective vector addresses are 003CH, 0034H, and 002CH.
• INTR (Interrupt Request) (Pin 10): This is a general-purpose, maskable interrupt input with the lowest priority. When INTR is activated, the 8085 responds by issuing an INTA- signal. Unlike the RST interrupts, INTR is non-vectored, meaning the interrupting device must provide the vector address (typically via the data bus) for the ISR.
• INTA- (Interrupt Acknowledge) (Pin 11): This is an active-low output signal. It is asserted by the 8085 to acknowledge an INTR request, indicating that it is ready to receive the vector address from the interrupting device.

6. Serial I/O Signals

These pins facilitate serial data communication with external devices.

- SID (Serial Input Data) (Pin 5): This is an input pin for receiving serial data. The RIM (Read Interrupt Mask) instruction can read the status of this pin.
- SOD (Serial Output Data) (Pin 4): This is an output pin for sending serial data. The SIM (Set Interrupt Mask) instruction can set the status of this pin. These pins are useful for simple serial communication, debugging, or interfacing with devices that use a serial protocol.

7. Reset Signals

These pins are used to reset the microprocessor and external devices.

- RESET IN- (Pin 36): This is an active-low input signal. When this pin is held low for a minimum number of clock cycles, the 8085 resets. This initializes the program counter (PC) to 0000H, disables interrupts (except TRAP), and clears certain internal registers, effectively restarting the microprocessor.
- RESET OUT (Pin 3): This is an output signal. It goes high when the 8085 is being reset (i.e., RESET IN- is active) or when the 8085 itself is initiating a reset sequence. This signal can be used to reset other peripheral devices in the system simultaneously with the 8085.

Summary of Key Points:

- The 8085 is a 40-pin IC, requiring +5V power and ground.
- X1/X2 and CLK OUT provide clocking for the system.
- The 16-bit address bus (A8-A15 and AD0-AD7 in address mode) allows access to 64KB of memory.
- The 8-bit data bus (AD0-AD7 in data mode) facilitates 8-bit data transfer.
- Multiplexing of AD0-AD7 reduces pin count, with ALE used for demultiplexing.
- Control signals (RD-, WR-, IO/M-, S0, S1) manage data flow and indicate machine cycle status.
- READY allows slow peripherals to insert wait states.
- HOLD/HLDA provide bus control for DMA operations.
- Interrupts (TRAP, RST 7.5/6.5/5.5, INTR/INTA-) allow external events to trigger CPU action.
- SID/SOD enable basic serial communication.
- RESET IN-/RESET OUT initialize the CPU and system.

Understanding these pins is crucial for interfacing the 8085 with memory, I/O devices, and building complete microcomputer systems.

2.) 8085 Microprocessor Architecture

The 8085 Microprocessor Architecture refers to the internal design and organization of the 8085 central processing unit (CPU). It defines how different components within the chip are structured and interact to perform its primary function: processing data and executing instructions. Understanding this architecture is fundamental to programming and interfacing with the 8085.

The 8085 is an 8-bit microprocessor, meaning it processes data in chunks of 8 bits. Its architecture is divided into several functional units, each with a specific role. Think of it like a small, highly organized factory where different departments specialize in various tasks to produce a final product.

Key Components of 8085 Microprocessor Architecture:

1. Arithmetic Logic Unit (ALU)
   - This is the computational engine of the microprocessor.
   - The ALU performs all arithmetic operations like addition, subtraction, increment, decrement.
   - It also handles logical operations such as AND, OR, XOR, and complements.
   - Example: If you want to add two numbers, say 5 and 3, the ALU is the part that actually calculates the sum, 8.

• Real-world analogy: The calculation department in a factory, responsible for all mathematical and comparison tasks.

2. Registers
• Registers are small, high-speed memory locations built directly into the CPU.
• They temporarily store data, addresses, and control information during program execution.
• The 8085 has various types of registers:
• Accumulator (A Register): An 8-bit register crucial for most arithmetic and logical operations. All 8-bit operations typically involve the accumulator, either as a source or destination for data.
• Temporary Registers (W and Z): These 8-bit registers are not user-accessible. They are used internally by the microprocessor to hold temporary data during instruction execution, for example, during the execution of a LXI instruction to load a 16-bit value into a register pair.
• Program Counter (PC): A 16-bit register that stores the memory address of the next instruction to be fetched from memory. It acts as a pointer, always keeping track of where the program is. After an instruction is fetched, the PC is automatically incremented.
• Stack Pointer (SP): A 16-bit register that points to the top of a special memory area called the Stack. The Stack is used for temporarily storing data or addresses, especially during subroutine calls and interrupts.
• General Purpose Registers (B, C, D, E, H, L): These are 8-bit registers that can be used by the programmer to store data during program execution. They can also be combined into 16-bit register pairs (BC, DE, HL) for certain operations, particularly for memory addressing. (Details on these will be covered later).
• Flag Register: An 8-bit register that stores status flags (like Zero, Carry, Parity, Sign) after arithmetic or logical operations. These flags provide information about the result of the last operation. (Details on this will be covered later).

3. Instruction Register and Decoder
• Instruction Register (IR): After an instruction is fetched from memory, it is temporarily stored in this 8-bit register.
• Instruction Decoder: This unit then interprets the binary code (opcode) in the Instruction Register. It deciphers what operation needs to be performed based on the instruction.
• Real-world analogy: A language interpreter. It takes a command (the instruction) and translates it into actions the factory (microprocessor) understands.

4. Timing and Control Unit
• This is the central nervous system of the microprocessor.
• It generates the necessary timing and control signals for all internal and external operations.
• These signals synchronize all operations within the microprocessor and with external devices.
• It controls the flow of data between the CPU and memory/I/O devices, based on the decoded instruction.
• It orchestrates the fetch, decode, and execute cycles of instructions.
• Example: When an instruction needs to read data from memory, this unit generates the appropriate read signal (RD) and controls the timing of data transfer.
• Real-world analogy: The factory manager who provides precise instructions and timing to all departments (ALU, registers, memory access) to ensure everything runs smoothly and in sequence.

5. Interrupt Control
• This unit handles interrupt requests from external devices.
• Interrupts are signals that can temporarily suspend the CPU's current operation to deal with a more urgent event.
• The 8085 has several interrupt inputs (TRAP, RST7.5, RST6.5, RST5.5, INTR) and an interrupt acknowledge output (INTA).
• Real-world analogy: A system for handling urgent calls or emergencies in the factory, allowing the manager to pause routine work and address critical issues.

6. Serial I/O Control
• The 8085 includes dedicated hardware for serial communication.
• It supports two serial data lines: SID (Serial Input Data) and SOD (Serial Output Data).
• This allows the microprocessor to send and receive data bit by bit, which is useful for connecting to

devices that require serial communication, like modems or other microcontrollers over a simple two-wire link.

7. Address Buffer and Data Buffer
• These are essentially buffers (temporary storage areas) that drive the external address and data buses.
• Address Buffer: Buffers the 16-bit address generated by the CPU to be sent to memory or I/O devices. It typically has a higher current driving capability to connect to external devices.
• Data Buffer: Buffers the 8-bit data being transferred between the CPU and memory/I/O devices. It's often bidirectional.
• Real-world analogy: Loading docks at the factory. One for sending out delivery instructions (addresses) and another for receiving raw materials or shipping finished goods (data).

8. Internal Data Bus
• This is an 8-bit internal pathway within the 8085 microprocessor.
• All internal components (ALU, registers, instruction register) communicate with each other via this bus.
• Data moves between these units through this common pathway.
• Real-world analogy: The internal corridor or conveyor belt system that connects all departments within the factory, allowing data (materials) to move efficiently from one workstation to another.

How Components Interact:
When the 8085 executes a program, the Timing and Control Unit, guided by the Program Counter, fetches an instruction from memory. This instruction is placed in the Instruction Register and then decoded. Based on the decoded instruction, the Timing and Control Unit generates the necessary signals to move data between registers, perform operations in the ALU, and interact with external memory or I/O devices. All internal data transfers happen over the Internal Data Bus, while external transfers use the Address and Data Buffers to connect to the external buses.

Summary of Key Points:
• The 8085 is an 8-bit microprocessor with a structured internal design.
• The ALU is responsible for all arithmetic and logical operations.
• Registers provide high-speed temporary storage for data, addresses (PC, SP), and intermediate results (Accumulator).
• The Instruction Register and Decoder interpret fetched instructions.
• The Timing and Control Unit is the orchestrator, generating signals for all operations.
• Interrupt Control handles urgent external requests, and Serial I/O Control manages bit-by-bit data transfer.
• Buffers (Address and Data) interface the internal architecture with external memory and I/O.
• The Internal Data Bus facilitates communication among all internal components.
Understanding this architecture is vital for comprehending how instructions are processed and how the 8085 interacts with the outside world.

3.) 8085 General Purpose Registers

Introduction to Registers in the 8085 Microprocessor

Within the realm of Computer Organization And Architecture, understanding how a microprocessor manages data is fundamental. The 8085 Microprocessor, as a classic example of an 8-bit CPU, relies heavily on its internal storage units known as registers. These registers are not part of the main memory (RAM) but are high-speed storage locations directly integrated into the CPU itself. They act as the processor's immediate workspace, akin to a small, fast workbench where tools and materials are kept ready for immediate use.

Why Registers are Essential for CPU Operations:
1. Exceptional Speed: Registers offer the fastest possible access time for data storage and retrieval, far surpassing that of main memory. This speed is critical for the processor to execute instructions and

process data with minimal delay, directly impacting the overall performance of the system.
2. Temporary Data Repository: They serve as crucial temporary holding areas for various types of information, including numerical data, characters, program instructions, memory addresses, and the intermediate results generated during computations.
3. Direct and Instantaneous Access: Unlike main memory, which requires complex addressing mechanisms and multiple clock cycles to access data via the system buses, registers are accessed directly by the CPU's internal logic. This direct access capability enables the CPU to retrieve or store data in registers almost instantaneously.

Types of Registers in the 8085:
The 8085 Microprocessor's architecture includes different categories of registers, each designed for specific purposes:
- General Purpose Registers (GPRs): These are versatile registers that a programmer can utilize for a broad range of data storage and manipulation tasks, offering flexibility in program design.
- Special Purpose Registers: These registers have predefined, dedicated functions essential for the CPU's operation, such as the Program Counter (to track instruction flow), Stack Pointer (for managing the stack), and the Flag Register (to indicate the status of operations). Our focus here is exclusively on the General Purpose Registers.

8085 General Purpose Registers: Detailed Explanation

The 8085 Microprocessor is equipped with six general-purpose registers, each uniquely identified by a single letter: B, C, D, E, H, and L. These registers are foundational for data handling in 8085 assembly language programming.

Characteristics of Individual GPRs:
1. Fixed Size: Every single one of these general-purpose registers is 8 bits wide. This means each register is capable of storing an 8-bit binary value, which is equivalent to one byte of data. An 8-bit value can represent numbers from 0 to 255, or specific character codes.
2. Versatility in Use: Programmers have the flexibility to use these registers to store any 8-bit piece of data required by the program. This could be a numerical operand for an arithmetic calculation, an ASCII character, or a byte of data fetched from a memory location.
3. Optimization for Data Access: By storing frequently accessed data in GPRs, the CPU minimizes the need to fetch data from slower main memory, significantly enhancing processing speed and program efficiency.

Register Pairs: Enhancing Data Handling to 16-bit
A powerful feature of the 8085's GPRs is their ability to be logically grouped together to form 16-bit register pairs. While the 8085 is fundamentally an 8-bit processor, the need to handle 16-bit values arises frequently, particularly for addressing the microprocessor's full memory space (up to 64 KB, which requires 16-bit addresses).

The designated 16-bit register pairs are:
- BC Pair: Comprises register B as the High-order byte (Most Significant Byte - MSB) and register C as the Low-order byte (Least Significant Byte - LSB). This pair is often used for general 16-bit data storage or for addressing.
- DE Pair: Consists of register D as the MSB and register E as the LSB. Similar to BC, it can store 16-bit data or memory addresses.
- HL Pair: Formed by register H as the MSB and register L as the LSB. The HL pair holds a particularly significant role as it is the primary register pair used for memory addressing. It acts as a data pointer for many memory access instructions.

The Strategic Importance of Register Pairs:
- Efficient Memory Addressing: The 8085 uses a 16-bit address bus to access its 64 KB memory space. Register pairs, especially HL, are indispensable for storing these 16-bit memory addresses. This allows the CPU to point to and access any location within its entire memory range.
- Facilitating 16-bit Operations: Although the 8085 is an 8-bit processor, using register pairs enables it to effectively manage and manipulate 16-bit data. This is crucial for operations such as loading a 16-bit constant into a register or performing basic 16-bit arithmetic (like 16-bit increment/decrement).

The Accumulator (Register A): The Heart of 8-bit Operations

Among all the general-purpose registers, the Accumulator, simply referred to as 'A', holds a preeminent and central position in the 8085 Microprocessor's operation. It is an 8-bit register, similar in size to B, C, D, E, H, and L, but its functional role is uniquely critical and specialized.

Defining Characteristics of the Accumulator:
1. Core for Arithmetic and Logic Unit (ALU) Operations: The Accumulator is the primary operand and destination register for almost all 8-bit arithmetic operations (e.g., addition, subtraction, increment, decrement) and logical operations (e.g., AND, OR, XOR, complement). Without the Accumulator, the ALU cannot perform most of its fundamental calculations.
2. Implicit Operand Design: In many 8085 instructions, when an arithmetic or logical operation is specified, one of the operands is automatically assumed to be present in the Accumulator. For instance, an **ADD B** instruction implies adding the content of register B to the content of the Accumulator. The result of such an operation is always stored back into the Accumulator, effectively overwriting its previous content.
3. Central Data Transfer Hub: The Accumulator serves as the main gateway for data exchange between the CPU and external memory or Input/Output (I/O) devices. When the CPU reads data from a memory location or an input port, that data is typically loaded into the Accumulator first. Conversely, any data intended for writing to memory or sending to an output port usually originates from the Accumulator.

Real-world Analogy for the Accumulator's Role:
Imagine a single-purpose calculator that only has one visible display. When you enter a number, it appears there. When you perform an operation like adding another number, the number already on the display is automatically used in the calculation, and the result immediately replaces it on the same display. This display is analogous to the 8085's Accumulator – it's where the active computation happens, and where results are temporarily held.

Overall Significance and Practical Application of General Purpose Registers

The strategic and effective utilization of GPRs is paramount for any programmer working with the 8085, directly influencing the performance and compactness of the assembly language code.

Key Advantages in Programming:
- Optimized Performance: By minimizing the need for the CPU to access slower external memory, GPRs contribute significantly to faster program execution. Keeping active data in registers drastically reduces memory bus traffic.
- Enhanced Program Efficiency: Using registers for temporary storage, loop counters, and pointers makes programs not only faster but also more efficient in terms of instruction cycles.
- Simplified Data Management: GPRs provide flexible spaces to temporarily hold variables and intermediate values, simplifying the logic of data manipulation within a program. For a diploma student, understanding how to assign variables to specific registers (e.g., 'count' in register C, 'data_offset' in register D) is a foundational skill.

Conceptual Examples of GPR Usage for a Diploma Student:
- Storing Loop Counts: A common practice is to load a value into a GPR (like register C) to act as a counter for a loop. The program would decrement C in each iteration until it reaches zero.
- Manipulating Data Segments: When processing a block of data in memory, the HL register pair might hold the starting address of the data block, while the DE pair could hold the starting address of a destination block where the data needs to be copied.
- Performing Calculations: If you need to add three numbers (A, B, C), you might load the first number into the Accumulator, add the second number (from register B) to it, and then add the third number (from register D) to the current sum in the Accumulator.

In summary, the 8085's General Purpose Registers (B, C, D, E, H, L, and the unique Accumulator A) are 8-bit internal memory locations within the CPU. They are critical for high-speed temporary data storage and for holding intermediate results. While each can store an 8-bit value, they can be paired (BC, DE, HL) to handle 16-bit values, most notably for memory addressing. The Accumulator plays a

pivotal role as the primary register for all 8-bit arithmetic and logical operations, and as a central point for data transfers. Mastering the use of these registers is essential for understanding the 8085's operation and for effective programming.

4.) 8085 Flag Register

1. Introduction to the 8085 Flag Register

The 8085 Microprocessor, like most Central Processing Units (CPUs), has a special register called the Flag Register. This register is not used for storing data directly, but rather to store the **status** or **conditions** that result from the execution of arithmetic and logical operations. Think of it as the CPU's status report card after performing calculations. It provides crucial information about the outcome of an operation, which can then be used to make decisions.

2. Location and Size of the 8085 Flag Register

The 8085 Flag Register is an 8-bit register. It is not directly accessible by programmers as a standalone register like the General Purpose Registers (B, C, D, E, H, L) or the Accumulator (A). Instead, it is typically accessed as part of a 16-bit register pair known as the Program Status Word (PSW). The PSW consists of the Accumulator (A) and the Flag Register. When you need to save or retrieve the status of the flags, you work with the PSW.

3. Individual Flags (Bits) within the Flag Register

The 8-bit Flag Register consists of five individual flip-flops (bits) that are actively used to indicate different conditions. The remaining three bits are reserved for future use and are not affected by operations in the 8085. Each flag is a single bit, set to 1 or reset to 0 based on the result of the last arithmetic or logical operation.

Let's explore each of these active flags:

a. Sign Flag (S)
- Position: This is the D7 (most significant bit) of the Flag Register.
- Function: The Sign Flag is set (1) if the most significant bit (MSB) of the result of an operation is 1. It is reset (0) if the MSB is 0. In signed number representation, the MSB indicates the sign of the number (1 for negative, 0 for positive).
- Example:
- If an operation results in 10010110 (binary), the S flag will be set to 1 because the MSB (D7) is 1, indicating a negative result if signed interpretation is used.
- If an operation results in 01101001 (binary), the S flag will be reset to 0 because the MSB (D7) is 0, indicating a positive result.
- Real-world understanding: Useful for checking if a calculation resulted in a positive or negative value, particularly in computations involving signed numbers.

b. Zero Flag (Z)
- Position: This is the D6 bit of the Flag Register.
- Function: The Zero Flag is set (1) if the result of an operation is 00H (all bits are zero). It is reset (0) if the result is anything other than 00H.
- Example:
- If you subtract two identical numbers (e.g., MOV A, 05H; SUB 05H), the result will be 00H, and the Z flag will be set to 1.
- If you add two numbers resulting in 08H (e.g., ADD 03H and 05H), the Z flag will be reset to 0.
- Real-world understanding: Extremely useful for loop control (e.g., decrementing a counter until it reaches zero) and for comparing two values (if their difference is zero, they are equal).

c. Auxiliary Carry Flag (AC)
- Position: This is the D4 bit of the Flag Register.

• Function: The Auxiliary Carry Flag is set (1) if there is a carry out from the D3 bit to the D4 bit during an arithmetic operation. It is reset (0) otherwise. This flag is primarily used for Binary Coded Decimal (BCD) arithmetic operations.

• Example:

• If you add 09H and 02H in binary:

0000 1001 (09H)
+ 0000 0010 (02H)

• ----------

0000 1011 (0BH)
Here, there is no carry from D3 to D4. So, AC = 0.

• If you add 09H and 08H in binary:

0000 1001 (09H)
+ 0000 1000 (08H)

• ----------

0001 0001 (11H)
Here, there is a carry from D3 (first 1 in 1001) to D4 (second 0 in 1000). So, AC = 1.

• Real-world understanding: Though not directly used by most programmers, it's vital for internal CPU operations, especially for BCD arithmetic correction (using instructions like DAA - Decimal Adjust Accumulator) in applications requiring precise decimal calculations (e.g., financial systems, calculators).

d. Parity Flag (P)

• Position: This is the D2 bit of the Flag Register.

• Function: The Parity Flag is set (1) if the result of an operation contains an even number of set bits (1s). It is reset (0) if the result contains an odd number of set bits.

• Example:

• If an operation results in 00001011 (binary), it has three set bits (1s). Three is an odd number, so the P flag will be reset to 0.

• If an operation results in 00001101 (binary), it has three set bits (1s). Three is an odd number, so the P flag will be reset to 0.

• If an operation results in 00000110 (binary), it has two set bits (1s). Two is an even number, so the P flag will be set to 1.

• Real-world understanding: Used for simple error checking in data transmission. By transmitting data along with its parity, a receiver can detect if a single bit error occurred during transmission.

e. Carry Flag (CY)

• Position: This is the D0 (least significant bit) of the Flag Register.

• Function: The Carry Flag is set (1) if an arithmetic operation results in a carry out from the D7 bit (the most significant bit of the 8-bit result). In subtraction, it acts as a borrow flag, being set if a borrow is required. It is reset (0) otherwise.

• Example:

• Addition:

1100 1010 (CAH)
+ 0110 0101 (65H)

• ----------

1 0010 1111 (Carry out from D7, result is 2FH)
Here, a carry is generated from D7. So, CY = 1.

• Subtraction (SUB B means A - B):

If A = 05H (0000 0101) and B = 07H (0000 0111)
Subtracting B from A requires a borrow. So, CY = 1.

• Real-world understanding: Crucial for multi-precision (multi-byte) arithmetic, where a carry from one 8-bit operation needs to be propagated to the next higher 8-bit operation (e.g., adding two 16-bit numbers using two 8-bit additions). Also used for error detection in certain algorithms and for implementing rotate instructions.

f. Unused Flags/Reserved Bits

• Positions: D1, D3, D5 bits of the Flag Register.

• Function: These bits are not used by the 8085 microprocessor. They are generally left at a fixed state (often 0 or 1, depending on the specific processor model/variant) or are undefined. Programmers should avoid relying on their values.

4. How Flags are Affected

Flags are updated automatically by the CPU after the execution of certain instructions.
- Arithmetic instructions (ADD, SUB, INR, DCR, etc.) universally affect most flags (S, Z, AC, P, CY).
- Logical instructions (ANA, ORA, XRA, CMP, etc.) also affect flags, typically setting CY to 0 and AC as per the operation.
- Data Transfer instructions (MOV, LXI, LDA, STA, etc.) generally do NOT affect any flags. They simply move data without performing any calculations.
- Branch instructions (JMP, CALL, RET, etc.) also do not affect flags directly, though conditional branch instructions *use* the flag values to decide whether to jump.

5. Importance and Real-World Application

The Flag Register is fundamental to the decision-making capability of the microprocessor.
- Conditional Branching: The most significant use is in conditional jump or call instructions (e.g., JZ - Jump if Zero, JNC - Jump if No Carry). These instructions check the state of a specific flag to decide whether to alter the program flow. This allows programs to implement loops, 'if-else' statements, and other control structures. You will delve deeper into this when studying 8085 Instruction Execution.
- Error Detection: The Parity Flag assists in basic data integrity checks for data communication.
- Multi-precision Arithmetic: The Carry Flag is essential for performing arithmetic operations on numbers larger than 8 bits, where carries must be propagated across multiple bytes.
- Algorithm Implementation: Flags are used in various algorithms for tasks like bit manipulation, array processing, and complex numerical computations where intermediate results' properties need to be tracked.
- Signed Number Arithmetic: The Sign Flag provides quick insight into the polarity of a calculated value.

6. Summary of Key Points

- The 8085 Flag Register is an 8-bit register that stores the status of operations.
- It is part of the Program Status Word (PSW) along with the Accumulator.
- Five active flags are:
- Sign (S): Indicates if the result's MSB is 1 (negative for signed numbers).
- Zero (Z): Indicates if the result is 00H.
- Auxiliary Carry (AC): Indicates carry from D3 to D4 (used for BCD).
- Parity (P): Indicates if the result has an even or odd number of set bits.
- Carry (CY): Indicates a carry out from D7 (or borrow in subtraction).
- Flags are automatically updated by arithmetic and logical instructions.
- They are crucial for conditional program execution, multi-precision arithmetic, and error detection, enabling the microprocessor to make intelligent decisions.

5.) 8085 Instruction Execution (Fetch, Decode, Execute operations)

Introduction to 8085 Instruction Execution

The 8085 microprocessor, like any CPU, operates by continuously fetching instructions from memory, understanding what they mean, and then performing the required operations. This fundamental process is known as the instruction execution cycle or fetch-decode-execute cycle. It's the core activity that allows a computer program to run. Imagine it as the microprocessor's constant workflow.

The entire process is orchestrated by the Control Unit, which uses timing and control signals to synchronize the operations between various components like memory, registers, and the ALU.

1. The Instruction Cycle

The instruction cycle is the basic operation cycle of a CPU. For the 8085, it typically consists of three main stages:

- Fetch: Retrieving the next instruction from memory.
- Decode: Interpreting the fetched instruction to understand what operation it represents.
- Execute: Performing the actual operation specified by the instruction.

This cycle repeats continuously as long as the microprocessor is powered on and running a program. Each complete cycle allows one instruction to be processed.

2. Phase 1: Fetch Operation

The fetch operation is the first step where the microprocessor retrieves an instruction from the main memory. It's like the CPU **reading** the next step from its instruction list.

Steps involved in fetching an instruction:

- The Program Counter (PC), a special 16-bit register, holds the memory address of the next instruction to be executed.
- The content of the PC is transferred to the Address Bus. The Address Bus is unidirectional and carries the memory address to the RAM chip.
- The Control Unit then issues a 'Memory Read' signal (RD-bar) on the Control Bus. This signal tells the memory unit to output the data stored at the specified address.
- The memory unit at the specified address places the instruction (an 8-bit opcode) onto the Data Bus. The Data Bus is bidirectional and carries data between the CPU and memory/I/O.
- The microprocessor receives this 8-bit instruction from the Data Bus and stores it in the Instruction Register (IR).
- After fetching the instruction, the Program Counter is automatically incremented by 1 to point to the next byte in memory, which could be another instruction or an operand for the current instruction.

Example: If PC contains 2000H, the CPU fetches the instruction at memory location 2000H. After fetching, PC becomes 2001H.

3. Phase 2: Decode Operation

Once an instruction is fetched and placed in the Instruction Register, the next step is to decode it. This phase is where the microprocessor understands what the fetched instruction actually means.

- The 8-bit opcode from the Instruction Register is sent to the Instruction Decoder within the CPU.
- The Instruction Decoder is a combinational logic circuit that interprets the bit pattern of the opcode. It recognizes which specific operation (e.g., ADD, MOV, JMP) the instruction represents.
- Based on the decoded instruction, the Instruction Decoder generates a sequence of micro-operations and corresponding control signals. These signals are specific electrical pulses that direct other parts of the CPU to perform the necessary actions for execution.
- This decoding process ensures that the right internal components (like the ALU, specific registers, or memory interface) are prepared for the upcoming execution phase.

Analogy: Imagine you pick up a recipe card (fetch). Now you read the card to understand if it's for 'baking a cake' or 'making coffee' (decode).

4. Phase 3: Execute Operation

The execute phase is where the actual operation specified by the decoded instruction is performed. This is the **doing** part of the cycle. The control signals generated during the decode phase guide this operation.

The execution can involve various types of operations:

- Data Transfer Operations (e.g., MOV R1, R2): If the instruction is to move data between registers, the Control Unit activates the internal data paths to copy data from the source register (R2) to the

destination register (R1). No ALU involvement.

• Arithmetic/Logic Operations (e.g., ADD B): If the instruction involves arithmetic or logical computation, the operands are sent to the Arithmetic Logic Unit (ALU). For example, the content of register B and the Accumulator are sent to the ALU, the ALU performs addition, and the result is stored back in the Accumulator. The Flag Register is also updated based on the result.

• Memory Access Operations (e.g., LDA 2050H): If the instruction requires reading data from or writing data to memory, the Control Unit manages the Address Bus, Data Bus, and issues appropriate control signals (RD-bar for read, WR-bar for write) to perform the memory transaction. The address specified by the instruction (e.g., 2050H) is put on the Address Bus.

• I/O Operations (e.g., IN 05H): For input/output, the Control Unit sends control signals to I/O devices via the Control Bus, and data is transferred over the Data Bus to or from an I/O port.

• Program Control Operations (e.g., JMP 3000H): For jump or call instructions, the Program Counter's content is updated with the new address (3000H in this case), thereby altering the flow of program execution.

Once the instruction is executed, the cycle completes, and the microprocessor is ready to fetch the next instruction.

5. Machine Cycle and T-States (Brief Context)

Each instruction execution typically involves one or more 'Machine Cycles'. A Machine Cycle is the time required to complete one operation like fetching an opcode, reading data, or writing data to memory/IO. Each Machine Cycle is further broken down into smaller time units called 'T-States'. A T-State is one clock period of the microprocessor. The 8085 uses T-states to precisely time each step within a machine cycle.

For instance, an Opcode Fetch machine cycle in 8085 typically takes 4 T-states. Data Read/Write machine cycles typically take 3 T-states.

6. Real-World Understanding and Analogy

Think of the entire instruction execution process as a chef following a recipe:

• Fetch: The chef looks at the recipe book to find the next step (e.g., **Add 2 cups of flour**). This is like the CPU fetching an instruction from memory.
• Decode: The chef reads and understands what **Add 2 cups of flour** means and identifies the ingredients and tools needed (e.g., flour, measuring cup). This is the CPU decoding the instruction.
• Execute: The chef actually takes the flour, measures 2 cups, and adds it to the bowl. This is the CPU performing the operation, like an ALU calculation or data transfer.

The Control Unit is like the chef's brain, coordinating all these actions. The Program Counter is like a bookmark in the recipe book, always pointing to the next step.

Summary of Key Points:

• Instruction execution in 8085 is a repetitive Fetch-Decode-Execute cycle.
• Fetch involves retrieving an instruction from memory using the Program Counter and storing it in the Instruction Register.
• Decode involves interpreting the instruction's opcode using the Instruction Decoder to generate control signals.
• Execute is where the actual operation (data transfer, arithmetic, logic, memory access, I/O, program control) is performed based on the decoded instruction and generated control signals.
• The Control Unit orchestrates the entire process, using timing signals and directing data flow.
• This cycle is broken down into Machine Cycles, which are further divided into T-States for precise timing.
• It's the fundamental process enabling the microprocessor to run programs by processing one

instruction at a time.