Module 2: Microprocessor Architectures: 8085 and 8086

This module provides a foundational understanding of two seminal microprocessors: the 8085 and the 8086. We begin with an Introduction to the 8085 Microprocessor, delving into its internal architecture, a detailed examination of its pin diagram, and the role of its various functional blocks. Next, we meticulously explore the 8085 Instruction Set in Part 1, focusing on essential data transfer, arithmetic, and logical instructions, complete with practical examples to solidify comprehension. Part 2 of the 8085 Instruction Set continues this exploration, covering branch, stack, and I/O instructions, again reinforced with illustrative examples. Our journey then transitions to a more advanced processor with an Introduction to the 8086 Microprocessor, where we analyze its architecture, the concept of segmented memory that distinguishes it, and its different operating modes. Finally, we provide an 8086 Instruction Set Overview, highlighting the key differences and significant enhancements it offers over its predecessor, the 8085.

2.1 Introduction to the 8085 Microprocessor: Architecture, Pin Diagram, and Functional Blocks

The Intel 8085, introduced in 1976, is an 8-bit microprocessor. It was widely used in embedded systems and played a crucial role in the development of microcomputer technology. Understanding the 8085 provides a solid basis for comprehending later, more complex microprocessor architectures.

2.1.1 Architecture of the 8085 Microprocessor

The 8085 is an 8-bit CPU, meaning it processes data in 8-bit chunks. It has a 16-bit address bus, allowing it to access 216=65,536 memory locations (64 KB). The internal architecture is composed of several key functional units working in coordination.

Key Components of 8085 Architecture:

1. **Arithmetic and Logic Unit (ALU):**
   - **Performs arithmetic operations (addition, subtraction, increment, decrement) and logical operations (AND, OR, XOR, NOT).**
   - **Results of operations are typically stored in the Accumulator.**
   - **It communicates with the Accumulator and temporary registers.**
2. **Accumulator (A Register):**
   - **An 8-bit register, the most important general-purpose register in the 8085.**
   - **All arithmetic and most logical operations involve the Accumulator as one of the operands and store the result in the Accumulator.**
   - **It is directly connected to the ALU.**
3. **General-Purpose Registers (B, C, D, E, H, L):**

- Six 8-bit registers that can be used to store data temporarily during program execution.
- These registers can also be paired up to form 16-bit register pairs: (B, C), (D, E), and (H, L).
  - The H-L pair is particularly important as it can be used to store a 16-bit memory address, allowing the CPU to access specific memory locations.

4. **Stack Pointer (SP):**
   - A 16-bit register that holds the memory address of the top of the stack.
   - The stack is a portion of RAM used for temporary storage during subroutine calls and interrupt handling. Data is pushed onto and popped from the stack.
5. **Program Counter (PC):**
   - A 16-bit register that stores the memory address of the next instruction to be fetched and executed.
   - It is automatically incremented after each instruction fetch.
   - Branch instructions (jumps, calls, returns) modify the PC to alter the program flow.
6. **Flag Register (Status Register):**
   - An 8-bit register containing five 1-bit flags that indicate the status of the most recent arithmetic or logical operation performed by the ALU.
   - S (Sign Flag): Set if the result of an operation is negative (MSB is 1). Cleared if positive.
   - Z (Zero Flag): Set if the result of an operation is zero. Cleared if non-zero.
   - AC (Auxiliary Carry Flag): Set if there is a carry from bit 3 to bit 4 during an arithmetic operation. Used in BCD (Binary Coded Decimal) arithmetic.
   - P (Parity Flag): Set if the result has an even number of 1s (even parity). Cleared if an odd number of 1s (odd parity).
   - CY (Carry Flag): Set if there is a carry out of the most significant bit (bit 7) during an addition, or a borrow during a subtraction.
7. **Instruction Register:**
   - An 8-bit register that temporarily stores the opcode (machine code for an instruction) of the instruction currently being fetched from memory.
8. **Instruction Decoder and Machine Cycle Encoding:**
   - Interprets the instruction stored in the Instruction Register and generates control signals for the internal operations of the CPU.
   - Determines the sequence of operations (machine cycles) required to execute the instruction.
9. **Timing and Control Unit:**
   - Generates timing and control signals for all the operations of the microprocessor and its connected peripherals.
   - Receives clock signals from an external crystal oscillator.
   - Includes control signals like RD (Read), WR (Write), ALE (Address Latch Enable), IO/M (I/O or Memory selection), S0, S1 (Status signals), etc.
10. **Address Buffer and Data Buffer:**
    - Address Buffer: Drives the 16-bit address onto the address bus.

    - Data Buffer: Bidirectional buffer for the 8-bit data bus.

2.1.2 Pin Diagram of the 8085 Microprocessor

The 8085 is a 40-pin integrated circuit. Each pin has a specific function, allowing the CPU to communicate with external memory, I/O devices, and other components.

Let's categorize the pins:

- Address Bus (A15-A8): 8 pins (pins 21-28) that carry the higher-order 8 bits of the 16-bit memory address. These are unidirectional (output only from 8085).
- Multiplexed Address/Data Bus (AD7-AD0): 8 pins (pins 12-19) that serve a dual purpose. They carry the lower-order 8 bits of the 16-bit memory address during the first clock cycle of a machine cycle, and then carry the 8-bit data during subsequent clock cycles. These are bidirectional.
    - The ALE (Address Latch Enable) signal (pin 30) is used to demultiplex the AD7-AD0 lines. When ALE is HIGH, AD7-AD0 act as address lines. When ALE is LOW, they act as data lines.
- Control and Status Signals:
    - ALE (Address Latch Enable): (pin 30) Output. Indicates that the AD7-AD0 lines contain a valid address.
    - RD (Read): (pin 32) Output, active low. Indicates that the CPU is performing a read operation (fetching data from memory or I/O).
    - WR (Write): (pin 31) Output, active low. Indicates that the CPU is performing a write operation (sending data to memory or I/O).
    - IO/M (I/O/Memory Select): (pin 34) Output. Differentiates between I/O and memory operations.
        - HIGH: I/O operation.
        - LOW: Memory operation.
    - S1, S0 (Status Signals): (pins 33, 29) Output. Provide additional status information about the current machine cycle (e.g., opcode fetch, memory read, I/O write).
        - S1 S0: 00 (HLTA), 01 (WRITE), 10 (READ), 11 (Opcode Fetch)
- Power Supply and Clock Signals:
    - VCC: (pin 40) +5V power supply.
    - VSS: (pin 20) Ground reference.
    - X1, X2: (pins 1, 2) Input. Connections for external crystal or RC network to generate the internal clock signals. The 8085's internal clock frequency is half of the crystal frequency. For example, a 6 MHz crystal generates a 3 MHz internal clock.
    - CLK OUT: (pin 37) Output. Provides a clock signal for synchronizing peripheral devices.
- Interrupt and External Signals:
    - TRAP: (pin 6) Input. Non-maskable interrupt (highest priority). Edge and level triggered. Cannot be disabled by software.
    - RST 7.5, RST 6.5, RST 5.5: (pins 7, 8, 9) Input. Maskable restart interrupts. Vectored interrupts (jump to specific memory locations).

    - INTR (Interrupt Request): (pin 10) Input. General purpose maskable interrupt. Non-vectored, requires external hardware to provide the interrupt vector address.
    - INTA (Interrupt Acknowledge): (pin 11) Output, active low. Acknowledges an INTR request.
    - HOLD: (pin 39) Input. Request from a peripheral (e.g., DMA controller) to take control of the buses.
    - HLDA (Hold Acknowledge): (pin 38) Output. Acknowledges a HOLD request, indicating the CPU has relinquished bus control.
    - READY: (pin 35) Input. Used by slow peripheral devices to tell the CPU to wait until the data is ready.
- Serial I/O Ports:
    - SID (Serial Input Data): (pin 5) Input. Reads serial data.
    - SOD (Serial Output Data): (pin 4) Output. Transmits serial data.
- RESET IN: (pin 36) Input, active low. Resets the CPU. When low, PC is reset to 0000H, interrupt enables are cleared, and internal registers are reset.
- RESET OUT: (pin 3) Output. Indicates that the CPU is being reset. Used to reset other peripheral devices.

Understanding the pin diagram is crucial for interfacing the 8085 with memory, I/O devices, and other components in a microcomputer system.

2.2 8085 Instruction Set (Part 1): Data Transfer, Arithmetic, and Logical Instructions with Examples

The 8085 instruction set is a collection of commands that the microprocessor understands. These instructions are represented by opcodes and operands, which the CPU decodes and executes. The instructions are broadly categorized based on their function.

Instruction Format: `MNEMONIC Operand1, Operand2` (Operand order can vary).

2.2.1 Data Transfer Instructions (Copy Operations): These instructions copy data between registers, between a register and memory, or between an immediate value and a register/memory. They do not affect any flags.

- Register to Register Transfer:
    - `MOV Rd, Rs`: Copy the content of source register `Rs` to destination register `Rd`.
        - `Rd` can be A, B, C, D, E, H, L.
        - `Rs` can be A, B, C, D, E, H, L.
        - *Numerical Example:* If register B contains `35H`, after `MOV A, B`, register A will contain `35H`. (B remains `35H`).
- Memory to Register / Register to Memory Transfer:
    - `MOV R, M`: Copy the content of memory location pointed by H-L pair to register `R`.

    - `MOV M, R`: Copy the content of register `R` to memory location pointed by H-L pair.
        - *Numerical Example:* If H-L pair contains `2000H` and memory location `2000H` contains `AAH`, after `MOV B, M`, register B will contain `AAH`.
- Immediate Data to Register / Memory:
    - `MVI R, Data`: Move immediate 8-bit `Data` to register `R`.
        - *Numerical Example:* `MVI C, 50H`. Register C will contain `50H`.
    - `MVI M, Data`: Move immediate 8-bit `Data` to memory location pointed by H-L pair.
        - *Numerical Example:* If H-L pair contains `3000H`, after `MVI M, 7BH`, memory location `3000H` will contain `7BH`.
- Load/Store Accumulator Direct:
    - `LDA Adr`: Load Accumulator with the content of the memory location specified by the 16-bit address `Adr`.
        - *Numerical Example:* If memory location `1000H` contains `F0H`, after `LDA 1000H`, Accumulator A will contain `F0H`.
    - `STA Adr`: Store content of Accumulator into the memory location specified by the 16-bit address `Adr`.
- Load/Store H-L Pair Direct:
    - `LHLD Adr`: Load H-L pair with content of `Adr` and `Adr+1`. The content of `Adr` goes to L, and `Adr+1` goes to H.
        - *Numerical Example:* If memory location `2000H` contains `12H` and `2001H` contains `34H`, after `LHLD 2000H`, L will be `12H` and H will be `34H`. So HL pair will be `3412H`.
    - `SHLD Adr`: Store H-L pair content into `Adr` and `Adr+1`. Content of L goes to `Adr`, and H goes to `Adr+1`.
- Load Register Pair Immediate:
    - `LXI Rp, Data16`: Load register pair `Rp` (BC, DE, HL, SP) with immediate 16-bit `Data16`.
        - *Numerical Example:* `LXI H, 5000H`. H will be `50H`, L will be `00H`.
- Exchange H-L with D-E:
    - `XCHG`: Exchange the contents of the H-L register pair with the D-E register pair.
        - *Numerical Example:* If HL=`1234H` and DE=`5678H`, after `XCHG`, HL=`5678H` and DE=`1234H`.
- Exchange H-L with Top of Stack:
    - `XTHL`: Exchange the contents of the H-L register pair with the two bytes at the top of the stack. L is exchanged with (SP), H is exchanged with (SP+1).
- Load Accumulator Indirect:
    - `LDAX B/D`: Load Accumulator with the content of memory location addressed by the BC or DE register pair.

- Store Accumulator Indirect:
    - `STAX B/D`: Store content of Accumulator into the memory location addressed by the BC or DE register pair.

2.2.2 Arithmetic Instructions: These instructions perform addition, subtraction, increment, and decrement operations. They affect the flags (S, Z, AC, P, CY) based on the result.

- Addition:
    - `ADD R`: Add content of register `R` to Accumulator. Result in A.
        - *Numerical Example:* If A=`10H`, B=`05H`, after `ADD B`, A will be `15H`. Flags: Z=0, P=1 (even parity for 15H=00010101B), CY=0, AC=0, S=0.
    - `ADD M`: Add content of memory pointed by H-L to Accumulator. Result in A.
    - `ADI Data`: Add immediate 8-bit `Data` to Accumulator. Result in A.
    - `ADC R`: Add content of register `R` to Accumulator with Carry flag. (A = A + R + CY).
    - `ADC M`: Add content of memory pointed by H-L to Accumulator with Carry flag.
    - `ACI Data`: Add immediate 8-bit `Data` to Accumulator with Carry flag.
- Subtraction:
    - `SUB R`: Subtract content of register `R` from Accumulator. Result in A. (A = A - R).
        - *Numerical Example:* If A=`10H`, B=`05H`, after `SUB B`, A will be `0BH`. Flags: Z=0, P=1 (even parity for 0BH=00001011B), CY=0 (no borrow), AC=0, S=0.
    - `SUB M`: Subtract content of memory pointed by H-L from Accumulator.
    - `SUI Data`: Subtract immediate 8-bit `Data` from Accumulator.
    - `SBB R`: Subtract content of register `R` from Accumulator with Borrow (Carry flag). (A = A - R - CY).
    - `SBB M`: Subtract content of memory pointed by H-L from Accumulator with Borrow.
    - `SBI Data`: Subtract immediate 8-bit `Data` from Accumulator with Borrow.
- Increment/Decrement:
    - `INR R`: Increment content of register `R` by 1. (Affects all flags except CY).
        - *Numerical Example:* If B=`0FH`, after `INR B`, B will be `10H`.
    - `DCR R`: Decrement content of register `R` by 1. (Affects all flags except CY).
    - `INR M`: Increment content of memory pointed by H-L by 1. (Affects all flags except CY).
    - `DCR M`: Decrement content of memory pointed by H-L by 1. (Affects all flags except CY).

  - `INX Rp`: Increment content of register pair `Rp` (BC, DE, HL) by 1. (Does NOT affect any flags).
    - *Numerical Example:* If HL=`1FFFH`, after `INX H`, HL will be `2000H`.
  - `DCX Rp`: Decrement content of register pair `Rp` by 1. (Does NOT affect any flags).
- Double Addition:
  - `DAD Rp`: Add content of register pair `Rp` (BC, DE, HL) to H-L pair. Result in H-L. (Only affects CY flag).
    - *Numerical Example:* If HL=`1000H`, DE=`2000H`, after `DAD D`, HL will be `3000H`.
- Decimal Adjust Accumulator (DAA):
  - `DAA`: Adjusts the content of the Accumulator to form a two-digit BCD number after an addition operation. It uses the Carry and Auxiliary Carry flags.
    - *Numerical Example:* If A=`18H` and we add `23H` (BCD numbers) without DAA: `ADD A, #23H` -> A=`3BH`. After `DAA`, A will be `41H` (because `18+23 = 41` decimal).

2.2.3 Logical Instructions: These instructions perform bitwise logical operations (AND, OR, XOR, NOT) on operands. They affect S, Z, P, CY, AC flags. CY is always reset to 0; AC is affected according to the operation.

- Logical AND:
  - `ANA R`: Logical AND content of register `R` with Accumulator. Result in A.
    - *Numerical Example:* A=`0F0H` (11110000B), B=`0F0H` (00001111B). After `ANA B`, A will be `00H`. CY=0.
  - `ANA M`: Logical AND content of memory pointed by H-L with Accumulator.
  - `ANI Data`: Logical AND immediate 8-bit `Data` with Accumulator.
- Logical OR:
  - `ORA R`: Logical OR content of register `R` with Accumulator. Result in A.
    - *Numerical Example:* A=`0F0H` (11110000B), B=`0F0H` (00001111B). After `ORA B`, A will be `0FFH`. CY=0.
  - `ORA M`: Logical OR content of memory pointed by H-L with Accumulator.
  - `ORI Data`: Logical OR immediate 8-bit `Data` with Accumulator.
- Logical XOR:
  - `XRA R`: Logical XOR content of register `R` with Accumulator. Result in A.
    - *Numerical Example:* A=`0F0H` (11110000B), B=`0F0H` (00001111B). After `XRA B`, A will be `0F0H` XOR `00001111B` = `11111111B` (`FFH`). CY=0.
  - `XRA M`: Logical XOR content of memory pointed by H-L with Accumulator.
  - `XRI Data`: Logical XOR immediate 8-bit `Data` with Accumulator.
- Compare:

- `CMP R`: Compare content of register `R` with Accumulator. (A - R) for flags, A remains unchanged.
    - *If A < R:* CY=1.
    - *If A = R:* Z=1, CY=0.
    - *If A > R:* CY=0.
- `CMP M`: Compare content of memory pointed by H-L with Accumulator.
- `CPI Data`: Compare immediate 8-bit `Data` with Accumulator.

- Rotate Accumulator:
    - `RLC`: Rotate Accumulator Left (Bit 7 moves to Bit 0 and also to CY).
        - *Numerical Example:* If A=`12H` (00010010B), CY=0. After `RLC`, A=`24H` (00100100B), CY=0.
        - If A=`81H` (10000001B), CY=0. After `RLC`, A=`03H` (00000011B), CY=1.
    - `RRC`: Rotate Accumulator Right (Bit 0 moves to Bit 7 and also to CY).
    - `RAL`: Rotate Accumulator Left through Carry (Bit 7 moves to CY, CY moves to Bit 0).
    - `RAR`: Rotate Accumulator Right through Carry (Bit 0 moves to CY, CY moves to Bit 7).
- Complement/Set/Clear:
    - `CMA`: Complement Accumulator (bitwise NOT A).
    - `CMC`: Complement Carry flag.
    - `STC`: Set Carry flag.

This first part of the instruction set covers the fundamental operations needed to manipulate data within the 8085.

2.3 8085 Instruction Set (Part 2): Branch, Stack, and I/O Instructions with Examples

Building upon the data manipulation capabilities, the 8085 instruction set also includes powerful instructions for controlling program flow, managing the stack, and interacting with external input/output devices.

2.3.1 Branch Instructions (Program Control): These instructions alter the sequential flow of program execution by changing the content of the Program Counter (PC). They can be conditional or unconditional.

- Unconditional Jumps:
    - `JMP Adr`: Jump unconditionally to the 16-bit `Adr`. (Loads `Adr` into PC).
        - *Numerical Example:* `JMP START`. Program execution immediately transfers to the instruction labeled `START`.
- Conditional Jumps: These instructions check the status of a specific flag and jump only if the condition is met. If not, execution continues sequentially.
    - `JC Adr`: Jump if Carry Flag is Set (CY=1).
    - `JNC Adr`: Jump if No Carry Flag (CY=0).

    - `JZ Adr`: Jump if Zero Flag is Set (Z=1).
    - `JNZ Adr`: Jump if No Zero Flag (Z=0).
    - `JP Adr`: Jump if Plus (Sign Flag S=0).
    - `JM Adr`: Jump if Minus (Sign Flag S=1).
    - `JPE Adr`: Jump if Parity Even (P=1).
    - `JPO Adr`: Jump if Parity Odd (P=0).
- Call Instructions: Used to transfer control to a subroutine. The return address (address of the instruction after the CALL) is saved on the stack.
    - `CALL Adr`: Unconditionally call subroutine at 16-bit `Adr`. (Pushes PC onto stack, then loads `Adr` into PC).
    - `CC Adr`: Call if Carry (CY=1).
    - `CNC Adr`: Call if No Carry (CY=0).
    - `CZ Adr`: Call if Zero (Z=1).
    - `CNZ Adr`: Call if No Zero (Z=0).
    - `CP Adr`: Call if Plus (S=0).
    - `CM Adr`: Call if Minus (S=1).
    - `CPE Adr`: Call if Parity Even (P=1).
    - `CPO Adr`: Call if Parity Odd (P=0).
- Return Instructions: Used to return from a subroutine. The return address is retrieved from the stack and loaded into PC.
    - `RET`: Unconditionally return from subroutine. (Pops PC from stack).
    - `RC`: Return if Carry (CY=1).
    - `RNC`: Return if No Carry (CY=0).
    - `RZ`: Return if Zero (Z=1).
    - `RNZ`: Return if No Zero (Z=0).
    - `RP`: Return if Plus (S=0).
    - `RM`: Return if Minus (S=1).
    - `RPE`: Return if Parity Even (P=1).
    - `RPO`: Return if Parity Odd (P=0).
- Restart Instructions (RST): Special single-byte call instructions primarily used for interrupts. They implicitly call a fixed memory location based on the RST number.
    - `RST n` (n = 0 to 7): Calls subroutine at fixed vector address (n×8).
        - `RST 0`: Calls 0000H (typically used for reset)
        - `RST 1`: Calls 0008H
        - ...
        - `RST 7`: Calls 0038H

2.3.2 Stack Operations: The stack is a Last-In, First-Out (LIFO) data structure implemented in a portion of the main memory. The Stack Pointer (SP) register always points to the top of the stack. In 8085, the stack grows downwards (towards lower memory addresses).

- Push onto Stack:

    - `PUSH Rp`: Push the content of register pair `Rp` (BC, DE, HL, PSW) onto the stack.
        - The high-order byte of `Rp` is pushed first onto `(SP-1)`, then the low-order byte onto `(SP-2)`.
        - SP is decremented by 2.
        - *Numerical Example:* If SP=`2000H` and HL=`1234H`, after `PUSH H`:
            - Memory location `1FFFH` gets `12H` (H-reg).
            - Memory location `1FFEH` gets `34H` (L-reg).
            - SP becomes `1FFEH`.
        - `PUSH PSW`: Pushes Accumulator (A) and Flag Register (PSW) onto the stack. A is pushed first (to `SP-1`), then Flags (to `SP-2`).
- Pop from Stack:
    - `POP Rp`: Pop two bytes from the stack into register pair `Rp`.
        - The content of `(SP)` is popped into the low-order byte of `Rp`, and `(SP+1)` into the high-order byte.
        - SP is incremented by 2.
        - *Numerical Example:* If SP=`1FFEH`, memory `1FFE` has `34H`, `1FFFH` has `12H`. After `POP H`:
            - L becomes `34H`.
            - H becomes `12H`.
            - SP becomes `2000H`.
        - `POP PSW`: Pops Flags (PSW) from `(SP)` and Accumulator (A) from `(SP+1)`.

2.3.3 I/O Instructions: These instructions are used to transfer data between the Accumulator and an I/O port. The 8085 uses memory-mapped I/O or I/O-mapped I/O. It supports 256 unique I/O port addresses (from `00H` to `FFH`).

- Input from Port:
    - `IN Port_Adr`: Read an 8-bit byte from the specified 8-bit `Port_Adr` and store it in the Accumulator.
        - *Numerical Example:* `IN 05H`. Reads data from I/O port address `05H` into Accumulator A.
- Output to Port:
    - `OUT Port_Adr`: Send the content of the Accumulator to the specified 8-bit `Port_Adr`.
        - *Numerical Example:* `OUT 0AH`. Sends data from Accumulator A to I/O port address `0AH`.

2.3.4 Control Instructions: These instructions control the CPU's state.

- `EI`: Enable Interrupts. Sets the Interrupt Enable Flip-Flop, allowing maskable interrupts to be processed.

- `DI`: Disable Interrupts. Resets the Interrupt Enable Flip-Flop, preventing maskable interrupts from being processed.
- `HLT`: Halt. Stops CPU execution until an interrupt occurs or the system is reset.
- `NOP`: No Operation. Does nothing. Used for timing delays or as placeholders.
- `RIM`: Read Interrupt Mask. Reads the current status of interrupts (mask bits, pending interrupts) into the Accumulator.
- `SIM`: Set Interrupt Mask. Writes the content of the Accumulator to set interrupt masks, set pending interrupts, and control serial output data (SOD).

The combination of data transfer, arithmetic, logical, branch, stack, and I/O instructions provides the 8085 with the versatility to perform a wide range of tasks in various applications.

2.4 Introduction to the 8086 Microprocessor: Architecture, Segmented Memory, and Operating Modes

The Intel 8086, introduced in 1978, was a significant leap forward from the 8085. It is a 16-bit microprocessor with a 20-bit address bus, capable of accessing 220=1,048,576 memory locations (1 MB). The 8086 laid the groundwork for the x86 architecture, which dominates personal computing today.

2.4.1 Architecture of the 8086 Microprocessor

The 8086 architecture is divided into two distinct functional units to achieve pipelining and improve performance:

1. Bus Interface Unit (BIU):
   - Handles all external bus operations: fetching instructions, reading/writing data, and I/O operations.
   - Instruction Queue (6 bytes): Prefetches up to 6 bytes of instruction code from memory and stores them in a FIFO (First-In, First-Out) queue. This helps in pipelining, as the EU can execute instructions while the BIU is fetching the next ones.
   - Segment Registers (CS, DS, SS, ES): Four 16-bit registers used for memory segmentation (explained below).
   - Instruction Pointer (IP): A 16-bit register that stores the offset address of the next instruction within the current code segment. Similar to the PC in 8085, but works with segments.
   - Address Generation Unit: Calculates the 20-bit physical address by combining a segment register value (shifted left by 4 bits) with an offset address.
   - Bus Control Logic: Generates timing and control signals for bus operations (read, write, interrupt acknowledge, etc.).
2. Execution Unit (EU):
   - Responsible for executing instructions. It receives instructions from the BIU's instruction queue.

- Arithmetic and Logic Unit (ALU): Performs 16-bit arithmetic and logical operations.
- General Purpose Registers (AX, BX, CX, DX): Four 16-bit general-purpose registers. Each can be accessed as a 16-bit register or as two separate 8-bit registers (e.g., AX as AH and AL).
  - AX (Accumulator): Primary register for arithmetic and logical operations.
  - BX (Base Register): Can be used as a base address for memory access.
  - CX (Count Register): Used as a loop counter in `LOOP` instructions and for string operations.
  - DX (Data Register): Used for I/O operations (port address) and for multiplication/division with AX.
- Pointer and Index Registers (SP, BP, SI, DI):
  - SP (Stack Pointer): A 16-bit register used with the SS register to point to the top of the stack.
  - BP (Base Pointer): A 16-bit register used with the SS register for accessing data on the stack.
  - SI (Source Index): A 16-bit register used with the DS register as an offset for string and source data operations.
  - DI (Destination Index): A 16-bit register used with the ES register as an offset for string and destination data operations.
- Flag Register (Status Register): A 16-bit register (though only 9 bits are actively used) that indicates the status of operations and controls CPU features.
  - Conditional Flags (similar to 8085):
    - CF (Carry Flag): Set if carry/borrow out of MSB.
    - PF (Parity Flag): Set if even parity.
    - AF (Auxiliary Carry Flag): Set if carry/borrow between nibbles.
    - ZF (Zero Flag): Set if result is zero.
    - SF (Sign Flag): Set if result is negative (MSB is 1).
    - OF (Overflow Flag): Set if signed arithmetic result overflows.
  - Control Flags:
    - TF (Trap Flag/Single Step): If set, CPU executes one instruction at a time, used for debugging.
    - IF (Interrupt Enable Flag): If set, enables maskable interrupts.
    - DF (Direction Flag): If set, string operations decrement index registers (right to left). If cleared, increment (left to right).

2.4.2 Segmented Memory Architecture

The 8086 has a 20-bit physical address bus, allowing it to access 1 MB of memory. However, its internal registers are only 16-bit. To access 1 MB using 16-bit registers, the 8086 employs a segmented memory architecture. Memory is divided into logical segments, and a physical address is generated by combining a 16-bit segment address with a 16-bit offset address.

Segment Registers: Each segment register (16-bit) points to the starting address of a 64 KB segment within the 1 MB address space.

- CS (Code Segment Register): Holds the base address of the segment containing the program instructions. The Instruction Pointer (IP) provides the offset within this segment.
- DS (Data Segment Register): Holds the base address of the segment containing most of the program's data.
- SS (Stack Segment Register): Holds the base address of the segment containing the program's stack. The Stack Pointer (SP) and Base Pointer (BP) provide offsets within this segment.
- ES (Extra Segment Register): An additional data segment register, often used for string operations or accessing other data areas.

Physical Address Calculation: The 20-bit physical address (PA) is calculated by the BIU using the following formula:

```
Physical Address = (Segment Register Value * 16) + Offset Address
```

Or, equivalently:

```
Physical Address = (Segment Register Value << 4) + Offset Address
```

Where `<< 4` means left-shifting the segment register value by 4 bits, which is equivalent to multiplying by 24=16. This shift effectively adds four zeros to the right of the 16-bit segment value, turning it into a 20-bit base address.

- *Numerical Example:*
  - If CS = `2000H` (Code Segment value)
  - If IP = `1234H` (Instruction Pointer offset)
  - Physical Address for next instruction = (`2000H` shifted left by 4) + `1234H`
  - `2000H` shifted left by 4 bits = `20000H`
  - `Physical Address = 20000H + 1234H = 21234H`

This mechanism allows programs to be larger than 64 KB and enables memory protection and multitasking.

2.4.3 Operating Modes of the 8086

The 8086 microprocessor can operate in two different modes:

1. Minimum Mode:
   - Used in single-processor systems where the 8086 is the only processor.
   - The 8086 directly generates all bus control signals (ALE, RD, WR, IO/M, etc.).

    - The `MN/MX` (Minimum/Maximum Mode) pin (pin 33) is held HIGH (connected to VCC) to select this mode.
    - Suitable for small-scale applications.
2. Maximum Mode:
    - Used in multiprocessor systems or systems that require a coprocessor (like the 8087 numeric coprocessor).
    - The 8086 works with an external bus controller chip (e.g., Intel 8288 Bus Controller).
    - The `MN/MX` pin is held LOW (connected to Ground) to select this mode.
    - The 8086 outputs status signals (S2, S1, S0) that the 8288 decodes to generate the necessary bus control signals.
    - Supports features like bus arbitration, which allows multiple processors to share the system bus.

The 8086's architecture, particularly its segmented memory and pipelining, represented a significant advancement, paving the way for more powerful and complex computing systems.

2.5 8086 Instruction Set Overview: Key Differences and Enhancements over 8085

The 8086 instruction set is a superset of the 8085's, meaning it includes most of the 8085's functionalities and introduces many new, powerful instructions. The most significant enhancements stem from its 16-bit architecture, segmented memory addressing, and new addressing modes.

Key Differences and Enhancements:

1. 16-bit Operations:
    - The 8086 naturally operates on 16-bit data. Most instructions can take 8-bit or 16-bit operands, indicated by the instruction or a prefix.
    - 8085: Primarily 8-bit operations. 16-bit operations required multiple 8-bit instructions (e.g., adding two 16-bit numbers).
    - 8086 Example: `ADD AX, BX` (Adds 16-bit content of BX to AX). `ADD AL, BL` (Adds 8-bit content of BL to AL).
2. Expanded General Purpose Registers:
    - 8085: Limited to A, B, C, D, E, H, L (8-bit registers), with BC, DE, HL as 16-bit pairs.
    - 8086: Four general-purpose 16-bit registers (AX, BX, CX, DX), each divisible into two 8-bit registers (AH/AL, BH/BL, CH/CL, DH/DL). This provides more flexibility and simplifies 8-bit operations.
3. Sophisticated Addressing Modes:
    - The 8086 introduces more complex and flexible addressing modes for accessing memory. This allows for more efficient data access, especially for arrays and data structures.
    - 8085: Direct, Register, Register Indirect (HL pair only), Immediate.
    - 8086 Examples of New Modes:

        - Register Relative: `MOV AX, [BX + 04H]` (Access memory location by adding 4 to content of BX).
        - Based Indexed: `MOV AL, [BX + SI]` (Access memory using base register BX and index register SI).
        - Based Indexed with Displacement: `MOV CL, [BP + DI + 20H]` (Combines base, index, and a constant offset).
        - These modes are crucial for C language arrays and pointers.
4. Segmented Memory Addressing:
    - The most fundamental difference from 8085. The 8086 uses 20-bit physical addresses derived from a 16-bit segment and a 16-bit offset.
    - 8085: Flat 64 KB address space. All 16-bit addresses directly point to memory.
    - 8086: 1 MB segmented address space. Requires understanding of segment registers (CS, DS, SS, ES) and offset registers (IP, SP, BP, SI, DI, BX).
5. String Manipulation Instructions:
    - The 8086 has dedicated instructions for efficient string operations (copying, comparing, scanning, loading, storing strings). These instructions automatically handle incrementing/decrementing source/destination pointers based on the Direction Flag (DF).
    - 8085: String operations had to be implemented using loops and manual pointer manipulation.
    - 8086 Examples: `MOVSB` (Move String Byte), `CMPSW` (Compare String Word), `SCASB` (Scan String Byte). These are very powerful when combined with `REP` (Repeat) prefix.
6. I/O Instructions (Port Addresses):
    - 8085: `IN/OUT` instructions use an 8-bit port address.
    - 8086: `IN/OUT` instructions can use an 8-bit port address (direct addressing, 0-255) or a 16-bit port address stored in DX (indirect addressing, 0-65535).
    - Example: `IN AL, 20H` (read byte from port 20H to AL). `MOV DX, 3FFH; IN AX, DX` (read word from port 3FFH to AX).
7. Loop Instructions:
    - The 8086 includes dedicated loop instructions that use the CX register as a counter, simplifying loop implementation.
    - 8085: Loops usually involved `DCR` and `JNZ`.
    - 8086 Examples: `LOOP label`, `LOOPE/LOOPZ label` (Loop while equal/zero), `LOOPNE/LOOPNZ label` (Loop while not equal/not zero).
8. Processor Control Instructions:
    - 8086: More extensive control over CPU flags and state, including `STC`, `CLC`, `CMC` (for Carry), `STD`, `CLD` (for Direction Flag), `STI`, `CLI` (for Interrupt Flag).
    - 8085: Fewer dedicated flag manipulation instructions.
9. Bit Manipulation Instructions:

- While not as extensive as later processors, the 8086 has some bit-level test instructions.
- Example: `TEST AX, 0001H` (Performs a logical AND without storing result, sets flags for conditional jumps based on specific bits).

10. Multiplication and Division:
    - The 8086 provides dedicated instructions for signed and unsigned multiplication and division of both 8-bit and 16-bit numbers.
    - 8085: Multiplication and division required complex software routines.
    - 8086 Examples:
        - `MUL BL`: Multiplies AL by BL, 16-bit result in AX.
        - `IMUL BX`: Signed multiply AX by BX, 32-bit result in DX:AX.
        - `DIV BL`: Divides AX by BL, quotient in AL, remainder in AH.
        - `IDIV BX`: Signed divide DX:AX by BX, quotient in AX, remainder in DX.
11. Instruction Queue/Pipelining:
    - The BIU's instruction queue prefetching is a key architectural enhancement for performance that the 8085 lacked.

In summary, the 8086 moved beyond the simpler 8-bit architecture of the 8085 by introducing 16-bit processing, segmented memory, a more robust set of registers, and specialized instructions for tasks like string manipulation and complex addressing. These advancements enabled the 8086 to handle more sophisticated software and larger memory spaces, making it a pivotal processor in computing history.