Lab Report No. 08:

Complete Processor

By:

Jiaqing Wan(16), Jiaxin Zheng(19)

Instructor: Dr. Xi Zhou

Classnumber: ZM1501

NAU ID:

5358836, 5358840

December 15, 2017

I. INTRODUCTION

A. Objectives

In this lab, we will fully integrate, test, and demonstrate all parts of the microprocessor. The student will complete the design of the control unit for the fetch and execution of all instructions. Testing will be by simulation on ModelSim and by board demonstration.

B. Background knowledge

2.1 What the procedures to fetch and execute one instruction in our 8-bit microprocessor? Using one example to help explain it.

Solution:

First, fetch upper byte 0x02 whose address is 0x00 from RAM and storing it into the IRU with incrementing PC. Then, fetch lower byte 0xE7 whose address is 0x01 from RAM and storing it into the IRL with incrementing PC. Finally, execute LOADI which means load 0xE7 to AC.

2.2 What is state machine? How many states are there in the state machine built in this lab?

Solution:

State machine is composed of state register and combinational logic circuit. It can transfer state according to control signal and preset state. It is a control center that coordinates related signal action and completes specific operation. There are 10 states in the state machine built in this lab.

2.3 Draw the state diagram of the state machine you built in this lab.

Solution:

We create 10 state in this lab, the control signal is clk and opcode. The state diagram as follow–

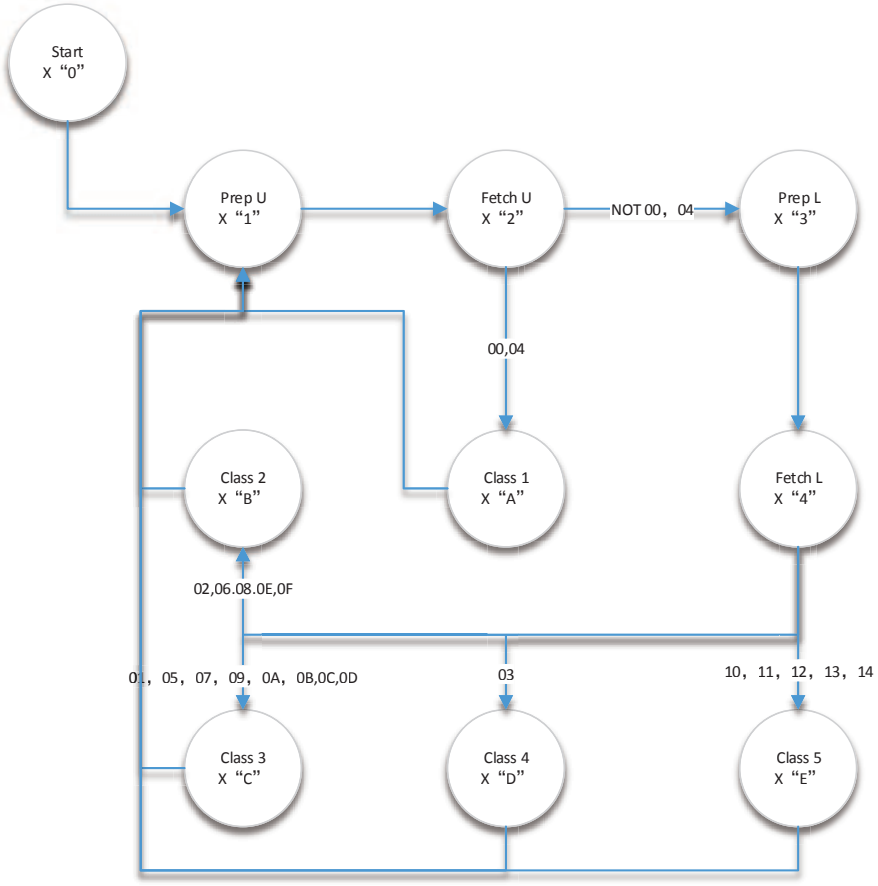

FIG. I.1 state machine of the project

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
000	02	E7	03	10	09	10	10	0A	.....
008	0E	03	04	FF	00	00	00	00	.....
010	00	00	00	00	00	00	00	00	.....

FIG. I.2 The initial of RAM

C. Simulation strategy

State Name	State Code	Description/Action	Next State
Start	**0**	Immediately on RESET	**PrepU**
		No action	
PrepU	**1**	Prepare for upper byte instruction fetch	**FetchU**
		MAR ⟵ PC	
FetchU	**2**	Fetch upper byte of instruction	**PrepL**
		IRU⟵ MDR, PC⟵PC+1	
PrepL	**3**	Prepare for lower byte instruction fetch	**FetchL**
		MAR⟵ PC	
FetchL	**4**	Fetch lower byte of instruction	**Class 2**
		IRL⟵MDR, PC⟵PC+1	
Class 2	**B**	Load the Accumulator	**PrepU**
		Load AC	
PrepU	**1**	Prepare for upper byte instruction fetch	**FetchU**
		MAR ⟵ PC	
FetchU	**2**	Fetch upper byte of instruction	**PrepL**
		IRU⟵ MDR, PC⟵PC+1	
PrepL	**3**	Prepare for lower byte instruction fetch	**FetchL**
		MAR⟵ PC	
FetchL	**4**	Fetch lower byte of instruction	**Class 4**
		IRL⟵MDR, PC⟵PC+1	
Class 4	**D**	Write AC into memory at IRL addr,Execute STORE 0x10	**PrepU**
		Load AC,STORE_MEM set	
PrepU	**1**	Prepare for upper byte instruction fetch	**FetchU**
		MAR⟵PC	
FetchU	**2**	Fetch upper byte of instruction	**PrepL**
		IRU⟵MDR, PC⟵PC+1	
PrepL	**3**	Prepare for lower byte instruction fetch	**FetchL**
		MAR⟵PC	
FetchL	**4**	Fetch lower byte of instruction	**Class 3**
		IRL⟵MDR, PC⟵PC+1	
Class 3	**C**	Memory data load to MDR[address], then through ALU	**Class 3**
		Load IRLSTORE_MEM set	
Class 3	**C**	Save result	**PrepU**
		Load AC	
PrepU	**1**	Prepare for upper byte instruction fetch	**FetchU**
		MAR ⟵ PC	
FetchU	**2**	Fetch upper byte of instruction	**PrepL**
		IRU⟵MDR, PC⟵PC+1	
PrepL	**3**	Prepare for lower byte instruction fetch	**FetchL**
		MAR⟵PC	
FetchL	**4**	Fetch lower byte of instruction	**Class 5**
		IRL⟵MDR, PC⟵PC+1	
Class 5	**E**	Load the Program Counter,Execute 0x0A	**PrepU**
		Load AC,Load PC STORE_MEM set	
PrepU	**1**	Prepare for upper byte instruction fetch	**FetchU**
		MAR⟵PC	
FetchU	**2**	Fetch upper byte of instruction	**Class 1**
		IRU⟵MDR, PC⟵PC+1	
Class 1	**A**	No action	**PrepU**
		PC+1	

D. Demonstration strategy

Input		
Change Mode	RESET	CLK
KYE[3]	KEY[2]	KEY[0]

output Mode 0	
State	HEX5HEX4
IRU Opcode	HEX3HEX2
IRL addr vlaue	HEX1HEX0
Mode	LEDR[8]
PC	LEDR[7..0]
Output Mode 1	
address	HEX5HEX4
MDR	HEX3HEX2
AC	HEX1HEX0
Mode	LEDR[8]
LOAD_AC	LEDR[6]
LOAD_IRU	LEDR[5]
LOAD_IRL	LEDR[4]
LOAD_PC	LEDR[3]
INCR_PC	LEDR[2]
FETCH	LEDR[1]
STORE_MEM	LEDR[0]

E. Contribution of each team member

Jiaxin Zheng: Write code.

Jiaqing Wan: Do simulation and write report.

II. LAB DETAILS

A. Instructor questions in the lab instruction

Have answered in the Background Knowledge.

B. VHDL code

CU.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
   entity CU is port(
         opcode    :     in std_logic_vector (7 downto 0);
         NFLG      :     in std_logic ;
         ZFLG      :     in std_logic ;
         RESET     :     in std_logic ;
         CLK       :     in std_logic ;
                                              --define the input signal
         STATE     :     out std_logic_vector (3 downto 0);
         LOAD_AC   :     out std_logic ;
         LOAD_IRU  :     out std_logic ;
         LOAD_IRL  :     out std_logic ;
         LOAD_PC   :     out std_logic ;
         INCR_PC   :     out std_logic ;
         FETCH     :     out std_logic ;
         STORE_MEN:      out std_logic               --define the output signal
      );
   end CU;

   architecture behavioral of CU is
      type state_type is (start,prepu,fetchu,prep1,fetch1,class1,class2,class3,class3_2,class4,class5);
          --10 state_type
      signal present_state, next_state : state_type;
   begin

   sync_proc:   -- synchronous process
   process (RESET, CLK,opcode)
   begin
      if RESET = '1' then
         present_state <= start;
      elsif (CLK'event and CLK = '0') then -- falling edge
         present_state <= next_state;
      end if;
   end process;
   comb_proc:   -- combinational process
   process (present_state, next_state,opcode)
   begin
      case present_state is
         when start =>                 --state 1  start
            STATE       <= x"0";
            LOAD_AC     <= '0';
            LOAD_IRU    <= '0';
            LOAD_IRL    <= '0';
```

```
            LOAD_PC     <= '0';
            INCR_PC     <= '0';
            FETCH       <= '0';
            STORE_MEN   <= '0';
            next_state  <= prepu;
         when prepu =>                --state 2  prepare to load IRU
            STATE       <= x"1";
            LOAD_AC     <= '0';
            LOAD_IRU    <= '0';
            LOAD_IRL    <= '0';
            LOAD_PC     <= '0';
            INCR_PC     <= '0';
            FETCH       <= '1';
            STORE_MEN   <= '0';
            next_state  <= fetchu;
         when fetchu =>              --state 3  fetchU
            STATE       <= x"2";
            LOAD_AC     <= '0';
            LOAD_IRU    <= '1';
            LOAD_IRL    <= '0';
            LOAD_PC     <= '0';
            INCR_PC     <= '1';
            FETCH       <= '0';
            STORE_MEN   <= '0';
            case opcode  is
               when x"00" =>
                  next_state <= class1;
               when x"04" =>
                  next_state <= class1;
               when others =>
                  next_state <= prep1;
            end case;
         when prep1 =>               --state 4  prepare to load IRL
            STATE       <= x"3";
            LOAD_AC     <= '0';
            LOAD_IRU    <= '0';
            LOAD_IRL    <= '0';
            LOAD_PC     <= '0';
            INCR_PC     <= '0';
            FETCH       <= '1';
            STORE_MEN   <= '0';
            next_state  <= fetch1;
         when fetch1 =>              --state 5    fetch
            STATE       <= x"4";
            LOAD_AC     <= '0';
            LOAD_IRU    <= '0';
            LOAD_IRL    <= '1';
            LOAD_PC     <= '0';
            INCR_PC     <= '1';
            FETCH       <= '0';
            STORE_MEN   <= '0';
            case opcode is
               when x"01" =>
                  next_state <= class3;
               when x"02" =>
                  next_state <= class2;
               when x"03" =>
                  next_state <= class4;
               when x"05" =>
```

```
                next_state <= class3;
            when x"06" =>
                next_state <= class2;
            when x"07" =>
                next_state <= class3;
            when x"08" =>
                next_state <= class2;
            when x"09" =>
                next_state <= class3;
            when x"0A" =>
                next_state <= class3;
            when x"0B" =>
                next_state <= class3;
            when x"0C" =>
                next_state <= class3;
            when x"0D" =>
                next_state <= class3;
            when x"0E" =>
                next_state <= class2;
            when x"0F" =>
                next_state <= class2;
            when x"10" =>
                next_state <= class5;
            when x"11" =>
                next_state <= class5;
            when x"12" =>
                next_state <= class5;
            when x"13" =>
                next_state <= class5;
            when x"14" =>
                next_state <= class5;
            when others =>
                next_state <= prepu;
        end case;
    when class1 =>              --state 6 :class1
        STATE       <= x"A";
        LOAD_IRU    <= '0';
        LOAD_IRL    <= '0';
        LOAD_PC     <= '0';
        INCR_PC     <= '1';
        FETCH       <= '0';
        STORE_MEN   <= '0';
        if opcode = x"04" then
        LOAD_AC     <= '1';
        next_state  <= prepu;
        else
        LOAD_AC <= '0';
        next_state  <= prepu;
        end if;
    when class2 =>              --state 7 :class2
        STATE       <= x"B";
        LOAD_AC     <= '1';
        LOAD_IRU    <= '0';
        LOAD_IRL    <= '0';
        LOAD_PC     <= '0';
        INCR_PC     <= '0';
        FETCH       <= '0';
        STORE_MEN   <= '0';
        next_state  <= prepu;
```

```
        when class3 =>              --state 8 :class3
           STATE        <= x"C";
           LOAD_IRU     <= '0';
           LOAD_PC      <= '0';
           INCR_PC      <= '0';
           FETCH        <= '0';
           STORE_MEN    <= '1';
           LOAD_IRL     <= '0';
           LOAD_AC  <= '0';
           next_state <= class3_2;
        when class3_2 =>            --state 8 : class3_2
           STATE        <= x"C";
           LOAD_IRU     <= '0';
           LOAD_PC      <= '0';
           INCR_PC      <= '0';
           FETCH        <= '0';
           STORE_MEN    <= '0';
           LOAD_IRL     <= '0';
           LOAD_AC  <= '1';
           next_state <= prepu;
        when class4 =>              --state 9 : class4
           STATE        <= x"D";
           LOAD_AC      <= '1';
           LOAD_IRU     <= '0';
           LOAD_IRL     <= '0';
           LOAD_PC      <= '0';
           INCR_PC      <= '0';
           FETCH        <= '0';
           STORE_MEN    <= '1';
           next_state <= prepu;
        when class5 =>              --state 10 : class5
           STATE        <= x"E";
           LOAD_AC      <= '1';
           LOAD_IRU     <= '0';
           LOAD_IRL     <= '0';
           INCR_PC      <= '0';
           FETCH        <= '0';
           STORE_MEN    <= '1';
           if NFLG = '1' then
              LOAD_PC      <= '1';
              else
              LOAD_PC      <= '0';
           end if;
           next_state  <= prepu;
        when others => null;
     end case;
  end process;
  end behavioral;
```

C. Block diagram

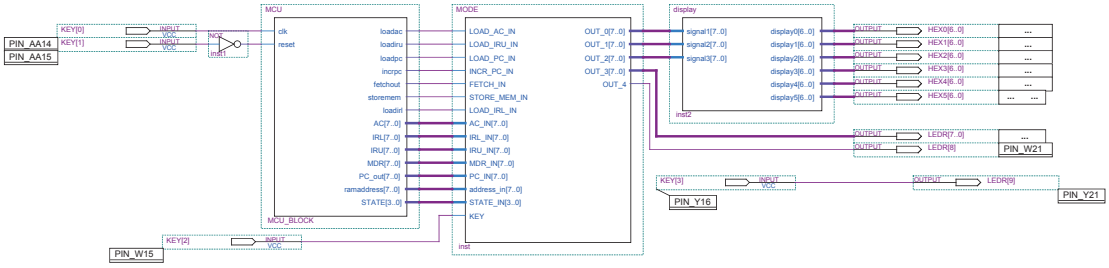

FIG. II.1 mcu_tb Block diagram

D. Simulation results and detailed explanation

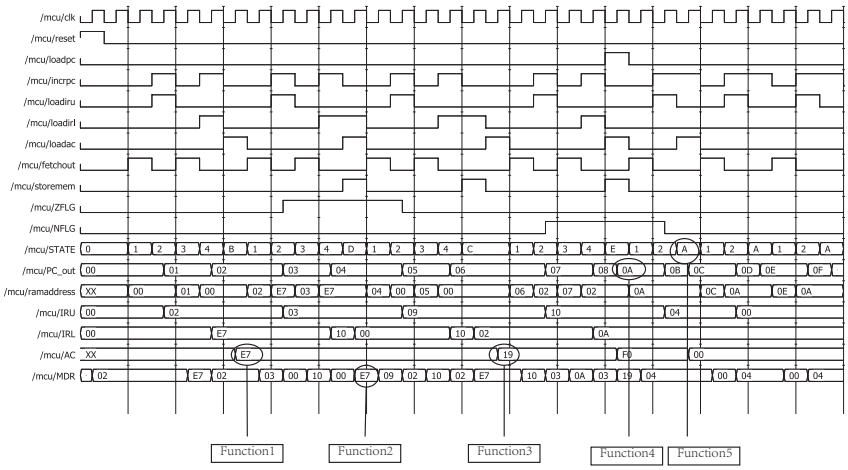

FIG. II.2 Wave

F	code	instruction	Feature result
Function 1	02 E7	AC←—E7	AC=E7
Function 2	03 10	MDR[x"10"]←—AC	MDR[x"10"]=E7
Function 3	09 10	AC←—0-MDR[x"10"]	AC=19
Function 4	10 0A	PC←—0A	PC=0A
Function 5	00 00	nop	nop

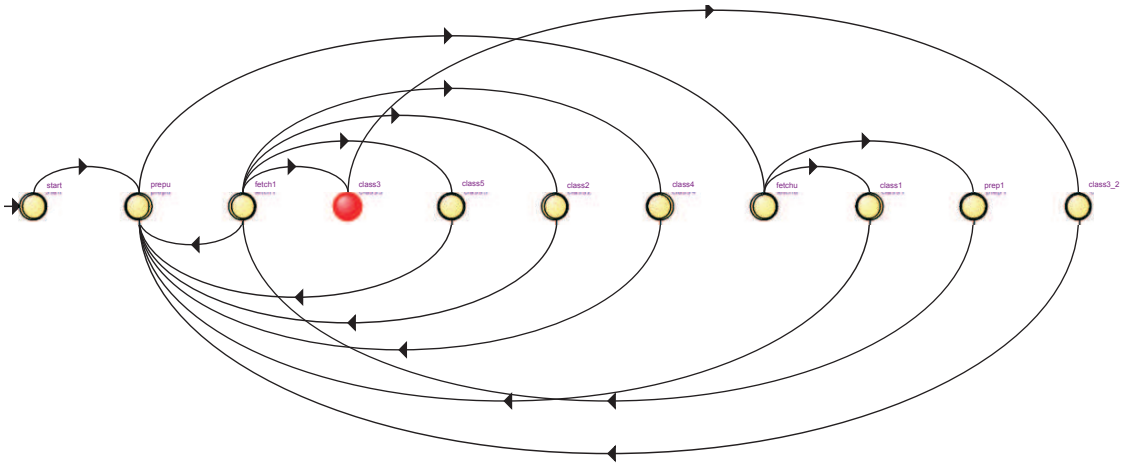

FIG. II.3 State machine viewer

E. Board demonstration result

The board demonstration result is following as the table at section "Simulation strategy" and the FIG: Wave.
In others case the result is the same logic we need.

III. CONCLUSION

The lab is successful.
I realize all function we should achieve. The most difficulty things is the Instructor questions. I refer to manual and ask classmates that I solved it.