BME328 LAB7

Simple Processor

35 Marks
(2 weeks) **Due Date:** Week 12

Objective:

-Design and implementation of simple 8 bit microprocessor

-The processor consists of ALU, Registers to store data, Control unit to execute instructions pointed to by PC

Organization:

1-8 bit Input data is entered through switches SW7..SW0 and stored in register R1 when instruction LD-R1 is executed

2- 8 bit ALU perform operation on input A and input B based on OP code of executed instruction

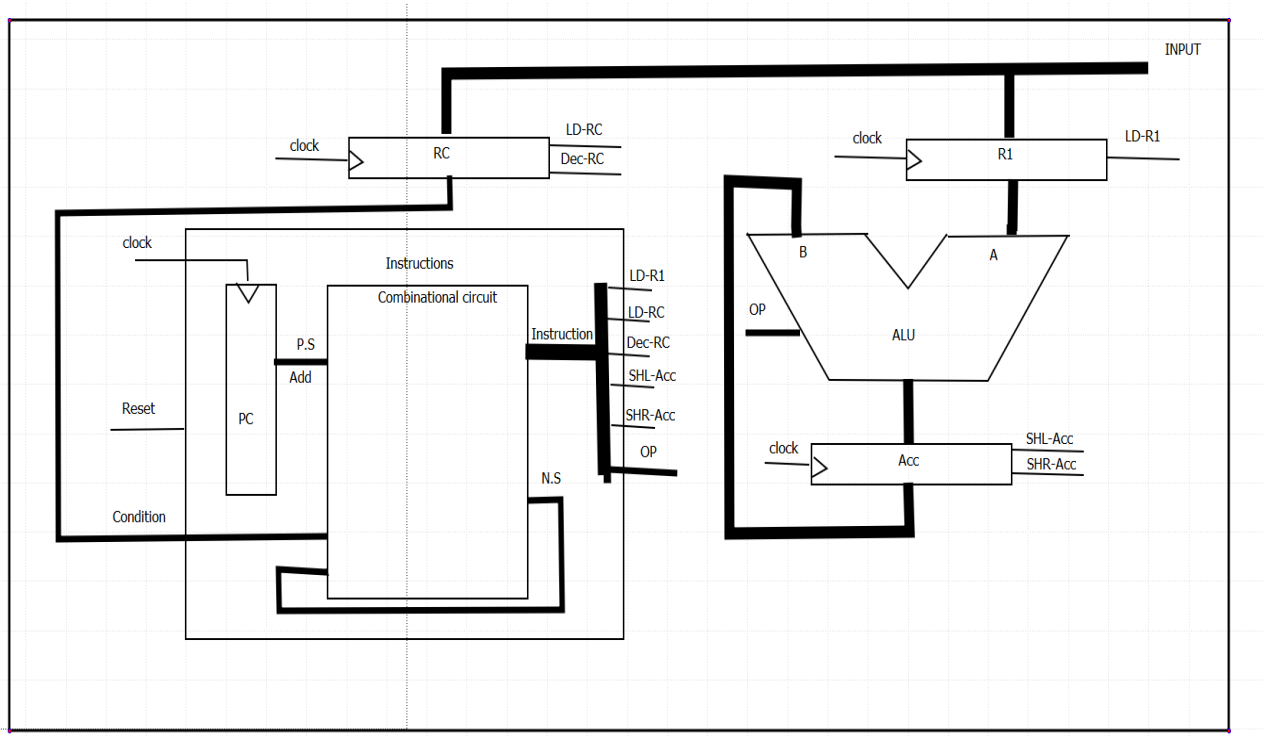
3-ALU input A is connected to R1 , input B is connected to Accumulator AC .

4-Results of each operation is displayed on two 7 Seg display unit connected to output of AC

5-RC register stores data from input switches SW7..SW0 when LD-RC instruction is executed and RC output is used as a conditional register for control flow

6-PC is a counter that is used to point to next instruction to be executed and starts from Instruction 0 to Instruction N. Each clock cycle it increment PC to point to next instruction

7-Combinational circuit stores the Instructions starting from address 0 to address N. Each instruction has its op code to be used by ALU.



SIMPLE PROCESSOR Organization

Part1: Instruction Set

1-Register Transfer	**Op Code**
LD-R1: Load R1 from input; R1=INPUT	**0**
LD-RC: Load RC from input; RC= INPUT	**1**

2-Arithmetic Operations:

ADDA: Add R1 to Acc; Acc= Acc + R1	**2**

SUBA: Sub R1 from Acc; Acc= ACC-R1 **3**

INCA: Increment Acc; Acc=Acc + 1 **4**

DECA: Dec Acc; Acc= Acc -1 **5**

DEC-RC: Dec RC; RC= RC-1 **6**

3-Logic Operations

ANDA: And Acc with R1; Acc=Acc&R1 **7**

ORA: Or Acc with R1; Acc = Acc | R1 **8**

XORA: XOR Acc with R1; Acc XOR R1 **9**

NAND: Invert Acc; Acc = Acc NAND R1 **A**

SLA: Shift Left Acc; Acc=Acc*2 **B**

SRA: Shift Right Acc; Acc = Acc/2 **C**

4-Control flow

BNEQZ: branch to start if RC !=0; goto state **D**

Part2: Implementation of Control Unit and Instructions

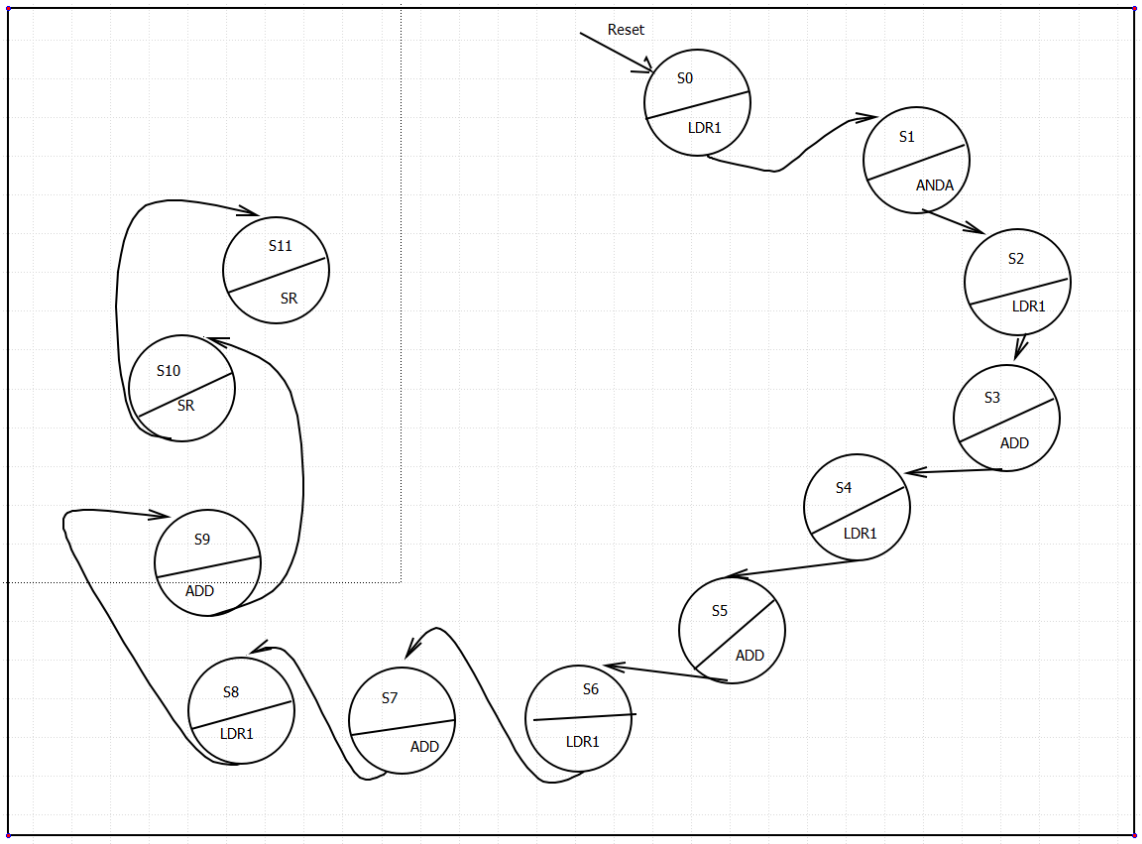
Using FSM to implement PC, Instruction Memory as combinational circuit

PC starts from S0 on Reset= active

On S0, first instruction will be executed so if first instruction to load a number in R1, then opcode=1, if need to clear Acc, Load R1=0, then use And instruction so next state op code is 8,..

Example:

Average of 4 numbers entered from input x1, x2, x3, x4
LDR1 0; R1 =0
ANDA; Acc=0
LDR1 X1; R1=X1
ADD ; Acc=X1
LDR1 X2; R1=X2
ADD; Acc= X1+X2
LDR1 X3; R1=X3
ADD ; Acc=X1+X2+X3
LDR1 X4; R1=X4
ADD; ACC=X1+X2+X3+X4
SR;
SR; Acc= (X1+X2+X3+X4)/4



Implementation of FSM for PC and Instructions

Example2:
Executing Loops
for(i=5; i<=0; i--){result=result + x}

```
        LDRC  5,
        LDR1 0
        ANDA
        LDR1 X
LOOP: ADD
        DECRC
        BNEQZ LOOP
```

VHDL FOR ALU

```
Port (
  A, B     : in  STD_LOGIC_VECTOR(7 downto 0);  -- 2 inputs 8-bit
    ALU_Opcode  : in  STD_LOGIC_VECTOR(3 downto 0);  -- 1 input
4-bit for selecting function
    ALU_Out   : out  STD_LOGIC_VECTOR(7 downto 0); -- 1 output
8-bit
    Carryout : out std_logic        -- Carryout flag
    );
end ALU;
architecture Behavioral of ALU is

signal ALU_Result : std_logic_vector (7 downto 0);
signal tmp: std_logic_vector (8 downto 0);

begin
   process(A,B,ALU_Opcode)
 begin
  case(ALU_ Opcode) is
   when "0010" => --ADD
   ALU_Result <= A + B ;
  when "0011" => -- Subtraction
```

```
   ALU_Result <= A - B ;
   when "0100" => --INCA
   ALU_Result <= A + 1 ;
  when "0101" => -- DECA
   ALU_Result <= A - 1;
 when "0111" => -- Logical and
   ALU_Result <= A and B;
  when "1000" => -- Logical or
   ALU_Result <= A or B;
  when "1001" => -- Logical xor
   ALU_Result <= A xor B;
  when "1010" => -- Logical nand
   ALU_Result <= A nand B;
  when "1011" => -- Logical shift left
   ALU_Result <= std_logic_vector(unsigned(A) sll 1);
  when "1100" => -- Logical shift right
   ALU_Result <= std_logic_vector(unsigned(A) srl 1);

  end case;
 end process;
 ALU_Out <= ALU_Result; -- ALU out
 tmp <= ('0' & A) + ('0' & B);
 Carryout <= tmp(8); -- Carryout flag
end Behavioral;
```