CHAPTER 4

Development of Novel Hardware Algorithms for ESP Controller Processor
CHAPTER 4

Development of Novel Hardware Algorithms for ESP Controller Processor

4.1 Introduction

This chapter discusses the novel algorithms used in realizing various instructions used in Processor design. Before we embark on to write algorithms for instructions, let us have an overview of the Instruction set for which the algorithms have been written.

4.2: Overview of the instruction set:

1) Data Transfer Group
2) Logical Group
3) Arithmetic Operations Group
4) Branch Group
5) Stack Group

All the instructions are similar to Intel 8085 8 bit processor. The novel algorithms for instructions (along with the necessary fragmented code) in each group is given in section 4.3.

The proposed ESP processor emulates the presently existing 8085 based systems. The system is primarily a Finite State Machine (FSM). The process Algorithm may, therefore, be easily be designed using Algorithmic State Machine charts.

All of the processor instructions have been coded which are required for ESP processor. The instructions[1] are realized along with their processing times. At the end of chapter a table which compares the processing speed of the proposed FPGA
implementation of ESP Controller System with the existing ESP Systems is given. Most of the instructions maintain the same processing clock cycles in the two cases. The proposed implementation scores over the existing systems for instructions such as SPHL, DAD, JNZ, HLT and PCHL, whereas it is the other way round for stack operation instructions.

4.3 The Novel Hardware Algorithms for the Microprocessor Architecture

Verilog Code 5

The verilog code 5 (i.e the source code) presents the RTL code for Microprocessor used in ESP controller. The design module is named “microproc”. After declaring the design module, the inputs/outputs are identified. The microprocessor is a FSM and is realized using case statement. All the conditional states of the request and the signals are coded in the same order as the ASM chart and are self–explanatory. The states of the ASM chart are identified by the signal “state”, in the code. The fragment of the code indicating the first three states is shown below for better understanding of the verilog code. The explanation of first three states is given in section 4.3 and the ASM chart. For every instruction the first two states are common. The algorithms differ from state 3 onwards and depending upon the instruction set the complexity varies. All the results can be verified using the timing diagrams for various instructions. In the next section the novel algorithms for various instructions are described followed by timing diagram of each instruction.

```verilog
`define read_delay_count 5'd1 // This is to
delay read pulse by 1 clk pulse.

module microproc (          // Declare design
    clk,                   // I/O s can be
    resetin_n,
   // I/O s can be
   // written in any order.
   `define     read_delay_count       5'd1      // This is to
   delay read pulse by 1 clk pulse.
   module microproc (          // Declare design
   clk,                   // I/O s can be
   resetin_n,
   // I/O s can be
   // written in any order.
```
// Declare the Inputs and Outputs of the module.
input            clk;    // System clock.
input           resetin_n;   // Power-on reset
input           SID;    // Serial Input Data.

output   resetout;   // Power-on reset to clear external outputs etc.
output  [7:0]  addr15_8; // Higher order address bus.
inout  [7:0]  ad;    // Data bus is bidirectional, multiplexing 8-bit data
// and lower-order address bus.
output                  rd_n;   // Active low read signal
output             wr_n;    // Active low write signal
output             ale;    // Address latch enable
output            iom_n;   // Signal to identify I/O or memory operation
output            s1;    // Status signals to identify various operations
output           s0;   // such as memory read, memory write etc.
output   SOD;   // Serial output data pin.

reg         [7:0]       addr15_8;
reg  [7:0]  D2;    // 'D2' is the same as 'ad' and is declared as 'reg'
// since it occurs as output in 'always' block.
reg         D1;

reg         [15:0]   pc;
reg         rd_n;
reg         wr_n;
reg         ale;
reg         iom_n;
reg         s0;
reg         s1;
reg    SOD; // Serial output

reg    [12:0] state;
reg    [7:0] opcode;
reg    [7:0] A; // Accumulator and
other registers
reg    [7:0] B;
reg    [7:0] C;
reg    [7:0] D;
reg    [7:0] E;
reg    [7:0] H;
reg    [7:0] L;
reg    [15:0] SP; // Stack Pointer

reg    [7:0] temp; // Registers for temporary storage
reg    [7:0] temp1;
reg    [7:0] temp2;
reg    [4:0] temp3;
reg    [8:0] temp4;
reg    [16:0] temp5;

reg    resetout;
reg    Z; // Zero flag
reg    CY; // Carry flag
reg    S; // Sign flag
reg    P; // Parity flag
reg    AC; // Auxiliary Carry flag

reg    [4:0] read_cnt; // Counter for delaying the read pulse.

wire    [7:0] ad; // Bi-directional (inout) signal
// must be declared as 'wire' and not as 'reg'.

reg    I7_5; // Interrupts pending.
reg    I6_5;
reg    I5_5;
reg    IE; // Interrupt enable register.
reg    R7_5; // Clear RST 7.5 interrupt flag.

always @ (posedge clk or negedge resetin_n) // FSM realization
of the Microprocessor
begin // as per the ASM chart.

if (resetin_n == 0)
begin // Clear registers to start with.
    resetout <= 1;
    pc <= 0;
    addr15_8 <= 0;
end
rd_n <= 1;
wr_n <= 1;
ale <= 0;
im_n <= 0;
s0 <= 1;
s1 <= 1;
D1 <= 0;
IM <= 8'hF; // Initialize Interrupt

Mask register -
// {MSE, M7.5, M6.5, M5.5}
SOD <= 0; // Clear SOD output.
I7_5 <= 0; // Clear interrupt pending.
I6_5 <= 0; // Clear interrupt pending.
I5_5 <= 0; // Clear interrupt pending.
IE <= 1; // Set interrupt enable register.
R7_5 <= 1; // Clear RST 7.5.
state <= 0 // Initialize state when the system is reset.
end
else
  case (state)
  0: begin
    resetout <= 0;
    pc <= 0; // Initialize program counter,
    addr15_8 <= 0; // address outputs etc.
    read_cnt <= 0;
    rd_n <= 1;
    wr_n <= 1;
    ale <= 0;
im_n <= 0;
s0 <= 1;
s1 <= 1;
D1 <= 0;
CY <= 0;
AC <= 0;
S <= 0;
P <= 0;
Z <= 0;
SP <= 16'hFFFF;
R7_5 <= 0; // Remove Clear RST 7.5 signal.
    state <= 1;
  end
  1: begin // "opcode" fetch machine cycle starts here.
    ale <= 1; // Assert ALE signal.
addr15_8 <= pc [15:8]; // Lower byte

address and data bus are multiplexed.
D2 <= pc [7:0]; // Program counter value stored.
rd_n <= 1; // Don't read as yet.
iom_n <= 0;
s1 <= 1;
s0 <= 1;
state <= 2;
end

2: begin
ale <= 0; // Deactivate address latch.
read_cnt <= read_cnt + 1; // Run counter to generate the read pulse.
rd_n <= 0; // Enable active low read signal to fetch
if (read_cnt != `read_delay_count)
begin
state <= 2; // Continue to count the read pulse.
end
else
begin
opcode <= ad;
read_cnt <= 0; // Clear read pulse counter and state <= 3; // continue processing.
end
end
3: begin
rd_n <= 1; // Don't read now.
case (opcode) // Execution of "opcode"
8'h7F:begin
A <= A; // MOV A,A
pc <= pc+1;
state <= 1;
end
8'h47:begin
// MOV B,A

4.4 Development of Algorithms for the Processor Instructions

Machine Cycles: The four basic machine cycles used in instructions are:

- OPCODE fetch
- Memory Read
- Operand Read
- Memory write

The above operations can be explained by taking examples[2] of various group of instructions and then novel algorithms[3] are developed for each instruction.

4.4.1 Algorithms for Data Transfer, logical and branch Group

1) MOV A,B  (Data transfer group)

This is a one byte instruction. This instruction doesn’t require any data from memory or doesn’t store any data to memory. So only opcode fetch is required to fetch and execute the instruction.

The fragment of the code is taken from the main verilog code of Microprocessor that is developed for this work.

```verilog
3: begin
   rd_n <= 1; // Don't read now.
   case (opcode) // Execution of "opcode"
```
// Data transfer group of
instructions  // Condition flags are not
affected.

INSTRUCTIONS  // MOVE, LOAD AND STORE

// MOV r1, r2 Instructions

8'h78: begin
A <= B;    // MOV A,B
pc <= pc+1;
state <= 1;

(state 1 and state 2 are given in verilog code 5)

The ASM chart for the proposed instruction is presented in Fig. 4.1. It shows the
algorithm of the instructions as has been realized.

Upon switching on the system, a power on reset signal is issued taking the
processor system to ‘Initialize’ state “0” as shown in the ASM chart. In this state, the
processor initializes various control signals such as the read, write, address latch
enable and IO/Memory signals as well as the program counter and the address bus of
the system. The read (rd_n), write (wr_n) and IO/Memory (iom_n) signals are active
low. The program counter (pc) and the address (addr) bus are of width 16 bits. A low
“Iom_n” signal indicates that the processor has selected the memory for access;
otherwise, it has selected Input/Output of the system. The “ale” signal, as the name
implies, is used to separate the time multiplexed 8-bits Address/Data bus named “ad
[7:0]”. The state of the FSM changes from one to another at every rising edge of the
system clock.

With the arrival of the clock, the FSM advances to the next state named “Assert
ALE”, where the “ale” signal is set. The address bus derives the concatenated values
of the MSB 8 bits of the program counter and the Address/Data bus. In the next state
marked “Opcode Fetch”, the user firmware stored in ROM is fetched. First, the Op
code of the instruction is fetched, followed by other bytes in subsequent states,
depending upon which instruction is being processed. In this state, the “ale” signal is withdrawn and the read signal is asserted, thus reading the “opcode” from the User memory. This appears on the Address/Data bus. With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the “opcode” is decoded. Depending upon the instruction, the FSM is taken to a group of other states in order to complete the processing of the instruction. For examples, the processing of the instruction, namely, MOV A, B is shown in the chart.

If the “opcode” read is 78 H, then it is the first one; else if it is AE H for the second instruction mentioned above. In the first case, the accumulator “A” gets the contents of the “B” register, the program counter is incremented by 1 and the control rolls back to state “1” with the arrival of the next clock, ready to process the next instruction. Thus for this instruction, the processing time is 4 clock cycles. The algorithm is similar for all MOV R,R (i.e. register to register operations)

2) XRA M (Logical Group)

3: begin
    rd_n <= 1; // Don't read now.

    case (opcode) // Execution of "opcode"

    8'hAE: begin // XRA M
        addr15_8 <= H; // Higher address for memory write.
        wr_n <= 1; // Don't write as yet.
        iom_n <= 0; // Memory access.
        s1    <= 0; // Advanced status of Memory
        s0    <= 1; // write operation.
        state <= 500; // Continue processing.
    end

    500: begin // Continue to execute XRA M instruction.
        ale   <= 1; // Assert ALE signal.
        D2    <= L; // Lower address for memory read.
        state <= 501; // Continue processing.
    end

67
For the second instruction, the “opcode” read is AE H, in which case the MSB bus address gets the value from H register and branches to the next state “500”. In this state, the LSB bus address is derived from L register and ALE signal is activated once again in order to send the memory address over the address bus. In the next state “501”, the ALE pulse is withdrawn and read pulse is issued in order to fetch the contents of the memory addressed by HL register pair. The next state is the executing phase of the actual instruction XRA M. In this state, the XOR of A and M is evaluated and read pulse is withdrawn. CY, AC flags are cleared and Z, S and P flags are set or cleared accordingly. Finally, the program counter is incremented by 1 and the control loops back to state “1” to continue processing the next instruction. The ASM chart is shown in Fig. 4.1.
Figure 4.1: ASM Chart for MOV A, B and XRA M
Timing Diagram 4.1: Simulation result for MOV A,B

Timing Diagram 4.2: Simulation result for XRA M (AA xor AA=00)
3: ADD B (Arithmetic Group)

Description: This instruction adds the contents of register “B” and accumulator and stores the result in accumulator. The examples of register are all general purpose registers such as A,B,C,D,E,H and L. In addition to the result in accumulator all flags are modified to reflect the result of operation.

begin
    rd_n <= 1; // Don't read now.
    case (opcode) // Execution of "opcode"
        8'h80:begin // ADD B instruction.
            CY, A <= A+B; // Add and Set the carry flag.
            Z <= ((A+B) == 0) ? 1:0; // Set the zero flag.
            S, temp4[6:0]<= A+B; // Set the sign flag - temp4 is dummy.
            P <= (^ (A+B) == 0) ? 1:0; // Compute the parity.
            pc <= pc+1; // Advance the PC.
            state <= 1;
        end

The algorithm for ADD B is given above.

Upon switching on the system, a power on reset signal is issued taking the processor system to ‘Initialize’ state “0” as shown in the ASM chart. In this state, the processor initializes various control signals such as the read, write, address latch enable and IO/Memory signals as well as the program counter and the address bus of the system. The read (rd_n), write (wr_n) and IO/Memory (iom_n) signals are active low. The program counter (pc) and the address (addr) bus are of width 16 bits. A low “iom_n” signal indicates that the processor has selected the memory for access; otherwise, it has selected Input/Output of the system. The “ale” signal, as the name implies, is used to separate the time multiplexed 8-bits Address/Data bus named “ad [7:0]”. The state of the FSM changes from one to another at every rising edge of the system clock. With the arrival of the clock, the FSM advances to the next state named “Assert ALE”, where the “ale” signal is set. The address bus derives the concatenated
values of the MSB 8 bits of the program counter and the Address/Data bus. In the next state marked “Opcode Fetch”, the user firmware stored in ROM is fetched. First, the Op code of the instruction is fetched, followed by other bytes in subsequent states, depending upon which instruction is being processed. In this state, the “ale” signal is withdrawn and the read signal is asserted, thus reading the “opcode” from the User memory. This appears on the Address/Data bus. With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the “opcode” is decoded. Depending upon the instruction, the FSM is taken to a group of other states in order to complete the processing of the instruction. The processing of the instruction, namely, ADD, B is shown in the ASM chart 4.2.

The “opcode” read is 80H, for ADD B instruction. The contents of “B” register are added to the accumulator “A”. The result is concatenation of “CY” i.e. carry flag register and accumulator so as to take care of overflow i.e. if the addition results in a carry, the number of bits will be nine. Here all flags are affected.

The auxiliary carry flag is set if carry is generated by addition of A[3:0] + B[3:0]. Zero flag is set if the operation results in 0 and rest if the result is not 0. The sign flag is set if bit A[7] is 1 after execution of arithmetic operation. The parity is obtained by “exclusive or” of the bits of the result A+B. If the result of XOR is 1 then P=0 else P=1. Finally the program counter is incremented by 1 and the control loops back to state “1” to continue processing the next instruction. The ASM chart is shown in Fig.4.2.
Timing Diagram 4.3: Simulation Result for ADD B (FF+FF=FE)

Here the contents of Register B which has a value FF are added to the accumulator which contains FF. The result is stored in accumulator which is FE. This can be observed in the timing diagram. Here the sign flag is set to 1 (S=1) because bit 7 in accumulator is 1. Parity is set to 0 (P=0) because the number of ones is odd in the result. Carry Flag is set to 1 (CY=1) since the arithmetic operation results in a carry.
Figure. 4.2: ASM Chart for processing ADD B

4) JMP Address (Branch Group)

This is a three byte instruction. So it requires 3 machine cycles to fetch the instruction:
First operand fetch is used to read lower order address, second operand fetch is used to read higher order address. The address read are loaded in PC. So the next instruction executed will be from new address. The algorithm for JMP instruction is explained below with the verilog code.

```verilog
3: begin
   rd_n  <= 1;     // Don't read now.
   case (opcode)           // Execution of "opcode"
      // JUMP INSTRUCTIONS
8'hC3:begin
   pc       <= pc + 1; // JMP instruction: advance pc
   state    <= 196;    // continue processing.
end
196:begin        // Continue execution of JMP instruction.
   ale     <= 1;
   addr15_8 <= pc[15:8];
   D2      <= pc [7:0]; // Load contents of pc onto address bus.
   iom_n   <= 0;
   s1      <= 1;       // Memory read operation.
   s0      <= 0;
   state   <= 197;     // Continue processing.
end
197:begin
   ale     <= 0;
   rd_n    <= 0;      // Enable read signal and
   state   <= 198;    // Continue execution.
end
198:begin
   temp1       <= ad; // load 2nd byte into temp1 reg.
   rd_n        <= 1;   // Disable read.
   pc          <= pc + 1; // Advance pc and
   state       <= 199;  // continue processing.
end
199:begin
   ale      <= 1;
   addr15_8 <= pc[15:8];
   D2       <= pc [7:0];
   iom_n    <= 0;
   s1       <= 1;
   s0       <= 0;
   state    <= 200;
end
200:begin
```

75
ale <= 0;
rd_n <= 0;    // Enable read signal
and
state <= 201;  // continue execution.
end
201:begin
    rd_n <= 1;          // Disable read.
    pc  <= {ad,temp1};   // Load the contents of
    jump address.
    state <= 1;          // to execute next
    instruction.
end

The description of the algorithm upto the state 2 is similar to the one described in previous instructions. For that matter it is common for all instructions. In state three the execution of the instruction is in progress and the opcode read “C3” and the contents of the PC are incremented by one. In the next state “196” ale signal is activated once again. The higher order bus derives the value of the pc[15:8] and the register D2 derives the value of the pc[7:0]. The signal “iom_n” is active low. In the next state “197” the ale signal is withdrawn and the read pulse is issued to load the 2nd byte i.e. the contents of “ad” into the “temp1” register. The read pulse is withdrawn and the program counter is incremented by one. State “199” and state “200” is similar to state “196” and state “197”. These two states enable loading 1st byte into data bus. In the next state “201” the read signal is with drawn and the PC has the concatenated value of “ad” and “temp1”. Finally the control loops back to state “1” to continue processing the next instruction. The ASM chart showing the description of the above algorithm is shown in Fig.4.3. The timing diagram for the simulation of “JMP” instruction is shown in the Timing Diagram 4.4. These values are in agreement with the processing time shown in the sample ROM program below.

Sample User Instruction in ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000:</td>
<td>8'h3E</td>
<td>MVI A, 55</td>
<td>7 clk</td>
</tr>
<tr>
<td>0001:</td>
<td>8'h55</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0002:</td>
<td>8'h7f</td>
<td>MOV A,A</td>
<td>4 clk</td>
</tr>
<tr>
<td>1800:</td>
<td>8'hC3</td>
<td>JMP 0000</td>
<td>10 clk</td>
</tr>
</tbody>
</table>
Figure 4.3: ASM Chart for JMP Instn.
Timing Diagram 4.4: Simulation Result for JMP instruction

The simulation result of JUMP instruction is presented in timing diagram 4.4. It takes 10 clock cycles. The instruction is JMP 0. The jump instruction is in PC value 1800 in the ROM. When the instruction is executed it should jump to location 0. The same thing is reflected in the timing diagram. At the end of PC value 1802 the program counter jumps to 0.

5: MOV M, A (Data Transfer Group)

In this instruction the contents of the specified register will be written into memory location specified by H, L pointers. Here the description of algorithm is same for the first two states as described in ADD B instruction.
3: begin
    rd_n  <= 1;  // Don't read now.

case (opcode) // Execution of "opcode"

8'h77: begin
    addr15_8 <= H;  // MOV M,A
    wr_n    <= 1;  // Don't write as yet.
    iom_n   <= 0;  // Memory access.
    sl      <= 0;  // Advanced status of
    Memory
    s0      <= 1;  // write operation.
    state   <= 87;  // Continue processing.
end

// Move from memory to
register
87: begin
    ale     <= 1;  // Assert ALE signal.
    D2      <= L;  // Lower address for memory
    write.
    state   <= 88;
end

88: begin
    ale     <= 0;  // De-activate address latch
    wr_n    <= 0;  // activate write pulse.
    D2      <= A;  // Send out the contents
    of A.
    state   <= 89;  // Continue processing.
end

89: begin
    wr_n    <= 1;  // De-activate write pulse.
    pc      <= pc+1;  // Advance the program counter
    and return
    state   <= 1;  // to execute the next
    instruction.
end

With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the opcode is decoded. The contents of H are sent to higher order address bus (addr15_8). The signal iom_n is made low and the status signals is taken to advanced state of memory write operation and branches to next state “87”. In this state the “ale” signal is asserted high. The contents of “L” register
are sent to “D2” register and branches out to next state “88”. Here the “ale” signal is deactivated and the writ signal wr_n is asserted. The contents of the accumulator are sent to the register “D2”. And then branches to the next state “89”. In this state the write signal is deactivated and the program counter is incremented by 1 and the control loops back to state one. The ASM chart is shown in Fig.4.4

Sample User Instruction in ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>1523:</td>
<td>8'h77</td>
<td>MOV M, A</td>
<td>7 clk</td>
</tr>
</tbody>
</table>

The timing diagram is shown in timing diagram 4.5.
6: MOV B, M (Data Transfer Group)

The instruction transfers the contents of memory to the specified register. The address of the memory is given by HL register pair. To read data from memory, it requires memory read operation.

1) OPCODE fetch: Address is given by PC to fetch OPCODE. PC is incremented by one.
2) Memory Read: Address is given by HL; as it is memory related operation. PC remains unchanged. When the signal “rd_n” = 0 the contents of the memory location pointed by HL will be transferred to data bus. Microprocessor will read the contents and deactivate the signal “rd_n”.

3: begin
  rd_n <= 1; // Don't read now.
  case (opcode)
    // Execution of "opcode"
    8'h46: begin // MOV B,M
      addr15_8 <= H; // Higher address for memory read.
      rd_n <= 1; // Don't read as yet.
      iom_n <= 0; // Memory access.
      s1 <= 1; // Advanced status of Memory
      s0 <= 0; // read operation.
      state <= 90; // Continue processing.
      end
    end
  90: begin
    ale <= 1; // Assert ALE signal.
    D2 <= L; // Lower address for memory read.
    state <= 91; // Continue processing.
    end
  91: begin
    ale <= 0; // De-activate address latch and
    rd_n <= 0; // activate write pulse.
    state <= 92; // Continue processing.
    end
  92: begin
    B <= ad; // Get the memory contents into B.
    rd_n <= 1; // De-activate read pulse.
    pc <= pc+1; // Advance the program counter and return
    state <= 1; // to execute the next instruction.
  end

With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the opcode is decoded. The contents of H are sent to higher order address bus (addr15_8). The signal iom_n is made low and the status signals is taken to advanced state of memory read operation and branches to next state.
“90”. In this state the “ale” signal is asserted high. The contents of “L” register are sent to “D2” register and branches out to next state “91”. Here the “ale” signal is deactivated and the read signal \(\text{rd}_n\) is asserted and then branches to the next state “92”. In this state the read signal is deactivated and the program counter is incremented by 1 and the control loops back to state one. The ASM chart is shown in Fig. 4.5.

**Sample User Instruction in ROM**

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>1524:</td>
<td>8'h46</td>
<td>MOV B, M</td>
<td>7 clk</td>
</tr>
</tbody>
</table>

**Timing Diagram 4.6: Simulation result for MOV B, M Instruction**

As seen in the waveform, the instruction MOV B, M is located in 1524 (pc value) commences at 22385 ns and completes at 22455 ns. Thus the processing time is 70 ns or 7 clock cycles since the clock runs at 100MHz.
The instruction loads the contents of H and L registers into the stack pointer register, the contents of the H register, provide the high-order address and the contents of the L register provide the low-order address. The contents of the H and L registers are not altered. No flags are affected.

3: begin
rd_n <= 1;  // Don't read now.
case (opcode)                   // Execution of
"opcode"

// STACK OPERATIONS

INSTRUCTIONS
8'hF9: begin
        SP    <= {H, L};       // SPHL
        pc    <= pc+1;
        state <= 1;
end

With the arrival of the third clock, the FSM goes to the state marked “Execute
Instruction”, where the actual processing of the instruction commences. To start with,
the read signal is withdrawn and the “opcode” is decoded. The Stack Pointer derives
the concatenated value of H and L registers. The program counter is incremented by
one and the control loops back to state 1. The ASM chart is shown in Fig.4.6

Sample User Instruction in ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>76:</td>
<td>8'hF9</td>
<td>SPHL</td>
<td>4 clk</td>
</tr>
</tbody>
</table>

As seen in the waveform, the instruction SPHL is located in 76 (pc value)
commences at 2925 ns and completes at 2965 ns. Thus the processing time is 40 ns 4
clock cycles since the clock runs at 100MHz.

Timing Diagram 4.7: Simulation result for SPHL Instruction
Figure. 4.6: ASM Chart for SPHL

8: MVI B, AA (Data Transfer Group)

This is a two byte instruction, so it requires 2 machine cycles to fetch the instructions:

1) Opcode fetch  2) Operand fetch

For execution of the instruction it doesn’t take any data from memory, so only two machine cycles are sufficient. For Opcode fetch cycle the address is given by PC. The program counter is incremented by one. Now it points to the operand. For operand fetch cycle, the address is given by PC. The PC contents are used to read operand. PC is again incremented by one to point to next instruction after MVI. The verilog algorithm for the above instruction is given below.

```
3: begin
    rd_n <= 1;  // Don't read now.
```

With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the opcode is decoded then the program counter is incremented by one and then branches to state 8. In this state “ale” is activated and the address bus derives the value of MSB 8 bits of the program counter. D2 register derives the value of LSB 8 bits of the program counter.

A low “iom_n” signal indicates that the processor has selected the memory for the access. The select signals are s1=1 and s0=0 for memory read operation. In the next state i.e. 9 the “ale” signal is withdrawn and the read signal is activated and
then branches out to state 10. This state is the execution phase and here in this state the data is transferred to register B and the PC is incremented by one and the control loops back to state “1” to continue processing the next instruction. The ASM chart is shown in Fig 4.7.

Sample User Instruction in ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>9:</td>
<td>8’h06</td>
<td>MVI B, AA</td>
<td>7 clk</td>
</tr>
</tbody>
</table>

Timing Diagram 4.8: Simulation result for MVI B, AA Instruction

As seen in the waveform, the instruction MVI B,AA is located in 09 (PC value) commences at 385 ns and completes at 455 ns. Thus the processing time is 70 ns or 7 clock cycles since the clock runs at 100MHz.
The image contains an ASM (Assembly Machine Language) chart for the instruction MVI B, AA. The chart is labeled "Figure 4.7: ASM Chart for MVI B,AA".

### Description: Load Register Pair with 16 Bit Data

This instruction loads 16 bit data specified with the instruction to the \( R_p \) register pair. In the instruction only high order register is specified for register pair i.e. if HL pair is to be loaded only H register will be specified in the instruction. The examples of \( R_p \) are BC pair, DE pair, HL pair, and stack pointer SP. No flags are modified.

### The Verilog Algorithm for the Above Instruction

The verilog algorithm for the above instruction is given below:

1. **Reset**
   - \( rd_n=wr_n=1, ale=0, iom_n=0, pc=0, \)

2. **Assert ALE**
   - \( ale=1, addr[15:0]=\{pc[15:8], ad\} \)

3. **Opcode Fetch**
   - \( ale=0, rd_n=0, opcode=ad \)

4. **Execute Instruction**
   - \( rd_n=1, opcode=8’h06, pc=pc+1 \)

5. **MVI B, AA**
   - \( ale=1, addr15_8=pc[15:8], D2=pc[7:0], iom_n=0, s1=1, s0<=0 \)

6. **State = 1**

---

9: **LXI B, 5555 (Data Transfer Group)**

Description: Load register pair with 16 bit data

This instruction loads 16 bit data specified with the instruction to the \( R_p \) register pair. In the instruction only high order register is specified for register pair i.e. if HL pair is to be loaded only H register will be specified in the instruction. The examples of \( R_p \) are BC pair, DE pair, HL pair, and stack pointer SP. No flags are modified. The verilog algorithm for the above instruction is given below.
3: begin
  rd_n <= 1;  // Don't read now.
end

case (opcode)  // Execution of "opcode"
  // Data transfer group of
  // Condition flags are not
  // MOVE, LOAD AND STORE
INSTRUCTIONS
  // MOV r1, r2

8'h01:begin  // LXI B,16-bit data.
  pc<= pc+1;  //Advance pc to get the first byte of data.
  state <= 94;  // Continue processing.
end

94: begin  // Continue execution of LXI B, #data instruction.
  ale <= 1;
  addr15_8 <= pc[15:8];
  D2 <= pc [7:0];
  iom_n <= 0;
  s1 <= 1;
  s0 <= 0;
  state <= 95;
end

95: begin
  ale <= 0;
  rd_n <= 0;
  state <= 96;
end

96: begin  // Execute 2nd byte of LXI B, #data instruction.
  C <= ad;  // Get byte 2 data.
  rd_n <= 1;
  pc <= pc + 1;
  state <= 97;
end

97: begin
  ale <= 1;
  addr15_8 <= pc[15:8];
  D2 <= pc [7:0];
  iom_n <= 0;
  s1 <= 1;
  s0 <= 0;
  state <= 98;
end
With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the opcode is decoded then the program counter is incremented by one and then branches to state 94. In this state “ale” is activated and the address bus derives the value of MSB 8 bits of the program counter. D2 register derives the value of LSB 8 bits of the program counter. A low “iom_n” signal indicates that the processor has selected the memory for the access.

The select signals are s1=1 and s0=0 for memory read operation. In the next state i.e. 95 the “ale” signal is withdrawn and the read signal is activated and then branches out to state 96. This state is the execution phase here and in this state the data is transferred to register C and the PC is incremented by one and branches to state 97. State 97 and state 98 are similar to state 94 and state 95. In state 99 the third byte of data is transferred to register B and the PC is incremented by one. Further the control loops back to state one. The ASM chart is shown in Fig. 4.8.

Sample User Instruction in ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:</td>
<td>8’h01</td>
<td>LXI B, 5555</td>
<td>10 clk</td>
</tr>
</tbody>
</table>
Timing Diagram 4.9: Simulation result for LXI B, 5555 Instruction

As seen in the waveform, the instruction LXI B, 5555 is located in 63 (PC value) commences at 2485 ns and completes at 2585 ns. Thus the processing time is 100 ns or 10 clock cycles since the clock runs at 100MHz.
Figure. 4.8: ASM Chart for LXI B, 5555 inst.
10: XCHG (Data Transfer Group)

Description: Exchange the contents of HL with DE pair.

This instruction exchanges contents of H register with D register and L register with E register. No flags are modified. The verilog algorithm for the above instruction is given below:

```verilog
3: begin
    rd_n  <= 1;  // Don't read now.
    case (opcode)   // Execution of "opcode"
        // Data transfer group of instructions
        // Condition flags are not affected.
        // MOVE, LOAD AND STORE INSTRUCTIONS
        // MOV r1, r2
     8'hEB:begin
        D  <= H;        // XCHG
        E  <= L;
        H  <= D;
        L  <= E;
        pc <= pc+1;   // Advance pc.
    state  <= 1;
end
```

With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the opcode is decoded. The contents of register DE and HL are exchanged, the program counter is incremented by one and the control loops back to state one. The ASM chart is shown in Fig.4.9

**Sample User Instruction in ROM**

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>75:</td>
<td>8'hEB</td>
<td>XCHG</td>
<td>4 clk</td>
</tr>
</tbody>
</table>

As seen in the waveform (Timing Diagram- 4.10) the instruction XCHG is located in 63 (PC value) commences at 2885 ns and completes at 2925 ns. Thus the processing time is 40 ns or 4 clock cycles since the clock runs at 100MHz.
Timing Diagram 4.10: Simulation result for XCHG Instruction

Figure 4.9: ASM Chart for XCHG
11: LDA 000A  (Data Transfer Group)

Description: Load accumulator direct from memory

This instruction copies the contents of the memory location to the accumulator. The
address of memory location is specified along with the instruction. It is a 3 byte
instruction. So it is stored in memory similar to LXI R_p, data instruction, i.e.
OPCODE as first byte, lower order address as second byte and higher order address as
third byte. No flags are modified.

3: begin
  rd_n <= 1; // Don't read now.
case (opcode)
  // Execution of "opcode"
  // Data transfer group of instructions
  // Condition flags are not effected.
  // MOVE, LOAD AND STORE INSTRUCTIONS
  // MOV r1, r2
  8'h3A:begin  //execute LDA #16 bit address
    pc   <= pc+1;  //advance pc by 1
    state <= 129;  //continue processing
  end
  129:begin  // execute LDA #16-bit address instruction
    ale       <= 1;
    addr15_8  <= pc[15:8];
    D2        <= pc [7:0];  // load the contents
    iom_n     <= 0;
    s1        <= 1;    // Memory read operation
    s0        <= 0;
    state     <= 130;  // continue procesing.
  end
  130:begin
    ale       <= 0;  //disable latch
    read_cnt  <= read_cnt + 1;  //Run counter to
    generate the read pulse.
    if (read_cnt != `read_delay_count)
      begin
        rd_n <= 0 ;//Enable active low read signal to fetch
        state<= 130 ;  //Continue to count the read pulse.
      end
    else
      begin
        templ  <= ad;  //laod 8-bit data into templ reg.
        read_cnt <= 0 ;  //Clear read pulse counter.
state <= 131; //continue processing.
end

131:begin
rd_n <= 1;       // read disable.
pc  <= pc + 1;   //advance pc by 1.
state <= 132;    //continue processing.
end

132:begin
ale <=1;
addr15_8 <= pc[15:8];   // load the contents of pc onto address bus.
D2  <= pc [7:0];
iom_n <= 0;
s1   <= 1;    // Memory read operation
s0   <= 0;
state <= 133;    // continue processing.
end

133:begin
ale <= 0;
//disable latch
read_cnt <= read_cnt + 1 ;//Run counter to generate the read pulse.

if (read_cnt != `read_delay_count)
begin
rd_n <= 0 ;       //Enable active low read signal to fetch
state <= 133 ;   //Continue to count the read pulse.
end
else
begin

temp2  <= ad;     // load 8-bit data into temp2 reg.
read_cnt <= 0 ;   //Clear read pulse counter.
state <= 134 ;    //continue processing.
end
end

134:begin
rd_n <= 1 ;        //read disable.
addr15_8 <= temp2; // Higher address for memory write.
wr_n  <= 1;        // Don't write as yet.
iom_n <= 0;        // Memory access.
s1    <= 0;        // Advanced status of Memory
s0    <= 1;        // write operation.
With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the opcode is decoded then the program counter is incremented by one and then branches to state 129. In this state “ale” is activated and the address bus derives the value of MSB 8 bits of the program counter. D2 register derives the value of LSB 8 bits of the program counter. A low “iom_n” signal indicates that the processor has selected the memory for the access. The select signals are s1=1 and s0=0 for memory read operation. In the next state i.e. 130 the “ale” signal is withdrawn and the read signal is activated and the contents of “ad” is transferred to register temp1 then branches out to state 131. In this state i.e. 131 the read pulse is withdrawn and the PC is incremented by 1 and then branches out to state
132. Similar operation as performed in state 129 is executed and branches out to state 133. In this state the signal “ale” is withdrawn and the read signal is asserted. The data is transferred to register temp2 and branches to state 134. In this state the write operation takes place and the contents of temp2 are placed on higher order address bus and branches to state 135. In this state the signal “ale” is asserted high and the contents of temp1 register are transferred to D2 and then branches out to state 136. In this state the “ale” signal is withdrawn and the read signal is enabled to fetch and branches out to state 137. In this state the contents are loaded to the accumulator and the read pulse is deactivated. The pc value is incremented and the control loops back to state 1. The ASM chart is shown in Fig. 4.10.

Sample User Instruction in ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>93:</td>
<td>8'h3A</td>
<td>LDA 000A</td>
<td>16 clk</td>
</tr>
</tbody>
</table>

Timing Diagram 4.11: Simulation result for LDA 000A Instruction

As seen in the waveform (Timing Diagram-4.11) the instruction LDA 000A is located in 93 (PC value) commences at 8710 ns and completes at 9030 ns. Thus the processing time is 320 ns or 16 clock cycles since the clock runs at 50MHz.
Figure 4.10: ASM Chart for LDA,000A
12: LDAX B (Data Transfer Group)

Description: Load accumulator indirect by using a memory pointer

This instruction copies the contents of the memory location to the accumulator. The address of the memory location is given by the R_p register pair specified along with the instruction. The examples of register pair are B (i.e. BC pair) and D (i.e. DE pair) only. No flags are modified.

3:begin
    rd_n  <= 1;  // Don't read now.
end

case (opcode)  // Execution of "opcode"
  8'h0A: begin  // Execute LDAX B
    addr15_8  <= B;  // Higher address for memory read.
    rd_n      <= 1;  // Don't read as yet.
    iom_n     <= 0;  // Memory access.
    s1        <= 1;  // Advanced status of Memory read.
    s0        <= 0;  // read operation.
    state     <= 237;  // Continue processing.
  end
237: begin  // Execution of LDAX B
    ale     <= 1;  // Assert ALE signal.
    D2      <= C;  // Lower address for memory read.
    state   <= 238;  // Continue processing.
  end
238: begin
    ale     <= 0;  // Disable latch
    read_cnt <= read_cnt + 1;  // Run counter to generate the read pulse.
    if (read_cnt != `read_delay_count)
      begin
        rd_n  <= 0 ;  // Enable active low read signal to fetch
        state <= 238;  // Continue to count the read pulse.
      end
    else
      begin
        read_cnt  <= 0 ;  // Clear read pulse counter.
        A         <= ad;  // Get the memory contents into A.
        pc        <= pc+1;  // Advance the program counter and
      return
        rd_n  <=1;
        state <= 1;  // to execute the next instruction.
    end
  end
With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the opcode is decoded. The contents of B are sent to higher order address bus \( \text{addr15}_{-8} \). The signal \( \text{iom}_n \) is made low and the status signal is taken to advanced state of memory read operation and branches to next state “237”. In this state the “ale” signal is asserted high. The contents of “C” register are sent to “D2” register and branches out to next state “238”. Here the “ale” signal is deactivated and the read signal \( \text{rd}_n \) is asserted and the memory contents are sent to accumulator. The program counter is incremented by 1 and the control loops back to state one. The ASM chart is shown in Fig. 4.11.

**Sample User Instruction in ROM**

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>60:</td>
<td>8’h0A</td>
<td>LDAX B</td>
<td>7 clk</td>
</tr>
</tbody>
</table>

**Timing Diagram 4.12: Simulation result for LDAX B Instruction**
As seen in the waveform (Timing Diagram-4.12) the instruction LDAX B is located in 60 (PC value) commences at 14810 ns and completes at 14670 ns. Thus the processing time is 140 ns or 7 clock cycles since the clock runs at 50MHz.

![ASM Chart for LDAX B](image)

**Figure 4.11: ASM Chart for LDAX B**

**13: LHLD 0008 (Data Transfer Group)**

**Description:** Load HL pair direct from memory.

This instruction copies the contents of the memory locations to H and L registers. The address of the memory location is specified along with the instruction. The contents of the memory location whose address is specified with the instruction are transferred to
“L” register and \((\text{address} + 1)\) contents to “H” register. This instruction is used to load H and L registers from memory. It is a three byte instruction. So its storing format is OPCODE as first byte, Lower order address as second byte and higher order address as third byte.

Operation: (Address) → L register; (Address +1) → H register

3:begin
  rd_n <= 1; // Don't read now.
  case (opcode)
  8'h2A:begin // execute LHLD instruction
    pc <= pc+1; // advance pc by 1
    state <= 247; // continue processing.
    end
  247:begin // execute LHLD 16-bit address instruction
    ale <= 1;
    addr15_8 <= pc[15:8];
    D2 <= pc [7:0]; // load the contents of pc onto address bus.
    iom_n <= 0;
    s1 <= 1; // Memory read operation
    s0 <= 0;
    state <= 248; // continue processing.
    end
  248:begin
    ale <= 0; // disable latch
    read_cnt <= read_cnt + 1; // run counter to generate the read pulse.
    if (read_cnt != `read_delay_count)
      begin
        rd_n <= 0; // Enable active low read signal to fetch
        state <= 248; // Continue to count the read pulse.
      end
    else
      begin
        read_cnt <= 0; // Clear read pulse counter.
        state <= 249; // continue processing.
      end
end
begin
    rd_n <= 1; // read disable.
    temp1 <= ad; // load 8-bit data into temp1 reg.
    pc <= pc + 1; //advance pc by 1.
    state <= 251; // continue processing.
end

begin
    ale <= 1;
    addr15_8 <= pc[15:8]; // load the contents of pc onto address bus.
    D2 <= pc [7:0];
    iom_n <= 0;
    s1 <= 1; // Memory read operation
    s0 <= 0;
    state <= 252; // continue processing.
end

begin
    ale <= 0; //disable latch
    read_cnt <= read_cnt + 1; //Run counter to generate the read pulse.

    if (read_cnt != `read_delay_count)
begin
    rd_n <= 0 ;//Enable active low read signal to fetch pulse.
    state <= 252 ; //Continue to count the read pulse.
end
else
begin
    read_cnt <= 0 ; //Clear read pulse counter.
    state <= 253 ; //continue processing.
end
end

begin
    rd_n <= 1 ; //read disable.
    temp2 <= ad; // load 8-bit data into temp2 reg.
    state <= 254; // continue processing.
end

begin
    addr15_8 <= temp2; // Higher address for memory read.
    rd_n <= 1; // Don't read as yet.
iom_n <= 0; // Memory access.
s1 <= 1; // Advanced status of Memory
s0 <= 0; // read operation.
state <= 255; // Continue processing.
end

255:begin

ale <= 1; // Assert ALE signal.
D2 <= temp1; // Lower address for memory read.
state <= 256; // Continue processing.
end

256:begin

ale <= 0; //disable latch
read_cnt <= read_cnt + 1; //Run counter to generate the read pulse.

if (read_cnt != `read_delay_count)
begin

rd_n <= 0; //Enable active low read signal to fetch
state <= 256; //Continue to count the read pulse.
end
else
begin

read_cnt <= 0; //Clear read pulse counter.
L <= ad; // Get the memory contents into L.
state <= 257; // to execute the next instruction
{temp2,temp1}=(temp2,temp1)+1; //increment address to point to next location.
end
end

257:begin

addr15_8 <= temp2; // Higher address for memory read.
rd_n <= 1; // Don't read as yet.
iom_n <= 0; // Memory access.
s1 <= 1; // Advanced status of Memory
s0 <= 0; // read operation.
State <= 258; // Continue processing.
end

258:begin

ale <= 1; // Assert ALE signal.
D2 <= temp1; // Lower address for memory read.
state <= 259; // Continue processing.
end
259:begin
    ale <= 0;        // disable latch
    read_cnt <= read_cnt + 1; // Run counter to
    generate the read pulse.
    if (read_cnt != `read_delay_count)
    begin
        rd_n <= 0 // Enable active low read signal to
        fetch
        state <= 259;  // Continue to count the read
        pulse.
    end
    else
    begin
        read_cnt <= 0 ;      // Clear read pulse counter.
        H    <= ad;    // Get the memory contents into
        return
        rd_n   <= 1;
        pc   <= pc+1; // Advance the program counter and
        return
        state <= 1; // to execute the next instruction
    end
end

With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the opcode is decoded then the program counter is incremented by one and then branches to state 247. In this state “ale” is activated and the address bus derives the value of MSB 8 bits of the program counter. D2 register derives the value of LSB 8 bits of the program counter. A low “iom_n” signal indicates that the processor has selected the memory for the access. The select signals are s1=1 and s0=0 for memory read operation. In the next state i.e. 248 the “ale” signal is withdrawn and the read signal is activated and then branches out to state 249. In this state the read signal is deactivated and the contents of “ad” is transferred to register temp1, and the program counter is incremented by one and then branches out to state 251. In this state i.e. 251 it is same as state 247 and is used for second operand fetch and branches out to state 252. In this state the “ale” signal is withdrawn and the rd_n is activated subsequently branches out to state 253. In this state the 8 bit data is loaded into temporary register “temp2” and then branches to state 254. In this state the contents of register “temp2” are sent on higher order address bus and the signal
iom_n is activated for memory read operation and branches out to 255. In this state 255 the “ale” signal is activated and the contents of register “temp1” is sent to D2 which is the lower address for memory read and then branches out to state 256. In this state the memory contents are got into register “L” while the read signal is low, the contents of reg “temp1 and temp2” are concatenated and incremented by one to point to next location and branches to state 257. The states 257 and 258 are similar to states 254 and 255 i.e. for memory read operation and branches out to state 259. In this state the memory contents are got into register H. and the program counter is incremented by one. Next the control loops back to state 1. The ASM chart is given in Fig. 4.12.

Sample User Instruction in ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>08</td>
<td>8’hA0</td>
<td>ANA B</td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>8’h67</td>
<td>MOV H, A</td>
<td></td>
</tr>
<tr>
<td>70:</td>
<td>8’h2A</td>
<td>LHLD 0008</td>
<td>20 clk</td>
</tr>
</tbody>
</table>

Timing Diagram 4.13: Simulation result for LHLD 0008 Instruction
From the timing diagram we observe that the contents of the memory location 08 i.e. “A0” are copied to reg “L” and the contents of memory location 09 are copied to “H” register. The instruction takes 22 clock cycles.
**14: SHLD 2000 (Data Transfer Group)**

**Description: Store HL pair direct from memory.**

This instruction copies the contents of register “H” and “L” to the memory locations. The address of the memory location is specified along with the instruction. The contents of “L” register are stored at the memory location whose address is specified and the contents of register “H” to the (address + 1) location. This instruction is used to store “H” and “L” registers direct to memory. The storing format is OPCODE as first byte, lower order address as second byte and higher order address as third byte. This can be observed in the timing diagram 4.14. The ASM chart is given in Fig.4.13

**Operation** : 
- \( L \text{ reg} \rightarrow \text{ (Address)} \)
- \( H \text{ reg} \rightarrow \text{ (Address + 1)} \)

**Example** : SHLD 2000 : Store HL pair to memory locations 2000 and 2001. In the timing diagram 4.14 shown below \( L=a0 \text{ h}, \ H=67 \text{ h}, \) at memory location 2000 and 2001

```verilog
3:begin
  rd_n <= 1;  // Don't read now.
case (opcode)  // Execution of "opcode"
  8'h22:begin  // execute SHLD instruction
    pc <= pc+1;  // advance pc by 1
    state <= 260;  // continue processing.
  end
260:begin  // execute SHLD 16-bit address instruction
  ale<=1;
  addr15_8 <= pc[15:8];
  D2 <= pc [7:0];  // load the contents of pc onto address bus.
  iom_n <= 0;
  sl <= 1;  // Memory read operation
  s0 <= 0;
  state <= 261;  // continue processing.
end
261:begin
  ale <= 0;  // disable latch
  read_cnt <= read_cnt + 1 ;  // run counter to generate the read pulse.
```

110
if (read_cnt != `read_delay_count)
begin
  rd_n <= 0;    // Enable active low read signal to
fetch
  state <= 261;  // Continue to count the read pulse.
end
else
begin
  read_cnt <= 0;    //Clear read pulse counter.
  state <= 262;    //continue processing.
end
end

262:begin
  rd_n <= 1;    // read disable.
  temp1 <= ad;   // load 8-bit data into temp1
  pc <= pc + 1;  //advance pc by 1.
  state <= 263;  // continue processing.
end

263:begin
  ale <= 1;
  addr15_8 <= pc[15:8];     // load the contents of
  pc onto address bus.
  D2 <= pc [7:0];
  iom_n <= 0;
  s1 <= 1;     // Memory read operation
  s0 <= 0;
  state <= 264;  // continue processing.
end

264:begin
  ale <= 0;       //disable latch
  read_cnt <= read_cnt + 1;  //Run counter to
generate the read pulse.
if (read_cnt != `read_delay_count)
begin
  rd_n <= 0;    //Enable active low read signal to
fetch
  state <= 264;  //Continue to count the read pulse.
end
else
begin
  read_cnt <= 0;    //Clear read pulse
  counter.
  state <= 265;    //continue processing.
end
end

265:begin
  rd_n <= 1 ; // read disable.
  temp2  <= ad; // load 8-bit data into temp2 reg.
  state  <= 266; // continue processing.
end

266:begin
  addr15_8 <= temp2; // Higher address for memory write.
  wr_n    <= 1;  // Don't write as yet.
  iom_n   <= 0;  // Memory access.
  s1      <= 0;  // Advanced status of Memory
  s0      <= 1;  // write operation.
  state   <= 267; // Continue processing.
end

267:begin
  ale    <= 1;  // Assert ALE signal.
  D2     <= temp1; // Lower address for memory write.
  state  <= 268;
end

268:begin
  ale    <= 0;  // De-activate address latch and write_cnt <= write_cnt+1;
  if (write_cnt != `write_delay_count)
    begin
      wr_n <= 0; // activate write pulse.
      D2   <= L; // Send out the contents of L.
      D1   <= 1; // Enable Sending out the contents of L.
      state <= 268; // Continue processing.
    end
  else
    begin
      write_cnt <= 0; // De-activate write pulse
      state    <= 269; // Continue processing.
    end
end

269:begin
  D1     <= 0; // Disable Sending out the contents of L.
  wr_n   <= 1; // De-activate write pulse.
{temp2, temp1} <= {temp2, temp1} + 1;  // increment address
  to point to next location.
end

state <= 270;  // to execute the next instruction.
end

270:begin
  addr15_8 <= temp2;    // Higher address for memory
  write.
  wr_n  <= 1;  // Don't write as yet.
  iom_n <= 0;  // Memory access.
  s1    <= 0;  // Advanced status of Memory
  s0    <= 1;  // write operation.
  state <= 271;  // Continue processing.
end

271:begin
  ale  <= 1;    // Assert ALE signal.
  D2   <= temp1;  // Lower address for memory
  write.
  state <= 272;
end

272:begin
  ale  <= 0;  // De-activate address latch
  and
  write_cnt <= write_cnt + 1;
  if (write_cnt != `write_delay_count)
  begin
    wr_n  <= 0;  // activate write pulse.
    D2    <= H;  // Send out the contents of H.
    D1    <= 1;  // Enable Sending out the contents
    of H.
    state <= 272;  // Continue processing.
  end
else
  begin
    write_cnt <= 0;  // De-activate write
    pulse
    state <= 273;  // Continue processing.
  end
end

273:begin
  D1  <= 0;  // Disable Sending out the contents of H.
  wr_n <= 1;  // De-activate write pulse.
  pc   <= pc+1;  // Advance the program counter and
  return
  state <= 1;  // to execute the next instruction.
end
Figure 4.13: ASM Chart for SHLD 2000

- **Reset**
  - $rd_n=wr_n=1$, $ale=0$
  - $iom_n=0$, $pc=0$, $addr=0$

- **Assert ALE**
  - $ale=1$
  - $addr[15:0]=(pc[15:8], ad)$

- **Opcode Fetch**
  - $ale=0$, $rd_n=0$
  - $opcode=ad$

- **Execute Instruction**
  - $rd_n=1$, $opcode=8'h2A$
  - $pc=pc+1$

- **Assert ALE**
  - $ale=1$, $addr15_8=pc[15:8]$
  - $D2=pc[7:0]$, $iom_n=0$
  - $so=0$, $s1=1$

- **Enable Read**
  - $ale<=0$, $rd_n<=0$

  - $ale=1$, $addr15_8=pc[15:8]$
  - $D2=pc[7:0]$, $iom_n=0$
  - $so=0$, $s1=1$

  - $ale=1$, $addr15_8=pc[15:8]$
  - $D2=pc[7:0]$, $iom_n=0$
  - $so=0$, $s1=1$

- **State 1**
  - $ale=0$, $wr_n=0$, $D2=L$
  - $D1=1$

- **State 2**
  - $ale=0$, $wr_n=0$, $D2=H$
  - $D1=1$
With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the opcode is decoded then the program counter is incremented by one and then branches to state 260. In this state “ale” is activated and the address bus derives the value of MSB 8 bits of the program counter. D2 register derives the value of LSB 8 bits of the program counter. A low “iom_n” signal indicates that the processor has selected the memory for the access. The select signals are s1=1 and s0=0 for memory read operation and branches to next state.

In the next state i.e. 261 the “ale” signal is withdrawn and the read signal is activated and then branches out to state 262. In this state the read signal is deactivated and the contents of “ad” is transferred to register temp1, and the program counter is incremented by one and then branches out to state 263. In this state i.e. 263 it is same as state 260 and is used for second operand fetch and branches out to state 264. In this state the “ale” signal is withdrawn and the rd_n is activated subsequently branches out to state 265. In this state the 8 bit data is loaded into higher order address bus, and the status signals are set for memory write operation and branches out to state 267. In this state the ale is asserted high and the contents of temp1 are sent to register D2 and branches out to state 268. In this state the contents of “L” register are sent out to D2 register and branches out to 269. In this state we concatenate temp2 and temp1 and increment by one.

The next state is 270 and 271 is used for higher address and lower address for memory write operation, and then branches to state 272. In state 272 the signal “ale” is withdrawn the reg “H” contents are sent to D2 and branches to 273. In this state the pc is incremented by one and the control loops back to state one.
Sample User Instruction in ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>73:</td>
<td>8'h22</td>
<td>SHLD 2000</td>
<td>22 clk</td>
</tr>
</tbody>
</table>

Timing Diagram 4.14: Simulation result for SHLD 2000 Instruction

From the timing diagram 4.14 we observe that the contents of “L” register i.e. “A0” are stored in memory location 2000 and contents of “H” register i.e. “67” in memory location 2001. As seen in the waveform the instruction SHLD 2000h is located in 73 (PC value) commences at 7050 ns and completes at 7490 ns. Thus the processing time is 440 ns or 22 clock cycles since the clock runs at 50MHz.
15: STA 2000 (Data Transfer Group)

Description: Store accumulator direct to memory. This instruction copies the contents of the accumulator to the memory location. The address of the memory location is specified along with the instruction. It is a three byte instruction. So its storing format is OPCODE as first byte, lower order address as second byte and higher order address as third byte.

- Operation: A → (Address)
- Flags: No flags are modified.
- Example: STA 2000 H: Store accumulator contents at memory location 2000 H

```plaintext
3:begin  
  rd_n  <= 1;  // Don't read now.
  case (opcode)  // Execution of "opcode"
  8'h32:begin   //execute STA #16 bit address
    pc  <= pc+1;  //advance pc by 1
    state  <= 138;  //continue processing
  end
138:begin  // execute STA #16-bit address instruction
  ale  <=1;  
  addr15_8  <= pc[15:8];  
  D2  <= pc [7:0];  // load the contents of pc onto address bus.
  iom_n  <= 0;  
  s1  <= 1;  // Memory read operation
  s0  <= 0;  
  state  <= 139;  // continue processing.
end
139:begin  
  ale  <= 0;  //disable latch
  read_cnt  <= read_cnt + 1;  //Run counter to generate the read pulse.
  if (read_cnt != `read_delay_count) begin
    rd_n  <= 0 ; //Enable active low read signal to fetch
  state  <= 139 ; //Continue to count the read pulse.
  end
else
begin
```
temp1 <= ad; //load 8-bit data into temp1 reg.

read_cnt <= 0; //Clear read pulse counter.
state <= 140; //continue processing.
end
end

140:begin
rd_n >= 1; //read disable.
pc <= pc + 1; //advance pc by 1.
state <= 141; //continue processing.
end

141:begin
ale <= 1;
addr15_8 <= pc[15:8]; //load the contents of pc onto address bus.
D2 <= pc [7:0];
iom_n <= 0;
s1 <= 1; //Memory read operation
s0 <= 0;
state <= 142; //continue processing.
end

142:begin
ale <= 0; //disable latch
read_cnt <= read_cnt + 1; //Run counter to generate the read pulse.
if (read_cnt != `read_delay_count)
begin
rd_n <= 0; //Enable active low read signal to fetch
state <= 142; //Continue to count the read pulse.
end
else
begin
reg.
temp2 <= ad; //load 8-bit data into temp2 counter.
read_cnt <= 0; //Clear read pulse
state <= 143; //continue processing.
end
end
143:begin
    rd_n     <= 1 ; //read disable.
    addr15_8 <= temp2; // Higher address for memory write.
    wr_n     <= 1; // Don't write as yet.
    iom_n    <= 0; // Memory access.
    s1       <= 0; // Advanced status of Memory
    s0       <= 1; // write operation.
    state    <=144; // Continue processing.
end

144:begin
    ale     <= 1; // Assert ALE signal.
    D2      <= temp1; // Lower address for memory read.
    state   <=145; // Continue processing.
end

145:begin
    ale     <= 0; // De-activate address latch and
    write_cnt <=write_cnt+1;
    if (write_cnt != `write_delay_count)
        begin
            wr_n <= 0; // activate write pulse.
            D2   <= A; // Send out the contents of A.
            D1   <= 1; // Enable Sending out the contents of A.
            state<= 145; // Continue processing.
        end
    else
        begin
            write_cnt <=0; // De-activate write pulse
            state    <=146; // Continue processing.
        end
end

146:begin
    D1     <= 0; // Disable Sending out the contents of temp.
    wr_n   <= 1; // De-activate write pulse.
    pc     <= pc+1; // Advance the program counter and
    return
    state <= 1; // to execute the next instruction.
end
With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the opcode is decoded then the program counter is
incremented by one and then branches to state 138. In this state “ale” is activated and the address bus derives the value of MSB 8 bits of the program counter. D2 register derives the value of LSB 8 bits of the program counter. A low “iom_n” signal indicates that the processor has selected the memory for the access.

The select signals are s1=1 and s0=0 for memory read operation. In the next state i.e. 139 the “ale” signal is withdrawn and the read signal is activated and the contents of “ad” is transferred to register temp1 then branches out to state 140. In this state i.e. 140 the read pulse is withdrawn and the pc is incremented by 1 and then branches out to state 141. Similar operation as performed in state 138 is executed and branches out to state 142. In this state the signal “ale” is withdrawn and the read signal is asserted. The data is transferred to register temp2 and branches to state 143.

In this state the write operation takes place and the contents of temp2 are placed on higher order address bus and branches to state 144. In this state the signal “ale” is asserted high and the contents of temp1 register are transferred to D2 and then branches out to state 145. In this state the “ale” signal is withdrawn and the write signal is enabled to send out the contents of “A” to reg “D2”. and branches out to state 146. In this state the write pulse is deactivated. The PC value is incremented and the control loops back to state 1. The ASM chart is shown in Fig. 4.14

<table>
<thead>
<tr>
<th>Sample User Instruction in ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
</tr>
<tr>
<td>90:</td>
</tr>
</tbody>
</table>
From the timing diagram 4.15 we observe that the contents of “A” register i.e. “AA” are stored in memory location 2000. As seen in the waveform the instruction STA 2000h is located in 90 (pc value) commences at 8390 ns and completes at 8710 ns. Thus the processing time is 320 ns or 16 clock cycles since the clock runs at 50MHz.

### 4.4.2 Algorithms for Arithmetic Operations Group

#### 16: ADC C (Arithmetic Group)

Description: This instruction adds the contents of register “C”, Carry flag “CY” and accumulator “A” and stores the result in accumulator. The examples of registers are all general purpose registers such as A, B, C, D, E, H and L. In addition to the result in accumulator all the flags are modified to reflect the result of operation.

- Operation: A+C+CY → A
• All flags are modified to reflect the result of operation.

Example: ADC C: A+C+CY → A

At PC value 73 the instruction ADC C is stored in ROM. When this instruction is executed (refer timing diagram 4.16)  

A = 1010 1011  
C = 0101 0110  
CY = 1

A = 0000 0010  
CY = 1

Therefore flags are also modified to reflect the result.

\[
\begin{array}{cccccc}
\text{CY} & \text{Z} & \text{P} & \text{S} & \text{AC} \\
1 & 0 & 0 & 0 & 0 \\
\end{array}
\]

With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the “opcode” is decoded. Depending upon the instruction, the FSM is taken to a group of other states in order to complete the processing of the instruction. The processing of the instruction, namely, ADC, C is shown in the flowchart 28. The “opcode” read is 89H, for ADC C instruction. The contents of “C” register are added to the accumulator “A” along with the carry. The result is stored in accumulator. Here all flags are affected. The auxiliary carry flag is
set if carry is generated by addition of $A[3:0]+C[3:0]+CY$; Zero flag is set if the operation results in 0 and rest if the result is not 0. The sign flag is set if bit $A[7]$ is 1 after execution of arithmetic operation. The parity is obtained by “exclusive or” of the bits of the result $A+C+CY$. If the result of XOR is 1 then $P=0$ else $P=1$. Finally the program counter is incremented by 1 and the control loops back to state “1” to continue processing the next instruction. The ASM chart is shown in Fig. 4.15

Sample User Instruction in ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>73:</td>
<td>8'h89</td>
<td>ADC C</td>
<td>5 clk</td>
</tr>
</tbody>
</table>

Timing Diagram 4.16: Simulation Result for ADC C  (56+ab +1=02)

Here the contents of Register C which has a value 56h are added to the accumulator which contains ABh along with carry 1. The result is stored in accumulator which is 02h. This can be observed in the timing diagram 4.16. Here the sign flag is set to 0 ($S=0$) because bit 7 in accumulator is 0. Parity is set to 0 ($P=0$) because the number of
ones is odd in the result. Carry Flag is set to 1 (CY=1) since the arithmetic operation results in a carry. The auxiliary carry flag is 0 and the carry flag is set to 1.

17: ADC M (Arithmetic Group)
Description: Add memory location and carry flag contents to accumulator.

This instruction uses HL pair as memory pointer. The contents of memory location addressed by HL pair are added to the accumulator and the result is stored in accumulator. All flags are also modified to reflect the result of operation.

- Operation: (HL) + CY + A → A or M + CY + A → A
• All flags are modified to reflect the result of addition.

• Example: ADC M : A + (HL) + CY → A (Refer timing diagram 4.17)

At PC value 86 the instruction ADC C is stored in ROM. When this instruction is executed (refer timing diagram 4.17)  

\[ A = 0000 \ 1111 \]
\[ M = 1100 \ 0110 \]
\[ CY = 0 \]
\[ A = 1101 \ 0101 \]

Therefore flags are also modified to reflect the result.

\[
\begin{align*}
    CY &= 0 \\
    Z &= 0 \\
    AC &= 1 \\
    S &= 1 \\
    P &= 0
\end{align*}
\]

3:begin
    rd_n <= 1; // Don't read now.
    case (opcode) // Execution of "opcode"
        8'h8E:begin // ADC M
            addr15_8 <= H; // Higher address for memory read.
            rd_n <= 1; // Don't read as yet.
            iom_n <= 0; // Memory access.
            s1 <= 1; // Advanced status of Memory read.
            s0 <= 0; // read operation.
            state <= 105; // Continue processing.
        end
    end

105:begin // execution of ADC M
    ale <= 1; // Assert ALE signal.
    D2 <= L; // Lower address for memory read.
    state <= 106; // Continue processing.
end

106:begin
    ale <= 0; // disable latch
    read_cnt <= read_cnt + 1; // Run counter to generate the read pulse.
    if (read_cnt != `read_delay_count)
        begin
            rd_n <= 0; // Enable active low read signal to fetch
            state <= 106; // Continue to count the read pulse.
            end
        else
            begin
                read_cnt <= 0; // Clear read pulse counter.
                state <= 107; // Continue processing
            end

126
end
107:begin
    temp <= ad; // Get the memory contents into temp.
    rd_n <= 1;  // De-activate read pulse.
    state <= 108; // to execute the next instruction.
end
108:begin
    temp4  <= A + temp + CY;  // continue execute ADC M data instruction
    state <= 37;  // continue processing
end
37:begin
    rd_n <= 1; // disable read and write
    wr_n <= 1;
    ale  <= 0; // disable latch
    AC   <= temp3[4]; // check carry and auxiliary carry flags
    CY <= temp4[8];
    A  <= temp4[7:0]; // store result from temporary reg to A
    P  <= (^A==0) ? 1:0; // check parity flag
    Z  <= (A == 0); // check zero flag
    S  <= A[7]; // check sign flag
    pc <= pc+1; // advance pc by 1
    state <= 1; // to execute next instruction
end

With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the “opcode” is decoded. The “opcode” read is 8E H, in which case the MSB bus address gets the value from H register and branches to the next state “105”. In this state, the LSB bus address i.e. D2 is derived from L register and ALE signal is activated once again in order to send the memory address over the address bus. In the next state “106”, the ALE pulse is withdrawn and read pulse is issued in order to fetch the contents of the memory addressed by HL register pair. The next state i.e. 107 is the executing phase of the actual instruction ADC M. In this state, the memory contents are got into temp register and read pulse is withdrawn and branches to state 108.
In this state the auxiliary carry flag is tested and the actual addition takes place and then branches to state 37. In this state CY, AC, Z, S and P flags are set or cleared accordingly. Finally, the program counter is incremented by 1 and the control loops back to state “1” to continue processing the next instruction. The ASM chart is shown in Fig.4.16.

Sample User Instruction in ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>86:</td>
<td>8'h8E</td>
<td>ADC M</td>
<td>10 clk</td>
</tr>
</tbody>
</table>

Timing Diagram 4.17: Simulation Result for ADC M (0F+C6 +0=D5)

Here the contents of memory C6h are added to the accumulator which has a value 0Fh along with carry 0. The result is stored in accumulator which is D5h. This can be observed in the timing diagram 4.17. Here the sign flag is set to 1 (S=1) because bit 7 in accumulator is 1. Parity is set to 0 (P=0) because the number of ones is odd in the result. Carry Flag is set to 0 (CY=0) since the arithmetic operation results in no carry. The auxiliary carry flag is 1 and the zero flag is set to 0 as result is not equal to zero.
Description: Add immediate data to accumulator.

This instruction adds the 8 bits of data specified along with the instruction to the accumulator and the result is stored in the accumulator. All flags are modified to
reflect the result of operation. The storing format of this instruction will be first byte OPCODE and second byte operand (data).

At PC value 19 the instruction ADI 55 is stored in ROM. When this instruction is executed (refer timing diagram 4.18) data=0101 0101

\[ A = 1010 1010 \]

Therefore flags are also modified to reflect the result.

\[
\begin{array}{cccccc}
\text{CY} &=& 0 & \quad \text{Z} &=& 0 & \quad \text{AC} &=& 0 & \quad \text{S} &=& 1 & \quad \text{P} &=& 1 \\
\end{array}
\]

3:begin
   rd_n <= 1; // Don't read now.
   case (opcode) // Execution of "opcode"
     8'hc6:begin //execute ADI A,8 BIT instruction
       pc <= pc+1; //advance pc by 1 to get the 8bit data

       state <= 33; //continue processing.
     end
     //begin
   end
33:begin //continue executing ADI A,8 bit data instruction
   ale <= 1; // latch enable
   addr15_8 <= pc[15:8]; //load pc contents onto address bus
   D2 <= pc[7:0]; //memory access
   iom_n <= 0; //memory read cycle
   s1 <= 1; //memory read cycle
   s0 <= 0;
   state <= 34; //continue processing.
end

34:begin
   ale <= 0; //disable latch
   read_cnt <= read_cnt + 1; //Run counter to generate the read pulse.
   if (read_cnt != `read_delay_count)
   begin
     rd_n <= 0; //Enable active low read signal to fetch
     state <= 34; //Continue to count the read pulse.
   end
   else
   begin
     read_cnt <= 0; //Clear read pulse counter.
     state <= 35; //continue processing
   end
35:begin
   rd_n <= 1; //disable read signal
   temp <= ad; //move immediate data to temporary register.
   state <= 36; //continue processing
end

A=1111 1111
36:begin
    temp4 <= A + temp; // execute ADI A, 8-bit data
end
37:begin
    rd_n <= 1;       // disable read and write
    wr_n <= 1;
    ale <= 0;       // disable latch
    AC <= temp3[4]; // check carry and auxiliary
    state <= 37;    // continue processing
end

With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the opcode is decoded then the program counter is incremented by one and then branches to state 33. In this state “ale” is activated and the address bus derives the value of MSB 8 bits of the program counter. D2 register derives the value of LSB 8 bits of the program counter. A low “iom_n” signal indicates that the processor has selected the memory for the access. The select signals are s1=1 and s0=0 for memory read operation. In the next state i.e. 34 the “ale” signal is withdrawn and the read signal is activated and branches to state 35. In this state i.e. 35 the contents of “ad” is transferred to register “temp” and then branches out to state 36. In state 36 the contents of “temp” and the contents of accumulator are added and stored in register “temp4”. The accumulator contents i.e. A[3:0] and temp[3:0] are added to check for the auxiliary flag and then branches out to state 37. In this state i.e. 37 the read pulse and write pulse are withdrawn. The signal “ale” is
deactivated, all the flags are tested, the pc is incremented by 1 and then the control
loops back to state 1. The ASM chart is shown in Fig.4.17.

Sample User Instruction in ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>16:</td>
<td>8'hC6</td>
<td>ADI 55</td>
<td>10 clk</td>
</tr>
</tbody>
</table>

Timing Diagram 4.18: Simulation Result for ADI 55 (55+AA=FF)

Here the data 55h is added to the accumulator which has a value AAh. The result is
stored in accumulator which is FFh. This can be observed in the timing diagram 4.18.
Here the sign flag is set to 1 (S=1) because bit 7 in accumulator is 1. Parity is set to 1
(P=1) because the number of ones is even in the result. Carry Flag is set to 0 (CY=0)
since the arithmetic operation results in no carry. The auxiliary carry flag is 0 and the
zero flag is set to 0 as result is not equal to zero. As seen in the waveform the
instruction ADI 55h is located in 16 (pc value) commences at 1430 ns and completes
at 1630 ns. Thus the processing time is 200 ns or 10 clock cycles since the clock runs
at 50MHz.
Figure. 4.17: ASM Chart ADI 55
19:   ACI 55 (Arithmetic Group)

Description: Add immediate data and carry flag to accumulator.

This instruction adds the immediate data, carry flag and accumulator and stores the result in accumulator. All flags are modified to reflect the result of operation. The storing format is first byte opcode and second byte operand (data).

- Operation: \( A + \text{data} + CY \rightarrow A \)

At PC value 10 the instruction ACI 55 is stored in ROM. When this instruction is executed (refer timing diagram 4.19) \( \text{data}=0101\ 0101 \)
\( A =0000\ 0000 \)
\( CY=\ 1 \)

\[ \text{----------------------------------------} \]
\( A=0101\ 0110 \)

Therefore flags are also modified to reflect the result.

<table>
<thead>
<tr>
<th>CY=0</th>
<th>Z=1</th>
<th>AC=0</th>
<th>S=0</th>
<th>P=1</th>
</tr>
</thead>
</table>
| 3:begin
| \( \text{rd}_n \) <= 1; \ // Don't read now.
| case (opcode) \ // Execution of "opcode"
| 8'hCE:begin \ //execute ACI A,8 BIT instruction
| \( \text{pc} \) <=pc+1; \ //advance pc by 1 to get the 8bit data
| \( \text{state} \) <=42; \ //continue processing.
| end
| 42:begin \ // execute ACI A,8-bit data instruction
| \( \text{ale} \) <= 1; \ //latch enable
| \( \text{addr15_8} \) <= \( \text{pc}[15:8] \); \ //load pc contents onto address bus
| \( \text{D2} \) <= \( \text{pc}[7:0] \); \ //memory access
| \( \text{iom}_n \) <= 0; \ //memory read cycle
| \( \text{s1} \) <= 1; \ //memory read cycle
| \( \text{s0} \) <= 0;
| \( \text{state} \) <= 43; \ //continue processing
| end
| 43:begin
| \( \text{ale} \) <= 0; \ //disable latch
| \( \text{read}_\text{cnt} \) <= \( \text{read}_\text{cnt} + 1 \); \ //Run counter to generate the read pulse.
| if (read_cnt != 'read_delay_count)
| begin
| \( \text{rd}_n \) <= 0 ; \ //Enable active low read signal to fetch
| \( \text{state} \) <= 43; \ //Continue to count the read pulse.
| end
| else
| begin
With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the opcode is decoded then the program counter is incremented by one and then branches to state 42. In this state “ale” is activated and the address bus derives the value of MSB 8 bits of the program counter. D2 register derives the value of LSB 8 bits of the program counter. A low “iom_n” signal indicates that the processor has selected the memory for the access. The select signals are s1=1 and s0=0 for memory read operation. In the next state i.e. 43 the “ale” signal is withdrawn and the read signal is activated and branches to state 44. In this state i.e. 44 the contents of “ad” is transferred to register “temp” and then branches out to state 45. In state 45 the contents of “temp” and the contents of accumulator are added
along with carry(CY) and stored in register “temp4”. The accumulator contents i.e. A[3:0] and temp[3:0] plus CY are added to check for the auxiliary flag and then branches out to state 37. In this state i.e. 37 the read pulse and write pulse are withdrawn. The signal “ale” is deactivated, all the flags are tested, the pc is incremented by 1 and then the control loops back to state 1. The ASM chart is shown in Fig.4.18

Timing Diagram 4.19: Simulation Result for ACI 55  (55+0+1=56)

Here the data 55h is added to the accumulator which has a value 00h along with carry=1. The result is stored in accumulator which is 56h. This can be observed in the timing diagram 4.19. Here the sign flag is set to 0 (S=0) because bit 7 in accumulator is 1. Parity is set to 1 (P=1) because the number of one’s is even in the result. Carry Flag is set to 0( CY=0) since the arithmetic operation results in no carry. The auxiliary carry flag is 0 and the zero flag is set to 0 as result is not equal to zero. As seen in the waveform the instruction ACI 55h is located in 10 (pc value) commences at 850 ns and completes at 1050 ns. Thus the processing time is 200 ns or 10 clock cycles since the clock runs at 50MHz.
Figure 4.18: ASM Chart ACI 55
20: DAD B (Arithmetic Group)

Description: This instruction adds the contents of the specified register to “HL” pair and stores the result in “HL” pair. The examples of register pair are “SP”, “BC”, “DE” and “HL”. Only carry flag is modified to reflect the result of operation.

- Operation: BC + HL → HL

At PC value 28 the instruction DAD B is stored in ROM. When this instruction is executed (refer timing diagram 24) =FE54 BC 54FE HL

5352 HL

With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the opcode is decoded then the concatenated value of “BC” is added to the concatenated value of “HL” and stored in register “temp5” and branches to state 57. In this state the read and write signals are withdrawn and tested for carry flag. The result is stored in the “HL” pair. Then the program counter is
incremented by 1 and the control loops back to state 1. The ASM chart is shown in Fig. 4.19.

Sample User Instruction in ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>28:</td>
<td>8’h09</td>
<td>DAD B</td>
<td>5 clk</td>
</tr>
</tbody>
</table>

Timing Diagram 4.20: Simulation Result for DAD B (54FE+FE54=5351)

Here the HL contents 54FE is added to the contents of BC i.e. FE54, the result is stored HL pair. This can be observed in the timing diagram 4.20. Only carry flag is modified.

As seen in the waveform the instruction DAD B is located in 28 (pc value) commences at 2450 ns and completes at 2550 ns. Thus the processing time is 100 ns or 5 clock cycles since the clock runs at 50MHz.
21: **SUB B (Arithmetic Group)**

This instruction subtracts the contents of the register “B” from the contents of the accumulator and the result is placed in accumulator. The contents of “B” register are not altered. All flags are modified to reflect the result. Examples of register “R” are A, B, C, D, E, H, and L.
• Operation: \( A - B \rightarrow A \)

At PC value 64 the instruction \( \text{SUB} \ B \) is stored in ROM. When this instruction is executed (refer timing diagram 4.21), \( A = 00 \quad FF \quad B \)

---

01 \( A \)

3:begin

\[
\begin{align*}
\text{rd}_n & \leftarrow 1; \quad \text{// Don't read now.} \\
\text{case (opcode)} & \quad \text{// Execution of "opcode"} \\
8'h90:begin & \quad \text{//Execute SUB B instruction} \\
\text{temp4} & \leftarrow A - B; \quad \text{//subtract B from A, store result in temporary register} \\
\text{temp3} & \leftarrow A[3:0] - B[3:0]; \quad \text{ //to check for auxiliary carry flag} \\
\text{state} & \leftarrow 37; \quad \text{//continue processing.}
\end{align*}
\]

end

37:begin

\[
\begin{align*}
\text{rd}_n & \leftarrow 1; \quad \text{//disable read and write} \\
\text{wr}_n & \leftarrow 1; \\
\text{ale} & \leftarrow 0; \quad \text{//disable latch} \\
\text{AC} & \leftarrow \text{temp3}[4]; \quad \text{//check carry and auxiliary carry flags} \\
\text{CY} & \leftarrow \text{temp4}[8]; \\
\text{A} & \leftarrow \text{temp4}[7:0]; \quad \text{//store result from temporary register to A} \\
\text{P} & \leftarrow (^A==0) ? 1:0; \quad \text{//check parity flag} \\
\text{Z} & \leftarrow (A == 0); \quad \text{//check zero flag} \\
\text{S} & \leftarrow A[7]; \quad \text{//check sign flag} \\
\text{Pc} & \leftarrow \text{pc}+1; \quad \text{//advance pc by 1} \\
\text{state} & \leftarrow 1; \quad \text{//to execute next instruction}
\end{align*}
\]

end

With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the “opcode” is decoded. The processing of the instruction, namely, \( \text{SUB} \) \( B \) is shown in the ASM chart Fig. 4.20. The “opcode” read is 90H, for \( \text{SUB} \ B \) instruction. The contents of “B” register are subtracted from the accumulator “A”. The result is stored in the accumulator. Here all flags are affected.
The auxiliary carry flag is set if carry is generated by subtraction of $A[3:0] - [3:0]$. Zero flag is set if the operation results in 0 and reset if the result is not 0. The sign flag is set if bit $A[7]$ is 1 after execution of arithmetic operation. The parity is obtained by “exclusive or” of the bits of the result $A-B$. If the result of xor is 1 then $P=0$ else $P=1$. The sign and zero flag are set accordingly. Finally the program counter is incremented by 1 and the control loops back to state “1” to continue processing the next instruction. The ASM chart is shown in Fig. 4.20.

**Sample User Instruction in ROM**

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>64:</td>
<td>8’h90</td>
<td>SUB B</td>
<td>5 clk</td>
</tr>
</tbody>
</table>

**Timing Diagram 4.21: Simulation Result for SUB B (00-FF=01)**

Here the register “B” contents “FF” is subtracted from the contents of the accumulator i.e “00” the result is stored “A” register. This can be observed in the timing diagram 4.21. All flags are modified. As seen in the waveform the instruction SUB B is
located in 64 (PC value) commences at 5710 ns and completes at 5810 ns. Thus the processing time is 100 ns or 5 clock cycles since the clock runs at 50MHz.

---

**Figure 4.20: ASM Chart for SUB B**

---

22: **SUB M (Arithmetic Group)**

**Description: Subtract memory location contents from accumulator.**

This instruction subtracts memory location contents (whose address is given by HL) from accumulator and the result is placed in accumulator. All flags are modified to reflect the result of operation.
• Operation : \( A \_ (HL) \rightarrow A \)

At PC value 85 the instruction \( \text{SUB M} \) is stored in ROM. When this instruction is executed (refer timing diagram 4.22). The accumulator \( A = \text{D}5\text{h}, \ H = 00\text{h}, \ L = 10\text{h}, \) at memory location0010: C6 is stored and the instruction \( \text{SUB M} \) is executed.

\[
\begin{array}{c}
1011 0101 \\
1100 0110 \\
\hline
0000 1111 = 0Fh
\end{array}
\]
The result 0F is stored in accumulator.

3:begin
\[
\begin{align*}
\text{rd\_n} & \leq 1; & \text{// Don't read now.} \\
\text{case (opcode)} & \leq \text{// Execution of "opcode"}
\end{align*}
\]
8'h96:begin \text{// SUB M}
\[
\begin{align*}
\text{addr15 – 8} & \leq H; & \text{// Higher address for memory read.} \\
\text{rd\_n} & \leq 1; & \text{// Don't read as yet.} \\
\text{iom\_n} & \leq 0; & \text{// Memory access.} \\
\text{s1} & \leq 1; & \text{// Advanced status of Memory} \\
\text{s0} & \leq 0; & \text{// read operation.} \\
\text{state} & \leq 109; & \text{// Continue processing.}
\end{align*}
\]
end
109:begin \text{// execution of SUB M}
\[
\begin{align*}
\text{ale} & \leq 1; & \text{// Assert ALE signal.} \\
\text{D2} & \leq L; & \text{// Lower address for memory read.} \\
\text{state} & \leq 110; & \text{// Continue processing.}
\end{align*}
\]
end
110:begin
\[
\begin{align*}
\text{ale} & \leq 0; & \text{// disable latch} \\
\text{read\_cnt} & \leq \text{read\_cnt} + 1; & \text{// Run counter to generate the read pulse.}
\end{align*}
\]
\[
\begin{align*}
\text{if \ (read\_cnt} & \neq \text{`read\_delay\_count)} \text{begin} \\
\text{rd\_n} & \leq 0; & \text{// Enable active low read signal to fetch} \\
\text{state} & \leq 110; & \text{// Continue to count the read pulse.}
\end{align*}
\]
end
else
\[
\begin{align*}
\text{read\_cnt} & \leq 0; & \text{// Clear read pulse counter.}
\end{align*}
\]
111:begin
    temp <= ad; // Get the memory contents into temp.
    rd_n <= 1;  // De-activate read pulse.
    state <= 41; // to execute the next instruction.
end

41:begin
    temp4 <= A - temp; // subtract temp from A
    temp3 <= A[3:0] -temp[3:0]; //to check for auxiliary carry flag
    state  <= 37;  //continue processing
end

37:begin
    rd_n <= 1; //disable read and write
    wr_n <= 1;
    ale  <= 0; //disable latch
    AC <= temp3[4]; //check carry and auxiliary carry flags
    CY <= temp4[8];
    A  <= temp4[7:0]; //store result from temporary reg to A
    P  <= (^A==0) ? 1:0 ;//check parity flag
    Z  <= (A == 0);  //check zero flag
    pc <= pc+1;   //advance pc by 1
    state <= 1;  //to execute next instruction
end

With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the “opcode” is decoded. The “opcode” read is 96H, in which case the MSB bus address gets the value from H register and branches to the next state “109”. In this state, the LSB bus address i.e.D2 is derived from L register and ALE signal is activated once again in order to send the memory address over the address bus. In the next state “110”, the ALE pulse is withdrawn and read pulse is issued in order to fetch the contents of the memory addressed by HL register pair. The next state i.e.111 is the executing phase of the actual instruction SUB M.
this state, the memory contents are got into temp register and read pulse is withdrawn and branches to state 41. In this state the auxiliary carry flag is tested and the actual subtraction takes place and then branches to state 37. In this state CY, AC, Z, S and P flags are set or cleared accordingly. Finally, the program counter is incremented by 1 and the control loops back to state “1” to continue processing the next instruction. The ASM chart is shown in Fig.4.21.

Sample User Instruction in ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>85:</td>
<td>8'h96</td>
<td>SUB M</td>
<td>10 clk</td>
</tr>
</tbody>
</table>

Timing Diagram 4.22: Simulation Result for SUB M (D5-C6=0F)

Here the contents of memory C6h are subtracted from the accumulator which has a value 0Fh along with carry 0. The result is stored in accumulator which is D5h. This can be observed in the timing diagram 4.22. Here the sign flag is set to 0 (S=0) because bit 7 in accumulator is 0. Parity is set to 1 (P=1) because the number of ones is even in the result. Carry Flag is set to 0 (CY=0) since the arithmetic operation results in no carry. The auxiliary carry flag is 1 and the zero flag is set to 0 as result is not equal to zero.
Figure 4.21: ASM Chart  SUB M
23: SBB C (Arithmetic Group)

Description: Subtract register and borrow flag from accumulator

This instruction subtracts register and the borrow flag from accumulator and the result is placed in accumulator. All flags are modified to reflect the result of the subtraction. In subtraction instructions, carry flag is called a borrow flag. Examples of registers are all general purpose registers such as A, B, C, D, E, H and L.

- Operation: A-C-Borrow→A

```bash
3:begin
    rd_n <= 1; // Don't read now.
case (opcode) // Execution of "opcode"
8'h99:begin //Execute SBB C instruction
    temp4 <= A - C - CY; //subtract C and carry from A, store result in temporary register
    temp3 <= A[3:0]-C[3:0]-CY; //to check for auxiliary carry flag
    state <= 37;
37:begin
    rd_n <= 1; //disable read and write
    wr_n <= 1;
    ale <= 0; //disable latch
    AC <= temp3[4]; //check carry and auxiliary carry flags
    CY <= temp4[8];
    A <= temp4[7:0]; //store result from temporary reg to A
    P <= (^A==0) ? 1:0; //check parity flag
    Z <= (A == 0); //check zero flag
    S <= A[7]; //check sign flag
    pc <= pc+1; //advance pc by 1
    state <= 1; //to execute next instruction
end
```

With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the “opcode” is decoded. Depending upon the instruction, the FSM is taken to a group of other states in order to complete the processing of the instruction. The processing of the instruction, namely, SBB C, is shown in the flowchart 36. The “opcode” read is 99H, for SBB C instruction. The contents of “C” register are subtracted from the accumulator “A” along with the borrow. The result is stored in accumulator. Here all flags are affected. The auxiliary
carry flag is set if carry is generated by addition of $A[3:0] - C[3:0] - CY$; Zero flag
is set if the operation results in 0 and rest if the result is not 0. The sign flag is set if
bit $A[7]$ is 1 after execution of arithmetic operation. The parity is obtained by
“exclusive or” of the bits of the result $A-C-CY$. If the result of xor is 1 then $P=0$ else
$P=1$. Finally the program counter is incremented by 1 and the control loops back to
state “1” to continue processing the next instruction. The ASM chart is shown in
Fig.4.22.

Sample User Instruction in ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>70:</td>
<td>8'h99</td>
<td>SBB C</td>
<td>5 clk</td>
</tr>
</tbody>
</table>

Timing Diagram 4.23: Simulation Result for SBB C (01-55-1=AB)

Here the contents of Register C which has a value 55h are subtracted from the
accumulator which contains 01h along with carry 1. The result is stored in
accumulator which is ABh. This can be observed in the timing diagram 4.23. Here the
sign flag is set to 1 ($S=1$) because bit 7 in accumulator is 1. Parity is set to 0 ($P=0$)
because the number of ones is odd in the result. Carry Flag is set to 1 ($CY=1$) since
the arithmetic operation results in a carry. The auxiliary carry flag is 1 and the carry flag is set to 1.

Fig. 4.22 ASM Chart for SBB C
24: SBB M (Arithmetic Group)

Description: Subtract memory and borrow flag from accumulator.

This instruction subtracts memory location and the borrow flag from accumulator and the result is placed in accumulator. The address and memory location is given by HL pair. The operations are same as SBB R.

- Operation: A-(HL)-Borrow→ A

All flags are modified to reflect the result of subtraction.

3:begin
    rd_n <= 1; // Don't read now.
case (opcode)
    // Execution of "opcode"
8'h9E:begin // SBB M
    addr15_8 <= H; // Higher address for memory read.
    rd_n <= 1; // Don't read as yet.
    iom_n <= 0; // Memory access.
    s1 <= 1; // Advanced status of Memory
    s0 <= 0; // read operation.
    state <= 112; // Continue processing.
end
112:begin // execution of SBB M
    ale <= 1; // Assert ALE signal.
    D2 <= L; // Lower address for memory read.
    state <= 113; // Continue processing.
end
113:begin
    ale <= 0; // disable latch
    read_cnt <= read_cnt + 1; // Run counter to generate the read pulse.
    if (read_cnt != `read_delay_count)
      begin
        rd_n <= 0; // Enable active low read signal to fetch
        state <= 113; // Continue to count the read pulse.
      end
    else
      begin
        read_cnt <= 0; // Clear read pulse counter.
        state <= 114; // continue processing
      end
end

114:begin
   temp <= ad;  // Get the memory contents into temp.
   rd_n <= 1;  // De-activate read pulse.
   state <= 49;  // to execute the next instruction.
end

49:begin  //execute SBI A,8-bit data instruction
   temp4 <= A - temp-CY ; //subtract temp and carry from A
   temp3 <= A[3:0]-temp[3:0]-CY;//to check for auxiliary carry flag
   state <= 37;  //continue processing
end

37:begin
   rd_n <= 1;  //disable read and write
   wr_n <= 1;
   ale <= 0;  //disable latch
   AC <= temp3[4];//check carry and auxiliary carry flag
   CY<= temp4[8];
   A <= temp4[7:0]; //store result from temporary reg to A
   P <= (^A==0) ? 1:0 ;  //check parity flag
   Z<= (A == 0); //check zero flag
   pc<=pc+1;  //advance pc by 1
   state <= 1;  //to execute next instruction
end

With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the “opcode” is decoded. The “opcode” read is 9E H, in which case the MSB bus address gets the value from H register and branches to the next state “112”. In this state, the LSB bus address i.e.D2 is derived from L register and ALE signal is activated once again in order to send the memory address over the address bus. In the next state “113”, the ALE pulse is withdrawn and read pulse is issued in order to fetch the contents of the memory addressed by HL register.
pair. The next state i.e. 114 is the executing phase of the actual instruction SBB M. In this state, the memory contents are got into temp register and read pulse is withdrawn and branches to state 49. In this state the auxiliary carry flag is tested and the actual addition takes place and then branches to state 37. In this state CY, AC, Z, S and P flags are set or cleared accordingly. Finally, the program counter is incremented by 1 and the control loops back to state “1” to continue processing the next instruction. The ASM chart is shown in Fig. 4.23.

Sample User Instruction in ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>87:</td>
<td>8'h9E</td>
<td>SBB M</td>
<td>10 clk</td>
</tr>
</tbody>
</table>

Timing Diagram 4.24: Simulation Result for SBB M (D5-C6-0=0F)

Here the contents of memory C6h are subtracted from the accumulator which has a value D5h along with borrow 0. The result is stored in accumulator which is 0Fh. This can be observed in the timing diagram 4.24. Here the sign flag is set to 0 (S=0) because bit 7 in accumulator is 1. Parity is set to 1 (P=1) because the number of ones is odd in the result. Carry Flag is set to 0 (CY=0) since the arithmetic operation results
in no carry. The auxiliary carry flag is 1 and the zero flag is set to 0 as result is not equal to zero.

Figure 4.23: ASM Chart SBB M
25: SUI 01 (Arithmetic Group)

Description: Subtract immediate data from accumulator.

- Operation: A - data → A
  All flags are modified to reflect the result of operation.
  
  At pc value 12 the instruction SUI 01 is stored in ROM. When this instruction is executed (refer timing diagram 4.25)

<table>
<thead>
<tr>
<th>A=56h</th>
<th>- Data= 01h</th>
</tr>
</thead>
<tbody>
<tr>
<td>=55h</td>
<td></td>
</tr>
</tbody>
</table>

  The value 55h is stored in accumulator

  Therefore flags are also modified to reflect the result.

<table>
<thead>
<tr>
<th>CY=0</th>
<th>Z= 1</th>
<th>AC= 1</th>
<th>S= 0</th>
<th>P=1</th>
</tr>
</thead>
</table>
| 8'hD6:begin //execute SUI A, 8 BIT instruction
  pc <= pc+1; //advance pc by 1 to get the 8bit data
  state <= 38; //continue processing.
end

38:begin //execute SUI A, 8-bit data instruction
  ale <= 1; //latch enable
  addr15_8 <= pc[15:8]; //load pc contents onto address bus
  D2 <= pc[7:0];
  iom_n <= 0; //memory access
  s1 <= 1; //memory read cycle
  s0 <= 0;
  state <= 39; //continue processing
end

39:begin
  ale <= 0; //disable latch
  read_cnt <= read_cnt + 1; //Run counter to generate the read pulse.
  if (read_cnt != `read_delay_count) begin
    rd_n <= 0; //Enable active low read signal to fetch
    state <= 39; //Continue to count the read pulse.
  end
  else begin
    read_cnt <= 0; //Clear read pulse counter.
    state <= 40; //continue processing
  end

155
With the arrival of the third clock, the FSM goes to the state marked “Execute
Instruction”, where the actual processing of the instruction commences. To start with,
the read signal is withdrawn and the opcode is decoded then the program counter is
incremented by one and then branches to state 38. In this state “ale” is activated and
the address bus derives the value of MSB 8 bits of the program counter. D2 register
derives the value of LSB 8 bits of the program counter. A low “iom_n” signal
indicates that the processor has selected the memory for the access. The select signals
are s1=1 and s0=0 for memory read operation. In the next state i.e. 39 the “ale” signal
is withdrawn and the read signal is activated and branches to state 40. In this state i.e.
40 the contents of “ad” is transferred to register “temp” and then branches out to
state 41. In state 41 the contents of “temp” are subtracted from accumulator and
stored in register “temp4”. The accumulator contents i.e. A[3:0] and temp[3:0] are
subtracted to check for the auxiliary flag and then branches out to state 37. In this
state i.e. 37 the read pulse and write pulse are withdrawn. The signal “ale” is deactivated, all the flags are tested, the pc is incremented by 1 and then the control loops back to state 1. The ASM chart is shown in Fig.4.24

Sample User Instruction in ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:</td>
<td>8'hD6</td>
<td>SUI 01</td>
<td>10 clk</td>
</tr>
</tbody>
</table>

Timing Diagram 4.25: Simulation Result for SUI 01 (56h - 01h=55h)

Here the data 01h is subtracted from the accumulator which has a value 56h. The result is stored in accumulator which is 55h. This can be observed in the timing diagram 4.25. Here the sign flag is set to 0 (S=0) because bit 7 in accumulator is 0. Parity is set to 1 (P=1) because the number of ones is even in the result. Carry Flag is set to 0 (CY=0) since the arithmetic operation results in no carry. The auxiliary carry flag is 0 and the zero flag is set to 0 as result is not equal to zero. As seen in the waveform the instruction SUI 01h is located in 12 (pc value) commences at 549810 ns and completes at 550010 ns. Thus the processing time is 200 ns or 10 clock cycles since the clock runs at 50MHz.
Figure 4.24: ASM Chart SUI 01
26: SBI FF (Arithmetic Group)

Description: Subtract immediate data and borrow flag from accumulator.

- Operation: \( A - \text{data} - \text{borrow} \rightarrow A \)

All flags are modified to reflect the result of operation.

At pc value 656 the instruction SBI FF is stored in ROM. When this instruction is executed (refer timing diagram 4.26)

\[
\begin{align*}
A &= \text{FFh} \\
- \text{Data} &= \text{FFh} \\
&= 00h - 01 = \text{FF}
\end{align*}
\]

The value FFh is stored in accumulator.

3: begin
    rd_n <= 1;  // Don't read now.
    case (opcode)  // Execution of "opcode"
    8'hDE: begin
        pc    <= pc+1;  // SBI #data
        state <= 46;
    end
46: begin  // Continue SBI A,#data instruction.
    ale    <= 1;
    addr15_8 <= pc[15:8];
    D2     <= pc [7:0];
    iom_n  <= 0;
    s1     <= 1;
    s0     <= 0;
    state  <= 47;
end
47: begin
    ale    <= 0;
    rd_n   <= 0;
    state  <= 48;
end
48: begin
    rd_n   <= 1;
    \{CY, A\} <= A-ad-CY// Subtract and Set the carry flag.
    Z      <= ((A-ad-CY) == 0) ? 1:0;  // Set the zero flag.
    \{S, temp4[6:0]\} <= A-ad-CY; // Set the sign flag - temp4 is dummy.
    P      <= (^ (A-ad-CY) == 0) ? 1:0; // Compute the parity.
    pc     <= pc+1;  // Advance the PC.
    state  <= 1;
end

With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with,
the read signal is withdrawn and the “opcode” is decoded and branches to the next state 46. In this state “ale” is activated and the address bus derives the value of MSB 8 bits of the program counter. D2 register derives the value of LSB 8 bits of the program counter. A low “iom_n” signal indicates that the processor has selected the memory for the access. The select signals are s1=1 and s0=0 for memory read operation and branches to state 47. In this state the “ale” signal is withdrawn and the “rd_n” signal is asserted to fetch the data and then branches out to state 48. In this state the actual arithmetic operation takes place and the flags are accordingly set/reset depending on the result.

Timing Diagram 4.26: Simulation Result for SBI FF (FFh – FFh -1 =FFh)

Here the data FFh is subtracted from the accumulator which has a value FFh along with borrow. The result is stored in accumulator which is FFh. This can be observed in the timing diagram 4.26. Here the sign flag is set to 1 (S=1) because bit 7 in accumulator is 1. Parity is set to 1 (P=1) because the number of ones is even in the result. Carry Flag is set to1 (CY=1) since the arithmetic operation results in no carry. The auxiliary carry flag is 1 and the zero flag is set to 0 as result is not equal to zero.
As seen in the waveform, the instruction SBI FFh is located in 656 (pc value) and commences at 549810 ns and completes at 550010 ns. The instruction takes 7 clock cycles.

Figure 4.25: ASM Chart for SBI FF
27: INR C (Arithmetic Group)
Description: Increment register contents by one.

This instruction increments the contents of the specified register by one and the result is stored in the same register. The examples of registers are all general purpose registers such as A, B, C, D, E, H and L. Only carry flag is not modified, all other flags are modified.

3: begin
    rd_n <= 1; // Don't read now.
case (opcode)
    // Execution of "opcode"
     8'h0C:begin
        // execute INR C instruction
        C <= C + 1; // increment C by 1
        temp3 <= C[3:0] + 1; // to check for auxiliary carry flag
        state <= 52; // continue processing.
    end
52:begin
    // execution of INC/DEC C
    rd_n <= 1;
    // disable read and write
    wr_n <= 1;
    ale <= 0;
    AC <= (temp3[4] == 1) ? 1:0; // check for all flags
    Z <= (C == 0)? 1:0; // check zero flag
    S <= (C[7] == 1) ? 1:0; // check sign flag
    pc <= pc+1; // advance pc by 1
    state <= 1; // to execute next instruction
end

With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the “opcode” is decoded and the contents of register “C” are incremented by 1 and tested for auxiliary flag and branches to state 52. Here the read and write signal are withdrawn and the remaining flags are tested, the
program counter is incremented by 1 and the control loops back to state one. The ASM chart is shown in Fig. 4.26

**Sample User Instruction in ROM**

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>72:</td>
<td>8'h0C</td>
<td>INC C</td>
<td>5 clk</td>
</tr>
</tbody>
</table>

Here the data 55h in register “C” is incremented to 56h and stored in the accumulator. This can be observed in the timing diagram 4.27.
28:INX   B (Arithmetic Group)

Description: This instruction increments the contents of register pair by one and the result is stored in the same register pair. Examples of register pair are BC.DE.HL and SP. Only higher order register is specified for register pair in instruction. No flags are modified.

3: begin
    rd_n <= 1;  // Don't read now.
    case (opcode)  // Execution of "opcode"
    begin
        Reset: rd_n=wr_n=1, ale=0, iom_n=0, pc=0, addr=0
        Assert ALE: ale=1, add[15:0]=pc[15:8], ad
        Opcode Fetch: ale=0, rd_n=0, opcode=ad
        Execute Instruction: rd_n=1, opcode<=8'h0C
            C=C+1, temp3=C[3:0]  
            rd_n=wr_n=1, ale=0, AC=(temp3[4]==1)?1:0
            Z=(C==0)?1:0
            S=(c[7]==1)?1:0
            Pc=pc+1
    end
end
8'h03:begin
   //execute INX B instruction
   pc <= pc + 1;  //advance pc by 1
   {B,C} <= {B,C} + 1; //increment BC pair contents by 1
   state <= 1; //to execute next instruction
end

With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the “opcode” is decoded and the contents of register pair “BC” are incremented by 1 and the program counter is incremented by one.

**Sample User Instruction in ROM**

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>75:</td>
<td>8'h03</td>
<td>INX B</td>
<td>4 clk</td>
</tr>
</tbody>
</table>

Timing Diagram 4.28: Simulation Result for INX  B

From the timing diagram we can observe that the contents of register pair are incremented by one. The value of register pair “BC” before execution is 56 56 and after execution it is incremented by one and the register pair “BC” has 56 57.
Description: This instruction decrements the contents of register pair by one and the result is stored in the same register pair. Examples of register pair are BC,DE,HL and SP. Only higher order register is specified for register pair in instruction. No flags are modified.

```
3: begin
    rd_n <= 1;  // Don't read now.
    case (opcode)
        8'h03:begin     // execute INX B
            pc <= pc+1;  // advance pc by 1
            {H,L} <= {H,L} - 1;  // decrement HL pair contents by 1
            state <= 1;  // to execute next instruction
    end
```

Figure 4.27: ASM Chart for INX B C

29:DCX H (Arithmetic Group)
With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the “opcode” is decoded and the contents of register pair “HL” are decremented by 1 and the program counter is incremented by one.

### Sample User Instruction in ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>38:</td>
<td>8'h2B</td>
<td>DCX H</td>
<td>4 clk</td>
</tr>
</tbody>
</table>

### Timing Diagram 4.29: Simulation Result for DCX H

From the timing diagram we can observe that the contents of register pair are decremented by one. The value of register pair “HL” before execution is 00 35 and after execution it is decremented by one and the register pair “HL” has 00 34.
30: DAA [Arithmetic Group]

Description: The contents of the accumulator are changed from a binary value to its equivalent two, 4 bit binary coded decimal i.e. BCD number. This is the only instruction that uses auxiliary carry flag to perform the operation of binary to BCD conversion.

DAA instructing working. The instruction checks the following conditions:

1: If the value of the lower order 4 bits D3-D0 in the accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low order 4 bits of accumulator.

2: If the value of the high order 4 bits D7-D4 in the accumulator is greater than 9 or if CY flag is set, the instruction adds 6 to the higher 4 bits of accumulator.

DAA instruction is used with add instructions ADD, ADI etc. to perform addition of
number in BCD. The add instruction adds the two BCD numbers in hexadecimal format and DAA instruction converts this hexadecimal result to BCD format.

*Note:* The restriction on DAA is condition of flags. If you execute any instruction which affects the flags CY and AC and then if we try to adjust accumulator to BCD format the result will be wrong as instruction will use AC and CY flags of previous instructions to take decisions.

**Example:** The sample ROM contains

```plaintext
42: ad <= 8'h06; //MVI B,99h  (8 clock cycles)
43: ad <= 8'h99;
44: ad <= 8'h78; //MOV A,B  (4 clock cycles)
45: ad <= 8'h80; //ADD B    (5 clock cycles)
46: ad <= 8'h27; //DAA      (5 clock cycles)
```

When we add 99 and 99 and want the result in BCD form. The following steps should be implemented.

```
1001 1001 99
+ 1001 1001 99
---------
CY=1 0011 0010 32
```

The answer is 32 in hexadecimal form. When DAA is executed it checks the above two conditions. Since the AC flag is set 6 is added to low order 4 bits. Contents of accumulator are converted from binary to equivalent BCD form.

```
0011 0010
+ 0110
---------
0011 1000
```

Result = 38 in BCD form and is stored in accumulator. This can be observed in timing diagram 34.

3: begin
```
rd_n <= 1;  // Don't read now.
case (opcode)  // Execution of "opcode"
8'h27:begin  //execute DAA instruction
  //check lower nibble data and continue processing.
  state <= 59;
end
59:begin  //execution of DAA instruction
  //check for higher nibble
  pc <= pc+1;  //advance pc by 1
end
```

169
With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the “opcode” is decoded. In this state the lower nibble data is tested for auxillary carry flag and if it is true the data is incremented by 6. And branches to state 59. In this state the carry flag is tested if the carry is generated the higher nibble is incremented by 6 and branches to state one. The ASM chart is shown in Fig.4.29.

Sample User Instruction in ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>46:</td>
<td>8’h27</td>
<td>DAA</td>
<td>5 clk</td>
</tr>
</tbody>
</table>

Timing Diagram 4.30: Simulation Result for DAA

From the timing diagram we can observe that the two hexadecimal numbers i.e. 99h + 99h are added which results in 32h along with CY=1. Since the auxillary carry flag is set the the lower nibble is added with 6. This results in the BCD value equal to 38 which is shown in the timing diagram 35. The instruction takes five clock cycles.
4.4.3 Algorithms for Logical Group

31:ANA B (Logical Group)

Description: Logically AND register with accumulator.

The contents of the specified register are logically ANDed with the contents of accumulator and the result is placed in the accumulator. The operation is performed bit by bit i.e. D0 bit of accumulator is ANDed with bit D0 of register “B” and so on up to D7 bit of accumulator ANDed with D7 bit of register. Examples of registers are all general purpose registers such as A, B, C, D, E, H and L.
• Operation: $A \text{ AND } B \rightarrow A$

• Flags: $S, Z, P$ are modified to reflect the result of operation. Carry flag is reset and AC is set.

**Description of Algorithm:**

With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the “opcode” is decoded. In this state the contents of register “B” are logically ANDed with the contents of accumulator. The carry flag is reset and the auxiliary flag is set. The other flags are modified to reflect the result of operation. The ASM chart is shown in Fig. 4.30.

**Sample User Instruction in ROM**

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>665:</td>
<td>8'hA0</td>
<td>ANA B</td>
<td>4 clk</td>
</tr>
</tbody>
</table>

**Timing Diagram 4.31: Simulation Result for ANA B**
From the timing diagram we can observe that the two hexadecimal numbers i.e. FFh and 00h are logically ANDed which results in 00h. The value 00h is stored in accumulator The instruction takes four clock cycles.

**Figure 4.30: ASM Chart for ANA B**

32: ORI FF (Logical Group)

**Description: Logically OR immediate data with accumulator.**

The contents of accumulator are ORed with 8 bit data specified along with the instruction and the result is stored in accumulator.

**Description of Algorithm:**

With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the opcode is decoded then the program counter is incremented by one and then branches to state 29. In this state “ale” is activated and
the address bus derives the value of MSB 8 bits of the program counter. D2 register derives the value of LSB 8 bits of the program counter. A low “iom_n” signal indicates that the processor has selected the memory for the access. The select signals are s1=1 and s0=0 for memory read operation. In the next state i.e. 30 the “ale” signal is withdrawn and the read signal is activated and branches to state 31. In this state i.e. 31 the read signal is deactivated and the contents of accumulator are logically ored with the data. The carry and auxiliary carry flags are reset. The parity, sign and zero flag are modified to reflect the result then the pc is incremented by 1 and the control loops back to state 1. The ASM chart is shown in Fig. 4.31

Sample User Instruction in ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>768:</td>
<td>8’H6</td>
<td>ORI FF</td>
<td>4 clk</td>
</tr>
</tbody>
</table>

Timing Diagram 4.32: Simulation Result for ORI FF
From the timing diagram we can observe that the two hexadecimal numbers i.e. FFh and AAh are logically ORed which results in FFh. The value FFh is stored in accumulator. The instruction takes seven clock cycles.

**Figure 4.31 : ASM Chart ORI FF**
33: RLC (Logical Group)

Description: Rotate Accumulator left.

This instruction rotates the contents of accumulator left by 1 bit i.e. it shifts the bits left by one position. D0 will be transferred to D1, D1 to D2, and so on D6 to D7, D7 to D0 as well as to the carry flag.

Operation: Accumulator

\[ D_0 \rightarrow D_{n+1} \quad (n=0 \text{ to } 6) \]
\[ D7 \rightarrow D0, \quad D7 \rightarrow CY \]

Here only carry flag is modified. D7 bit is copied to carry flag. No other flags are modified. This can be observed in the timing diagram 4.33.

With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the opcode is decoded then the carry flag is modified by storing the bit D7. The contents of accumulator are rotated left by concatenating the contents bits \( A[6:0] \) and \( A[7] \). Further the contents of program counter are incremented by one. The ASM chart is shown in Fig. 4.32.

Sample User Instruction in ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>811:</td>
<td>8’h07</td>
<td>RLC</td>
<td>4 clk</td>
</tr>
</tbody>
</table>

Timing Diagram 4.33: Simulation Result for RLC
**34: CMP B (Logical Group)**

**Description:** Compare register with accumulator.

This instruction compares the register contents with accumulator. The operation of comparing is performed by subtracting register from accumulator. The contents of register or accumulator are not altered. The result of comparison is indicated by setting flags as follows:

A > B: CY flag is Reset and Zero flag is Reset.
A = B: Zero flag is Set.
A < B: CY is Set.

The S, P, AC flags are modified to reflect the result of operation.

With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the opcode is decoded. The flags are set or reset according to the algorithm described in the ASM chart. If A=B zero flag is set, A>B.
carry flag and zero flag is reset, and if A<B then carry flag is set. The ASM chart is shown in Fig. 4.33.

Sample User Instruction in ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>777:</td>
<td>8`Hb8</td>
<td>CMP B</td>
<td>4 clk</td>
</tr>
</tbody>
</table>

Timing Diagram 4.34: Simulation Result for CMP B

Figure 4.33: ASM Chart for CMP B
4.4.4 Algorithms for Stack Group

35: PUSH B (Stack Control)

Description: Push the contents of register pair on to stack.

When the instruction is executed the contents of the register pair are copied onto the stack in the following sequence: The stack pointer is decremented by one and the contents of higher order register of register pair are copied into that location. The stack pointer is again decremented by one and the contents of lower order register of register pair are copied at that location. The examples of register pair are BC reg. pair, HL reg. pair, DE reg. pair.

Operation:

\[
\begin{align*}
\text{Lower order } R_p & \rightarrow (SP - 2) \\
\text{Higher order } R_p & \rightarrow (SP - 1) \quad SP-2 \rightarrow SP
\end{align*}
\]

No flags are modified.

The description of the algorithm for the instruction is as follows. With the arrival of the third clock the opcode is fetched from the ROM i.e. “8'hC5” and branches to state 350. In this state the read and write signals are withdrawn and the higher order address of “SP” is sent to higher order address bus i.e. addr15_8 and branches to state 351. In this state the signal “ale” is asserted, the lower order address of SP is sent to reg “D2” and branches to state 352.

Here the contents of higher order register “B” are written on to stack and the write signal “wr_n” is enabled for the write operation and branches to state 353. In this state the write signal is withdrawn and the stack pointer is decremented by 1. In the next state i.e. 354 the process repeats for getting the contents of lower order register “C”. In state 357 the write signal is withdrawn and the control loops back to state 1.
### Sample User Instruction in ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>98:</td>
<td>8’Hc5</td>
<td>PUSH B</td>
<td>14 clk</td>
</tr>
</tbody>
</table>

**Timing Diagram 4.35: Simulation Result for PUSH B**

From the timing diagram we can observe that the instruction requires three machine cycles. State 0 to 3 is one machine cycle and state 350 to 353 is the second machine cycle and from state 354 to 357 is the third machine cycle. Here the contents of the register pair “BC” i.e. B=ce, and C=ce are stored in stack memory defined in the code by the user. The stack is a group of memory locations in RAM. The instruction takes 14 clock cycles. Here the stack pointer is initialized at 5350 which can be observed in the timing diagram.
Figure 4.34: ASM Chart for PUSH B
4.4.5 Algorithms for Machine control group

Interrupt Logic Control Instructions

The processor designed here provides four instructions to control interrupt logic: These are

1: EI (Enable Interrupt)
2: DI (Disable Interrupt)
3: SIM (Set Interrupt Mask)
4: RIM (Read Interrupt Mask)

36: EI – (Enable Interrupt)

Description: It is used to enable all maskable interrupts. This instruction sets an interrupt bit of interrupt control logic.

Operation: 1→IE. Here no flags are modified.

```
8'hFB:begin          // EI instruction
    IE <= 1;              // Set IE bit.
    pc <= pc+1;           // Advance pc by 1 and return to
    state <= 1;           // execute the next instruction.
end
```

Here the algorithm is very simple. With the arrival of the third clock the read signal is withdrawn and the opcode is decoded. The interrupt enable bit is set and the program counter is incremented by one and the control loops back state 1.

Sample User Instruction in ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Instruction</th>
<th>Processing Time in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>835:</td>
<td>8'Hfb</td>
<td>EI</td>
<td>4 clk</td>
</tr>
</tbody>
</table>
Timing Diagram 4.36: Simulation Result for EI (Enable Interrupt)

As seen from the timing instruction EI is located in 835 (PC value). commences at 298945 ns and completes at 298985 ns. The instruction takes 4 clock cycles. The bit IE is set when this instruction is executed.

4.4.6 : Algorithm for Input/Output Instruction

I/O Mapped I/O or Peripheral Mapped I/O

In a microprocessor based system, data is input from external device through input device. The data is output from microprocessor to external device through output device. Generally input device is buffer and output device is latch. In our design the microprocessor has a separate 8 bit addressing scheme i.e. I/O space for I/O devices. I/O address ranges from 00h to 7Fh for INPUT and from 80h to FFh for output. If I/O device is mapped into I/O address, IN and
OUT instructions are used to communicate with I/O device. Here in our design we have created the input and output devices as ROM and RAM.

37: IN 7F

Description: Input data to accumulator from a port with 8 bit address.

This instruction takes data from input port and stores in accumulator. The address of port is 8 bits and is specified along with the instruction. The range of addresses will be 00 to 7F.

Operation: (Port address) → A

Here no flags are modified.

When IN 7F instruction is executed, the microprocessor reads data from input port (INPUTS DESIGN). The address of input port is given with the instruction i.e. 7Fh. The data is read from input port (i.e. INPUTS DESIGN) and transferred to accumulator. To read data from input device the processor sends address on higher order address bus, it will be decoded and used to select one I/O device. Then the microprocessor generates the signal “iom_n” =1 and the signal “rd_n”=0. When the peripheral gets the select signal and “iom_n” =1 signal the peripheral will send the data. This data is accepted by the processor. This data is then transferred to accumulator. This can be observed in the timing diagram 4.37.

The ASM chart for the instruction is given in Fig. 4.35.
Timing Diagram 4.37: Simulation Result for IN 7F

With reference to the timing diagram the storing format of this IN 7Fh instruction is (i) Opcode for IN and (ii) 7Fh Operand (7Fh is the address of I/O device). The sequence of events occurring can well be seen in the timing diagram 4.37. First opcode “db” is fetched from memory i.e. ROM and decodes it, then it knows that the next byte after that is an operand so that the next machine cycle will be operand fetch to take port address. This address is stored in the temporary register “D2” and the microprocessor starts next machine cycle i.e. I/O read. In I/O read machine cycle, the address is given by temporary register. It is an 8 bit address so same contents are transferred on address bus and the signal “ale” is made high. The status of iom_n line will be high, as it is I/O operation. All other operations are same as memory read operations.
Figure 4.35: ASM Chart IN 7F
39: OUT 80h

Description: Output data to output port from accumulator.

This instruction is used to transfer data to an output device. The 8 bit address of output device is specified along with instruction. The contents of accumulator are transferred to the output device. Here the output device is a RAM. To send data to an output device the microprocessor sends, an 8 bit address which will be decoded and used to select one of the peripherals, with this address the signal “wr_n” control signal and data is required.

Operation: A → (Port address)

No flags are affected

With the arrival of the third clock, the FSM goes to the state marked “Execute Instruction”, where the actual processing of the instruction commences. To start with, the read signal is withdrawn and the opcode is decoded followed by incrementing the program counter and the branches to state 406. In this state i.e. 406 the signal “ale” is activated the higher order address bus derives the value of MSB 8 bits of the program counter. D2 register derives the value of LSB 8 bits of the program counter and branches to state 407.

In this state the signal “ale” is deactivated and the read signal is asserted low to fetch the operand to read port address and branches to state 408. In this state the data is sent on higher order address bus and iom_n is asserted high for I/O access and branches to state 409. In this state the write signal is asserted low and the contents of accumulator are written to the register.
D2 and branches to state 410. In this state the signal “iom_n” is initialized to 0 and the pc value is incremented by one. From here the control loops back to state 1. The ASM chart is shown in Fig. 4.36.

**Timing Diagram 4.38: Simulation Result for OUT 80**

With reference to the timing diagram the storing format of OUT 80h instruction is (1) Opcode for OUT and (2) 80h operand ( 80h is the address of I/O device). The sequence of events occurring can be well seen in timing diagram 4.38. To start with the opcode is fetched and decoded. Then the next byte is the operand fetch to read the port address. We can observe that the contents of accumulator are transferred to the output which is similar to RAM module.
Figure 4.36: ASM Chart OUT 80
4.4.7 Comparison of processing speed of present implementation of ESP Control System with existing ESP system

All the processor instructions have been coded. Table 4.1 gives the summary of the instructions realized along with the processing times. The table also compares the processing speed of the FPGA implementation of ESP Controller System with the existing ESP systems. Most of the instructions maintain the same processing clock cycles in the two cases. The FPGA implementation scores over the existing system such as SPHL, DAD, JNZ, HLT and PCHL, whereas it is the other way round for stack operations. As can be seen from the table, the FPGA implementation is faster than the existing systems by over 30 times. However, the memory utilization is exactly the same in both the cases.

Table 4.1 Comparison of processing speed of present implementation of ESP Control systems with existing ESP system

<table>
<thead>
<tr>
<th>Sample Instructions</th>
<th>Proposed FPGA Implementation of ESPC System</th>
<th>Existing ESPC Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No. of clock cycles</td>
<td>Processing speed in nanoseconds working @ 100MHz</td>
</tr>
<tr>
<td>MOV reg, reg; XCHG / SPHL</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>MVI reg, data; ANI/CPI data</td>
<td>7</td>
<td>70</td>
</tr>
<tr>
<td>LXI rp, 16 bit data</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>LXI rp, 16 bit data</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>ADD/ADC/SBB reg / DAD rp</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>CMA; STC; CMC; DAA</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>INR/DCR reg / INX/DCX rp</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>ANA reg; RLC/RAL;RAR/RRC</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>MOV reg, M; XRA M;</td>
<td>7</td>
<td>70</td>
</tr>
<tr>
<td>CMP reg</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>Instruction</td>
<td>Addr1</td>
<td>Addr2</td>
</tr>
<tr>
<td>-------------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>JMP</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>JNZ</td>
<td>10/4</td>
<td>100/40</td>
</tr>
<tr>
<td>CALL</td>
<td>20</td>
<td>200</td>
</tr>
<tr>
<td>RET</td>
<td>12</td>
<td>120</td>
</tr>
<tr>
<td>PUSH</td>
<td>14</td>
<td>140</td>
</tr>
<tr>
<td>POP</td>
<td>12</td>
<td>120</td>
</tr>
</tbody>
</table>

References:


http://opencores.org/project/cpu8080


