Assembly Language Programming of 8085
Topics

1. Introduction
2. Programming model of 8085
3. Instruction set of 8085
4. Example Programs
5. Addressing modes of 8085
6. Instruction & Data Formats of 8085
1. Introduction

- A **microprocessor** executes instructions given by the user.
- Instructions should be in a language known to the **microprocessor**.
- **Microprocessor** understands the language of 0’s and 1’s only.
- This language is called **Machine Language**.
• For e.g.
  01001111
  – Is a valid machine language instruction of **8085**
  – It copies the contents of one of the internal registers of **8085** to another
A Machine language program to add two numbers

00111110 ;Copy value 2H in register A
00000010
00000110 ;Copy value 4H in register B
00000100
10000000 ;A = A + B
Assembly Language of 8085

- It uses English like words to convey the action/meaning called as MNEMONICS
- For e.g.
  - MOV to indicate data transfer
  - ADD to add two values
  - SUB to subtract two values
Assembly language program to add two numbers

MVI A, 2H ;Copy value 2H in register A
MVI B, 4H ;Copy value 4H in register B
ADD B ;A = A + B

Note:

• Assembly language is specific to a given processor
• For e.g. assembly language of 8085 is different than that of Motorola 6800 microprocessor
Microprocessor understands Machine Language only!

- **Microprocessor** cannot understand a program written in Assembly language.
- A program known as **Assembler** is used to convert an Assembly language program to machine language.
Low-level/High-level languages

• Machine language and Assembly language are both
  – Microprocessor specific (Machine dependent)
    so they are called
  – Low-level languages

• **Machine independent** languages are called
  – High-level languages
  – For e.g. BASIC, PASCAL, C++, C, JAVA, etc.
  – A software called **Compiler** is required to convert a high-level language program to machine code
2. Programming model of 8085

Accumulator
ALU
Flags
Instruction Decoder
Register Array
Memory Pointer Registers
Timing and Control Unit

16-bit Address Bus
8-bit Data Bus
Control Bus
<table>
<thead>
<tr>
<th>Accumulator (8-bit)</th>
<th>Flag Register (8-bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B (8-bit)</td>
<td>C (8-bit)</td>
</tr>
<tr>
<td>D (8-bit)</td>
<td>E (8-bit)</td>
</tr>
<tr>
<td>H (8-bit)</td>
<td>L (8-bit)</td>
</tr>
<tr>
<td>Stack Pointer (SP) (16-bit)</td>
<td></td>
</tr>
<tr>
<td>Program Counter (PC) (16-bit)</td>
<td></td>
</tr>
</tbody>
</table>

8- Lines  
Bidirectional  

16- Lines  
Unidirectional
Overview: 8085 Programming model

1. Six general-purpose Registers
2. Accumulator Register
3. Flag Register
4. Program Counter Register
5. Stack Pointer Register
1. **Six general-purpose registers**
   - **B, C, D, E, H, L**
   - Can be combined as register pairs to perform 16-bit operations (BC, DE, HL)

2. **Accumulator – identified by name A**
   - This register is a part of ALU
   - 8-bit data storage
   - Performs arithmetic and logical operations
   - Result of an operation is stored in accumulator
3. Flag Register
   - This is also a part of ALU
   - 8085 has five flags named
     - **Zero** flag (Z)
     - **Carry** flag (CY)
     - **Sign** flag (S)
     - **Parity** flag (P)
     - **Auxiliary Carry** flag (AC)
• These flags are five flip-flops in flag register
• Execution of an arithmetic/logic operation can set or reset these flags
• Condition of flags (set or reset) can be tested through software instructions
• 8085 uses these flags in decision-making process
4. Program Counter (PC)
   – A 16-bit memory pointer register
   – Used to sequence execution of program instructions
   – Stores address of a memory location
     • where next instruction byte is to be fetched by the 8085
   – when 8085 gets busy to fetch current instruction from memory
     • PC is incremented by one
     • PC is now pointing to the address of next instruction
5. Stack Pointer Register
   - a 16-bit memory pointer register
   - Points to a location in **Stack** memory
   - Beginning of the stack is defined by loading a 16-bit address in stack pointer register
3. Instruction Set of 8085

- Consists of
  - 74 operation codes, e.g. MOV
  - 246 Instructions, e.g. MOV A,B
- 8085 instructions can be classified as
  1. Data Transfer (Copy)
  2. Arithmetic
  3. Logical and Bit manipulation
  4. Branch
  5. Machine Control
1. Data Transfer (Copy) Operations

1. **Load** a 8-bit number in a **Register**
2. **Copy** from **Register to Register**
3. **Copy** between **Register and Memory**
4. **Copy** between **Input/Output Port and Accumulator**
5. **Load** a 16-bit number in a **Register pair**
6. **Copy** between **Register pair and Stack memory**
## Example Data Transfer (Copy)

### Operations

| 1. **Load** a 8-bit number 4F in register B |
| 2. **Copy** from Register B to Register A |
| 3. **Load** a 16-bit number 2050 in Register pair HL |
| 4. **Copy** from Register B to Memory Address 2050 |
| 5. **Copy** between Input/Output Port and Accumulator |

### Instructions

| 1. MVI B, 4FH |
| 2. MOV A,B |
| 3. LXI H, 2050H |
| 4. MOV M,B |
| 5. OUT 01H |
| 6. IN 07H |
2. Arithmetic Operations

1. **Addition** of two 8-bit numbers
2. **Subtraction** of two 8-bit numbers
3. **Increment/ Decrement** a 8-bit number
Example Arithmetic Operations / Instructions

1. Add a 8-bit number 32H to Accumulator  
   **ADI 32H**

2. Add contents of Register B to Accumulator  
   **ADD B**

3. Subtract a 8-bit number 32H from Accumulator  
   **SUI 32H**

4. Subtract contents of Register C from Accumulator  
   **SUB C**

5. Increment the contents of Register D by 1  
   **INR D**

6. Decrement the contents of Register E by 1  
   **DCR E**
3. Logical & Bit Manipulation Operations

1. **AND** two 8-bit numbers
2. **OR** two 8-bit numbers
3. **Exclusive-OR** two 8-bit numbers
4. **Compare** two 8-bit numbers
5. **Complement**
6. **Rotate** Left/Right Accumulator bits
<table>
<thead>
<tr>
<th>Example Logical &amp; Bit Manipulation Operations</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Logically <strong>AND</strong> Register H with Accumulator</td>
<td>ANA H</td>
</tr>
<tr>
<td>2. Logically <strong>OR</strong> Register L with Accumulator</td>
<td>ORA L</td>
</tr>
<tr>
<td>3. Logically <strong>XOR</strong> Register B with Accumulator</td>
<td>XRA B</td>
</tr>
<tr>
<td>4. <strong>Compare</strong> contents of Register C with Accumulator</td>
<td>CMP C</td>
</tr>
<tr>
<td>5. <strong>Complement</strong> Accumulator</td>
<td>CMA</td>
</tr>
<tr>
<td>6. <strong>Rotate</strong> Accumulator Left</td>
<td>RAL</td>
</tr>
</tbody>
</table>
4. Branching Operations

These operations are used to control the flow of program execution

1. Jumps
   - Conditional jumps
   - Unconditional jumps

2. Call & Return
   - Conditional Call & Return
   - Unconditional Call & Return
<table>
<thead>
<tr>
<th>Example Branching Operations</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Jump to a 16-bit Address 2080H if Carry flag is SET</td>
<td>JC 2080H</td>
</tr>
<tr>
<td>2. Unconditional Jump</td>
<td>JMP 2050H</td>
</tr>
<tr>
<td>3. Call a subroutine with its 16-bit Address</td>
<td>CALL 3050H</td>
</tr>
<tr>
<td>4. Return back from the Call</td>
<td>RET</td>
</tr>
<tr>
<td>5. Call a subroutine with its 16-bit Address if Carry flag is RESET</td>
<td>CNC 3050H</td>
</tr>
<tr>
<td>6. Return if Zero flag is SET</td>
<td>RZ</td>
</tr>
</tbody>
</table>
These instructions affect the operation of the processor. For e.g.

- **HLT**: Stop program execution
- **NOP**: Do not perform any operation
4. Writing a Assembly Language Program

• Steps to write a program
  – Analyze the problem
  – Develop program Logic
  – Write an Algorithm
  – Make a Flowchart
  – Write program Instructions using Assembly language of 8085
Program **8085** in Assembly language to add two 8-bit numbers and store 8-bit result in register **C**.

1. **Analyze the problem**
   - Addition of two 8-bit numbers to be done

2. **Program Logic**
   - Add two numbers
   - Store result in register **C**
   - Example
     
     \[
     \begin{align*}
     \text{10011001} & \quad (99H) \quad A \\
     +00111001 & \quad (39H) \quad D \\
     \text{11010010} & \quad (D2H) \quad C
     \end{align*}
     \]
3. **Algorithm**

1. Get two numbers
2. Add them
3. Store result
4. Stop

**Translation to 8085 operations**

- Load 1\textsuperscript{st} no. in register D
- Load 2\textsuperscript{nd} no. in register E
- Copy register D to A
- Add register E to A
- Copy A to register C
- Stop processing
4. Make a Flowchart

Start

Load Registers D, E

Copy D to A

Add A and E

Copy A to C

Stop

- Load 1\textsuperscript{st} no. in register D
- Load 2\textsuperscript{nd} no. in register E
- Copy register D to A
- Add register E to A
- Copy A to register C
- Stop processing
5. Assembly Language Program

1. Get two numbers
   a) Load 1st no. in register D
   b) Load 2nd no. in register E

2. Add them
   a) Copy register D to A
   b) Add register E to A

3. Store result
   a) Copy A to register C

4. Stop
   a) Stop processing
Program 8085 in Assembly language to add two 8-bit numbers. Result can be more than 8-bits.

1. Analyze the problem

   - Result of addition of two 8-bit numbers can be 9-bit
   - Example
     
     \[ 10011001 \ (99H) \ A \]
     \[ +10011001 \ (99H) \ B \]
     \[ 100110010 \ (132H) \]

   - The 9\textsuperscript{th} bit in the result is called CARRY bit.
• How 8085 does it?
  – Adds register A and B
  – Stores 8-bit result in A
  – SETS carry flag (CY) to indicate carry bit

```
10011001
+ 99H
10011001
+ 99H
10011001
+ 99H
10011001
+ 99H
00110010
32H
1
CY
```
• Storing result in Register memory

Step-1  Copy A to C

Step-2
  a) Clear register B
  b) Increment B by 1
2. **Program Logic**

1. Add two numbers
2. Copy 8-bit result in A to C
3. If CARRY is generated
   - Handle it
4. Result is in register pair BC
3. **Algorithm**

1. Load two numbers in registers D, E
2. Add them
3. Store 8 bit result in C
4. Check CARRY flag
5. If CARRY flag is SET
   - Store CARRY in register B
6. Stop

**Translation to 8085 operations**

- Load registers D, E
- Copy register D to A
- Add register E to A
- Copy A to register C
- Use Conditional Jump instructions
  - Clear register B
  - Increment B
- Stop processing
4. Make a Flowchart

Start

Load Registers D, E

Copy D to A

Add A and E

Copy A to C

If CARRY NOT SET

False

Clear B

True

Increment B

Stop
5. Assembly Language Program

- Load registers D, E
- Copy register D to A
- Add register E to A
- Copy A to register C
- Use Conditional Jump instructions
- Clear register B
- Increment B
- Stop processing

MVI D, 2H
MVI E, 3H
MOV A, D
ADD E
MOV C, A
JNC END
MVI B, 0H
INR B
END: HLT
4. Addressing Modes of 8085

- Format of a typical Assembly language instruction is given below-

  [Label:] Mnemonic [Operands] ;;comments
  HLT
  MVI A, 20H
  MOV M, A ;Copy A to memory location whose address is stored in register pair HL
  LOAD: LDA 2050H ;Load A with contents of memory location with address 2050H
  READ: IN 07H ;Read data from Input port with address 07H
The various formats of specifying operands are called addressing modes.

Addressing modes of 8085:
1. Register Addressing
2. Immediate Addressing
3. Memory Addressing
4. Input/Output Addressing
1. Register Addressing

• Operands are one of the internal registers of 8085
• Examples-
  MOV A, B
  ADD C
2. Immediate Addressing

- Value of the operand is given in the instruction itself
- Example:
  - MVI A, 20H
  - LXI H, 2050H
  - ADI 30H
  - SUI 10H
3. Memory Addressing

- One of the operands is a memory location.
- Depending on how address of memory location is specified, memory addressing is of two types:
  - Direct addressing
  - Indirect addressing
3(a) **Direct Addressing**

- **16-bit** Address of the memory location is specified in the instruction directly.
- **Examples:**
  - **LDA 2050H** ; load A with contents of memory location with address 2050H
  - **STA 3050H** ; store A with contents of memory location with address 3050H
3(b) **Indirect Addressing**

- A *memory pointer* register is used to store the address of the memory location.

**Example:**

```
MOV M, A  ; copy register A to memory location whose address is stored in register pair HL
```

![Diagram of memory pointer register and memory location]
4. Input/Output Addressing

- 8-bit address of the port is directly specified in the instruction
- Examples:
  - IN 07H
  - OUT 21H
5. Instruction & Data Formats

8085 Instruction set can be classified according to size (in bytes) as

1. 1-byte Instructions
2. 2-byte Instructions
3. 3-byte Instructions
1. One-byte Instructions

- Includes Opcode and Operand in the same byte
- Examples -

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Operand</th>
<th>Binary Code</th>
<th>Hex Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV</td>
<td>C, A</td>
<td>0100 1111</td>
<td>4FH</td>
</tr>
<tr>
<td>ADD</td>
<td>B</td>
<td>1000 0000</td>
<td>80H</td>
</tr>
<tr>
<td>HLT</td>
<td></td>
<td>0111 0110</td>
<td>76H</td>
</tr>
</tbody>
</table>
1. Two-byte Instructions

- First byte specifies Operation Code
- Second byte specifies Operand
- Examples:

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Operand</th>
<th>Binary Code</th>
<th>Hex Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVI</td>
<td>A, 32H</td>
<td>0011 1110, 0011 0010</td>
<td>3EH, 32H</td>
</tr>
<tr>
<td>MVI</td>
<td>B, F2H</td>
<td>0000 0110, 1111 0010</td>
<td>06H, F2H</td>
</tr>
</tbody>
</table>
1. Three-byte Instructions

- First byte specifies Operation Code
- Second & Third byte specifies Operand
- Examples-

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Operand</th>
<th>Binary Code</th>
<th>Hex Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>LXI</td>
<td>H, 2050H</td>
<td>0010 0001, 0101 0000, 0010 0000</td>
<td>21H, 50H, 20H</td>
</tr>
<tr>
<td>LDA</td>
<td>3070H</td>
<td>0011 1010, 0111 0000, 0011 0000</td>
<td>3AH, 70H, 30H</td>
</tr>
</tbody>
</table>
Separate the digits of a hexadecimal numbers and store it in two different locations

- LDA 2200H ; Get the packed BCD number
- ANI F0H ; Mask lower nibble

\[
\begin{align*}
0100 & 0101 & 45 \\
1111 & 0000 & F0 \\
\hline \\
0100 & 0000 & 40
\end{align*}
\]

- RRC
- RRC
- RRC
- RRC ; Adjust higher digit as a lower digit.
- RRC 0000 0100 after 4 rotations
• STA 2300H ; Store the partial result
• LDA 2200H ; Get the original BCD no.
• ANI 0FH ; Mask higher nibble

\[
\begin{array}{ccc}
0100 & 0100 & 45 \\
0000 & 1111 & 0F \\
\hline
0000 & 0100 & 05 \\
\end{array}
\]

• STA 2301H ; Store the result
• HLT ; Terminate program execution
Block data transfer

- MVI C, 0AH ; Initialize counter i.e no. of bytes
  Store the count in Register C, ie ten
- LXI H, 2200H ; Initialize source memory pointer
  Data Starts from 2200 location
- LXI D, 2300H ; Initialize destination memory pointer

BK: MOV A, M ; Get byte from source memory block i.e 2200 to accumulator.
- STAX D ; Store byte in the destination memory block i.e 2300 as stored in D-E pair
Contd.

- INX H ; Increment source memory pointer
- INX D ; Increment destination memory pointer
- DCR C ; Decrement counter to keep track of bytes moved
- JNZ BK ; If counter 0 repeat steps
- HLT ; Terminate program