INTRODUCTION TO 8085 INSTRUCTIONS
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**
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 a 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
- Timing and Control Unit
- Register Array
- Memory Pointer Registers

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

Copying data from a source to destination refers to data transfer function.

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
<table>
<thead>
<tr>
<th>Operations</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. <strong>Load</strong> a 8-bit number 4F in register <strong>B</strong></td>
<td><strong>MVI</strong> B, 4FH</td>
</tr>
<tr>
<td>2. <strong>Copy</strong> from Register <strong>B</strong> to Register <strong>A</strong></td>
<td><strong>MOV</strong> A,B</td>
</tr>
<tr>
<td>3. <strong>Load</strong> a 16-bit number 2050 in Register pair <strong>HL</strong></td>
<td><strong>LXI</strong> H, 2050H</td>
</tr>
<tr>
<td>4. <strong>Copy</strong> from Register <strong>B</strong> to Memory Address 2050</td>
<td><strong>MOV</strong> M,B</td>
</tr>
<tr>
<td>5. <strong>Copy</strong> between Input/Output Port and Accumulator</td>
<td><strong>OUT</strong> 01H</td>
</tr>
<tr>
<td></td>
<td><strong>IN</strong> 07H</td>
</tr>
</tbody>
</table>
Data Transfer (Copy) Operations

6. 1 byte instruction. Processor stops executing and enters wait state.

7. 1 byte instruction. No operation. Generally used to increase processing time or substitute in place of instruction.

HLT

NOP
2. Arithmetic Operations

1. **Addition** of two 8-bit numbers
2. **Subtraction** of two 8-bit numbers
3. **Increment/ Decrement** a 8-bit number
<table>
<thead>
<tr>
<th>Operations</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Add a 8-bit number 32H to Accumulator</td>
<td>ADI 32H</td>
</tr>
<tr>
<td>2. Add contents of Register B to Accumulator</td>
<td>ADD B</td>
</tr>
<tr>
<td>3. Subtract a 8-bit number 32H from Accumulator</td>
<td>SUI 32H</td>
</tr>
<tr>
<td>4. Subtract contents of Register C from Accumulator</td>
<td>SUB C</td>
</tr>
<tr>
<td>5. Increment the contents of Register D by 1</td>
<td>INR D</td>
</tr>
<tr>
<td>6. Decrement the contents of Register E by 1</td>
<td>DCR E</td>
</tr>
</tbody>
</table>
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>Operations</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logically AND Register H with Accumulator</td>
<td>ANA H</td>
</tr>
<tr>
<td>Logically OR Register L with Accumulator</td>
<td>ORA L</td>
</tr>
<tr>
<td>Logically XOR Register B with Accumulator</td>
<td>XRA B</td>
</tr>
<tr>
<td>Compare contents of Register C with Accumulator</td>
<td>CMP C</td>
</tr>
<tr>
<td>Complement Accumulator</td>
<td>CMA</td>
</tr>
<tr>
<td>Rotate 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. <strong>Jump</strong> to a 16-bit Address 2080H if Carry flag is SET. This is conditional jump. <strong>JNC, JZ, JNZ, JP, JM, JPE, JPO</strong></td>
<td><strong>JC 2080H</strong></td>
</tr>
<tr>
<td>2. <strong>Unconditional Jump</strong></td>
<td><strong>JMP 2050H</strong></td>
</tr>
<tr>
<td>3 byte instruction. 2(^{nd}) and 3(^{rd}) byte specify 16 bit memory address.</td>
<td></td>
</tr>
<tr>
<td>2. <strong>Call</strong> a subroutine with its 16-bit Address</td>
<td><strong>CALL 3050H</strong></td>
</tr>
<tr>
<td>4. <strong>Return back</strong> from the Call</td>
<td><strong>RET</strong></td>
</tr>
<tr>
<td>5. <strong>Call</strong> a subroutine with its 16-bit Address if Carry flag is <strong>RESET</strong></td>
<td><strong>CNC 3050H</strong></td>
</tr>
<tr>
<td>6. <strong>Return</strong> if Zero flag is <strong>SET</strong></td>
<td><strong>RZ</strong></td>
</tr>
</tbody>
</table>