Project A:

Complete Processor

By:

Jiaqing Wan(16)

Instructor: Dr. Xi Zhou

Classnumber: ZM1501

January 2, 2018

I. STATE LIST/DIAGRAM

	Source State	Destination State	Condition
1	class1	prepu	
2	class2	prepu	
3	class3	class3_2	
4	class3_2	prepu	
5	class4	prepu	
6	class5	prepu	
7	fetch1	SP_LOAD	(opcode[0]).(!opcode[1]).(opcode[2]).(!opcode[3]).(opcode[4]).(!opcode[5]).(!opcode[6]).(!opcode[7])
8	fetch1	prepu	(!opcode[0]).(!opcode[1]).(!opcode[2]).(!opcode[3]).(!opcode[4]) + (!opcode[0]).(!opcode[1]).(!opcode[2]).(!opcode[3]).(opcode[4]).(!opcode[5]).(!opcode[6]).(opcode[7]) + (!opcode[0]).(!opcode[1]).(!opcode[2]).(!opcode[3]).(opcode[4]).(!opcode[5]).(opcode[6]) + (!opcode[0]).(!opcode[1]).(!opcode[2]).(!opcode[3]).(opcode[4]).(opcode[5]) + (!opcode[0]).(!opcode[1]).(!opcode[2]).(opcode[3]).(!opcode[5]).(!opcode[6]).(opcode[7]) + (!opcode[0]).(!opcode[1]).(!opcode[2]).(opcode[3]).(!opcode[4]).(!opcode[5]).(opcode[6]) + (!opcode[0]).(!opcode[1]).(!opcode[2]).(opcode[3]).(opcode[4]).(!opcode[5]).(!opcode[6]).(opcode[7]) + (!opcode[0]).(!opcode[1]).(opcode[2]).(!opcode[3]).(!opcode[4]).(!opcode[5]).(opcode[6]) + (!opcode[0]).(!opcode[1]).(opcode[2]).(!opcode[3]).(opcode[4]).(!opcode[5]).(!opcode[6]).(opcode[7]) + (!opcode[0]).(!opcode[1]).(opcode[2]).(opcode[3]).(!opcode[4]).(!opcode[5]).(opcode[6]) + (!opcode[0]).(!opcode[1]).(opcode[2]).(opcode[3]).(!opcode[4]).(opcode[5]) + (!opcode[0]).(!opcode[1]).(opcode[2]).(opcode[3]).(opcode[4]) + (!opcode[0]).(opcode[1]).(!opcode[2]).(!opcode[3]).(!opcode[5]).(!opcode[6]).(opcode[7]) + (!opcode[0]).(opcode[1]).(!opcode[2]).(!opcode[3]).(opcode[5]) + (!opcode[0]).(opcode[1]).(!opcode[2]).(opcode[3]).(!opcode[4]).(!opcode[5]).(!opcode[6]).(opcode[7]) + (!opcode[0]).(opcode[1]).(!opcode[2]).(opcode[3]).(!opcode[4]).(opcode[5]) + (!opcode[0]).(opcode[1]).(opcode[2]).(!opcode[3]).(!opcode[4]).(!opcode[5]).(opcode[6]) + (!opcode[0]).(opcode[1]).(opcode[2]).(!opcode[3]).(!opcode[4]).(opcode[5]).(!opcode[6]).(opcode[7]) + (!opcode[0]).(opcode[1]).(opcode[2]).(!opcode[3]).(opcode[4]).(!opcode[5]).(opcode[6]) + (!opcode[0]).(opcode[1]).(opcode[2]).(opcode[3]).(!opcode[4]).(opcode[5]).(!opcode[6]).(!opcode[7]) + (!opcode[0]).(opcode[1]).(opcode[2]).(opcode[3]).(opcode[4]).(!opcode[5]).(!opcode[6]).(!opcode[7]) + (opcode[0]).(!opcode[1]).(!opcode[2]).(!opcode[3]).(!opcode[4]).(!opcode[5]).(!opcode[6]).(!opcode[7])
9	fetch1	JSR_1	(!opcode[0]).(!opcode[1]).(!opcode[2]).(opcode[3]).(opcode[4]).(!opcode[5]).(!opcode[6]).(!opcode[7])
10	fetch1	class2	(!opcode[0]).(!opcode[1]).(!opcode[2]).(opcode[3]).(!opcode[4]).(!opcode[5]).(!opcode[6]).(!opcode[7]) + (!opcode[0]).(opcode[1]).(!opcode[2]).(!opcode[3]).(!opcode[4]).(opcode[5]).(!opcode[6]).(!opcode[7]) + (!opcode[0]).(opcode[1]).(!opcode[2]).(!opcode[4]).(!opcode[5]).(!opcode[6]).(!opcode[7]) + (opcode[0]).(opcode[1]).(opcode[2]).(opcode[3]).(!opcode[4]).(!opcode[5]).(!opcode[6]).(!opcode[7])
11	fetch1	class3	(!opcode[0]).(!opcode[1]).(opcode[2]).(opcode[3]).(!opcode[4]).(!opcode[5]).(!opcode[6]).(!opcode[7]) + (!opcode[0]).(opcode[1]).(!opcode[2]).(opcode[3]).(!opcode[4]).(!opcode[5]).(!opcode[6]).(!opcode[7]) + (opcode[0]).(!opcode[1]).(!opcode[4]).(!opcode[5]).(opcode[6]).(!opcode[7]) + (opcode[0]).(opcode[1]).(!opcode[2]).(opcode[3]).(!opcode[4]).(!opcode[5]).(!opcode[6]).(!opcode[7]) + (opcode[0]).(opcode[1]).(opcode[2]).(!opcode[3]).(!opcode[4]).(!opcode[5]).(!opcode[6]).(!opcode[7])
12	fetch1	class4	(opcode[0]).(opcode[1]).(!opcode[2]).(!opcode[3]).(!opcode[4]).(!opcode[5]).(!opcode[6]).(!opcode[7])
13	fetch1	class5	(!opcode[0]).(!opcode[1]).(!opcode[3]).(opcode[4]).(!opcode[5]).(!opcode[6]).(!opcode[7]) + (!opcode[0]).(opcode[1]).(!opcode[2]).(!opcode[3]).(opcode[4]).(!opcode[5]).(!opcode[6]).(!opcode[7]) + (opcode[0]).(!opcode[2]).(!opcode[3]).(opcode[4]).(!opcode[5]).(!opcode[6]).(!opcode[7])
14	fetchu	RTS_1	(opcode[0]).(!opcode[1]).(!opcode[2]).(opcode[3]).(opcode[4]).(!opcode[5]).(!opcode[6]).(!opcode[7])
15	fetchu	PUSH_1	(!opcode[0]).(opcode[1]).(opcode[2]).(!opcode[3]).(opcode[4]).(!opcode[5]).(!opcode[6]).(!opcode[7])
16	fetchu	prep1	(!opcode[0]).(!opcode[1]).(!opcode[3]).(!opcode[4]).(!opcode[5]).(!opcode[6]).(opcode[7]) + (!opcode[0]).(!opcode[1]).(!opcode[3]).(!opcode[4]).(!opcode[5]).(opcode[6]) + (!opcode[0]).(!opcode[1]).(!opcode[3]).(!opcode[4]).(opcode[5]) + (!opcode[0]).(!opcode[1]).(!opcode[3]).(opcode[4]) + (!opcode[0]).(!opcode[1]).(opcode[3]) + (!opcode[0]).(opcode[1]).(!opcode[2]) + (!opcode[0]).(opcode[1]).(opcode[2]).(!opcode[3]).(!opcode[4]) + (!opcode[0]).(opcode[1]).(opcode[2]).(!opcode[3]).(opcode[4]).(!opcode[5]).(!opcode[6]).(opcode[7]) + (!opcode[0]).(opcode[1]).(opcode[2]).(!opcode[3]).(opcode[4]).(opcode[5]) + (!opcode[0]).(opcode[1]).(!opcode[2]).(opcode[3]).(!opcode[4]).(!opcode[5]).(!opcode[6]).(opcode[7]) + (!opcode[0]).(opcode[1]).(!opcode[2]).(opcode[3]).(!opcode[4]).(opcode[5]).(!opcode[6]).(!opcode[7]) + (!opcode[0]).(opcode[1]).(!opcode[2]).(opcode[3]).(opcode[4]).(!opcode[5]).(!opcode[6]).(!opcode[7]) + (opcode[0]).(!opcode[1]).(!opcode[2]).(!opcode[3]).(!opcode[4]).(!opcode[5]).(!opcode[6]).(!opcode[7])
17	fetchu	POP_1	(opcode[0]).(opcode[1]).(opcode[2]).(!opcode[3]).(opcode[4]).(!opcode[5]).(!opcode[6]).(!opcode[7])
18	fetchu	class1	(!opcode[0]).(!opcode[1]).(!opcode[3]).(!opcode[4]).(!opcode[5]).(!opcode[6]).(!opcode[7])
19	JSR_1	JSR_2	
20	JSR_2	JSR_3	
21	JSR_3	prepu	
22	POP_1	POP_2	
23	POP_2	POP_3	
24	POP_3	prepu	
25	prep1	fetch1	
26	prepu	fetchu	
27	PUSH_1	PUSH_2	
28	PUSH_2	prepu	
29	RTS_1	RTS_2	
30	RTS_2	RTS_3	
31	RTS_3	prepu	
32	SP_LOAD	prepu	
33	start	prepu	

FIG. I.1 state table

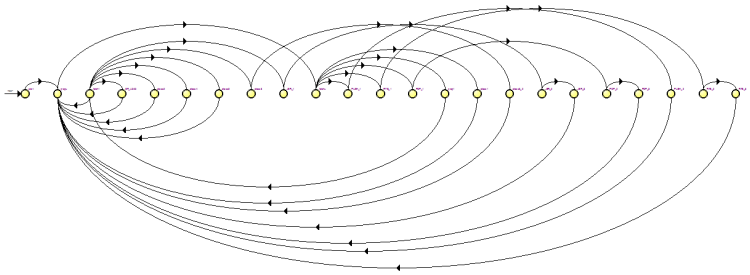

FIG. I.2 state diagram

II. VHDL CODE FOR SP

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
-------------------------------------------------------------------------------
entity sp is
   port (
      RESET :      in std_logic;
      CLK :        in std_logic;
      LOAD_SP :    in std_logic;
      INCR_SP :    in std_logic;
      SUBT_SP :    in std_logic;
      addrvalue : in std_logic_vector (7 downto 0);
      SP :      inout std_logic_vector (7 downto 0)
   );
end sp;
-------------------------------------------------------------------------------
architecture behav of sp is
begin
   process(RESET, CLK, LOAD_SP, INCR_SP, SUBT_SP,addrvalue)
   begin
      if RESET = '1' then
         SP <= "00000000";
      elsif (CLK'event and CLK = '1') and LOAD_SP = '1' then
         SP <= addrvalue;
      elsif (CLK'event and CLK = '1') and INCR_SP = '1' then
         SP <= SP+'1';
      elsif (CLK'event and CLK = '1') and SUBT_SP = '1' then
         SP <= SP-'1';
      else null;
      end if;
   end process;
end behav;
```

III. VHDL CODE FOR CONTROL UNIT

```
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_vector (1 downto 0) ;
         STORE_MEN:      out std_logic ;              --define the output signal
         LOAD_SP   :     out std_logic := '0';
         INCR_SP   :     out std_logic := '0';
         SUBT_SP   :     out std_logic := '0';
         AC0PC1    :     out std_logic := '0'
      );
   end CU;

   architecture behavioral of CU is
      type state_type is (start,prepu,fetchu,prep1,fetch1,class1,class2,class3,class3_2,class4,class5,
          SP_LOAD,PUSH_1,PUSH_2,POP_1,POP_2,POP_3,JSR_1,JSR_2,JSR_3,RTS_1,RTS_2,RTS_3);
      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       <= "00";
            STORE_MEN   <= '0';
            LOAD_SP     <= '0';
            INCR_SP     <= '0';
            SUBT_SP     <= '0';
            AC0PC1      <= '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';
            LOAD_SP     <= '0';
            INCR_SP     <= '0';
            SUBT_SP     <= '0';
            FETCH       <= "01";
            STORE_MEN   <= '0';
            AC0PC1      <= '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';
            LOAD_SP     <= '0';
            INCR_SP     <= '0';
            SUBT_SP     <= '0';
            FETCH       <= "00";
            STORE_MEN   <= '0';
            AC0PC1      <= '0';
            case opcode  is
               when x"00" =>
                  next_state <= class1;
               when x"04" =>
                  next_state <= class1;
               when x"16" =>
                  next_state <= PUSH_1;
               when x"17" =>
                  next_state <= POP_1;
               when x"19" =>
                  next_state <=RTS_1;
               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';
            LOAD_SP     <= '0';
            INCR_SP     <= '0';
            SUBT_SP     <= '0';
            FETCH       <= "01";
            STORE_MEN   <= '0';
            AC0PC1      <= '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';
            LOAD_SP      <= '0';
            INCR_SP      <= '0';
            SUBT_SP      <= '0';
            FETCH        <= "00";
            STORE_MEN    <= '0';
            AC0PC1       <= '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 x"15" =>
                  next_state <= SP_LOAD;
               when x"18" =>
                  next_state <=JSR_1;
               when others =>
                  next_state <= prepu;
            end case;
         when class1 =>              --state 6 :class1
            STATE        <= x"5";
            LOAD_IRU     <= '0';
            LOAD_IRL     <= '0';
            LOAD_PC      <= '0';
            INCR_PC      <= '1';
```

```
            LOAD_SP     <= '0';
            INCR_SP     <= '0';
            SUBT_SP     <= '0';
            FETCH       <= "00";
            STORE_MEN   <= '0';
            AC0PC1      <= '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"6";
            LOAD_AC     <= '1';
            LOAD_IRU    <= '0';
            LOAD_IRL    <= '0';
            LOAD_PC     <= '0';
            INCR_PC     <= '0';
            LOAD_SP     <= '0';
            INCR_SP     <= '0';
            SUBT_SP     <= '0';
            FETCH       <= "00";
            STORE_MEN   <= '0';
            AC0PC1      <= '0';
            next_state  <= prepu;
        when class3 =>              --state 8 :class3
            STATE       <= x"7";
            LOAD_IRU    <= '0';
            LOAD_PC     <= '0';
            INCR_PC     <= '0';
            LOAD_SP     <= '0';
            INCR_SP     <= '0';
            SUBT_SP     <= '0';
            FETCH       <= "00";
            STORE_MEN   <= '0';
            LOAD_IRL    <= '0';
            LOAD_AC  <= '0';
            AC0PC1      <= '0';
            next_state <= class3_2;
        when class3_2 =>              --state 8 :class3
            STATE       <= x"7";
            LOAD_IRU    <= '0';
            LOAD_PC     <= '0';
            INCR_PC     <= '0';
            LOAD_SP     <= '0';
            INCR_SP     <= '0';
            SUBT_SP     <= '0';
            FETCH       <= "00";
            STORE_MEN   <= '0';
            LOAD_IRL    <= '0';
            LOAD_AC  <= '1';
            AC0PC1      <= '0';
            next_state <= prepu;
        when class4 =>              --state 9 : class4
            STATE       <= x"8";
            LOAD_AC     <= '0';
            LOAD_IRU    <= '0';
```

```
            LOAD_IRL    <= '0';
            LOAD_PC     <= '0';
            INCR_PC     <= '0';
            LOAD_SP     <= '0';
            INCR_SP     <= '0';
            SUBT_SP     <= '0';
            FETCH       <= "00";
            STORE_MEN   <= '1';
            AC0PC1      <= '0';
            next_state <= prepu;
        when class5 =>              --state 10 : class5
            STATE       <= x"9";
            LOAD_AC     <= '0';
            LOAD_IRU    <= '0';
            LOAD_IRL    <= '0';
            INCR_PC     <= '0';
            LOAD_SP     <= '0';
            INCR_SP     <= '0';
            SUBT_SP     <= '0';
            FETCH       <= "00";
            STORE_MEN   <= '0';
            AC0PC1      <= '0';
            if NFLG = '1' then
               LOAD_PC     <= '1';
               else
               LOAD_PC     <= '0';
            end if;
            next_state  <= prepu;
        when SP_LOAD =>
            STATE       <= x"A";
            LOAD_AC     <= '0';
            LOAD_IRU    <= '0';
            LOAD_IRL    <= '0';
            LOAD_PC     <= '0';
            INCR_PC     <= '0';
            LOAD_SP     <= '1';
            INCR_SP     <= '0';
            SUBT_SP     <= '0';
            FETCH       <= "00";
            STORE_MEN   <= '0';
            next_state <= prepu;
        when PUSH_1 =>           --SP = SP-1;
            STATE       <= x"B";
            LOAD_AC     <= '0';
            LOAD_IRU    <= '0';
            LOAD_IRL    <= '0';
            LOAD_PC     <= '0';
            INCR_PC     <= '0';
            LOAD_SP     <= '0';
            INCR_SP     <= '0';
            SUBT_SP     <= '1';
            FETCH       <= "10";
            STORE_MEN   <= '0';
            AC0PC1      <= '0';
            next_state <= PUSH_2;
        when PUSH_2 =>          --M[SP]<=AC
            STATE       <= x"B";
            LOAD_AC     <= '0';
            LOAD_IRU    <= '0';
```

```
            LOAD_IRL    <= '0';
            LOAD_PC     <= '0';
            INCR_PC     <= '0';
            LOAD_SP     <= '0';
            INCR_SP     <= '0';
            SUBT_SP     <= '0';
            FETCH       <= "10";
            STORE_MEN   <= '1';
            AC0PC1      <= '0';
            next_state <= prepu;
        when POP_1 =>           --AC<=M[SP]
            STATE       <= x"C";
            LOAD_AC     <= '0';
            LOAD_IRU    <= '0';
            LOAD_IRL    <= '0';
            LOAD_PC     <= '0';
            INCR_PC     <= '0';
            LOAD_SP     <= '0';
            INCR_SP     <= '0';
            SUBT_SP     <= '0';
            FETCH       <= "10";
            STORE_MEN   <= '0';
            next_state <= POP_2;
        when POP_2 =>           --AC<=M[SP]
            STATE       <= x"C";
            LOAD_AC     <= '1';
            LOAD_IRU    <= '0';
            LOAD_IRL    <= '0';
            LOAD_PC     <= '0';
            INCR_PC     <= '0';
            LOAD_SP     <= '0';
            INCR_SP     <= '0';
            SUBT_SP     <= '0';
            FETCH       <= "10";
            STORE_MEN   <= '0';
            next_state <= POP_3;
        when POP_3 =>           --SP <= SP+1
            STATE       <= x"C";
            LOAD_AC     <= '0';
            LOAD_IRU    <= '0';
            LOAD_IRL    <= '0';
            LOAD_PC     <= '0';
            INCR_PC     <= '0';
            LOAD_SP     <= '0';
            INCR_SP     <= '1';
            SUBT_SP     <= '0';
            FETCH       <= "00";
            STORE_MEN   <= '0';
            AC0PC1      <= '0';
            next_state <= prepu;
        when JSR_1 =>           --SP<=SP-1
            STATE       <= x"D";
            LOAD_AC     <= '0';
            LOAD_IRU    <= '0';
            LOAD_IRL    <= '0';
            LOAD_PC     <= '0';
            INCR_PC     <= '0';
            LOAD_SP     <= '0';
            INCR_SP     <= '0';
```

```
            SUBT_SP      <= '1';
            FETCH        <= "00";
            STORE_MEN    <= '0';
            AC0PC1       <= '0';
            next_state <= JSR_2;
        when JSR_2 =>              --M[SP]<=PC
            STATE        <= x"D";
            LOAD_AC      <= '0';
            LOAD_IRU     <= '0';
            LOAD_IRL     <= '0';
            LOAD_PC      <= '0';
            INCR_PC      <= '0';
            LOAD_SP      <= '0';
            INCR_SP      <= '0';
            SUBT_SP      <= '0';
            FETCH        <= "10";
            STORE_MEN    <= '1';
            AC0PC1       <= '1';
            next_state <= JSR_3;
        when JSR_3 =>              --PC<=address
            STATE        <= x"D";
            LOAD_AC      <= '0';
            LOAD_IRU     <= '0';
            LOAD_IRL     <= '0';
            LOAD_PC      <= '1';
            INCR_PC      <= '0';
            LOAD_SP      <= '0';
            INCR_SP      <= '0';
            SUBT_SP      <= '0';
            FETCH        <= "00";
            STORE_MEN    <= '0';
            AC0PC1       <= '0';
            next_state <= prepu;
        when RTS_1 =>              --PC<=M[SP]
            STATE        <= x"E";
            LOAD_AC      <= '0';
            LOAD_IRU     <= '0';
            LOAD_IRL     <= '1';
            LOAD_PC      <= '0';
            INCR_PC      <= '0';
            LOAD_SP      <= '0';
            INCR_SP      <= '0';
            SUBT_SP      <= '0';
            FETCH        <= "10";
            STORE_MEN    <= '0';
            AC0PC1       <= '0';
            next_state <= RTS_2;
        when RTS_2 =>              --PC<=M[SP]
            STATE        <= x"E";
            LOAD_AC      <= '0';
            LOAD_IRU     <= '0';
            LOAD_IRL     <= '1';
            LOAD_PC      <= '1';
            INCR_PC      <= '0';
            LOAD_SP      <= '0';
            INCR_SP      <= '0';
            SUBT_SP      <= '0';
            FETCH        <= "00";
            STORE_MEN    <= '0';
```

```
            AC0PC1      <= '0';
            next_state <= RTS_3;
        when RTS_3 =>           --SP<=SP+1
            STATE       <= x"E";
            LOAD_AC     <= '0';
            LOAD_IRU    <= '0';
            LOAD_IRL    <= '0';
            LOAD_PC     <= '0';
            INCR_PC     <= '0';
            LOAD_SP     <= '0';
            INCR_SP     <= '1';
            SUBT_SP     <= '0';
            FETCH       <= "00";
            STORE_MEN   <= '0';
            AC0PC1      <= '0';
            next_state <= prepu;
        when others => null;
    end case;
end process;
end behavioral;
```

IV. SIMULATION RESULTS FOR ACTIVITY

Test for N is 3.

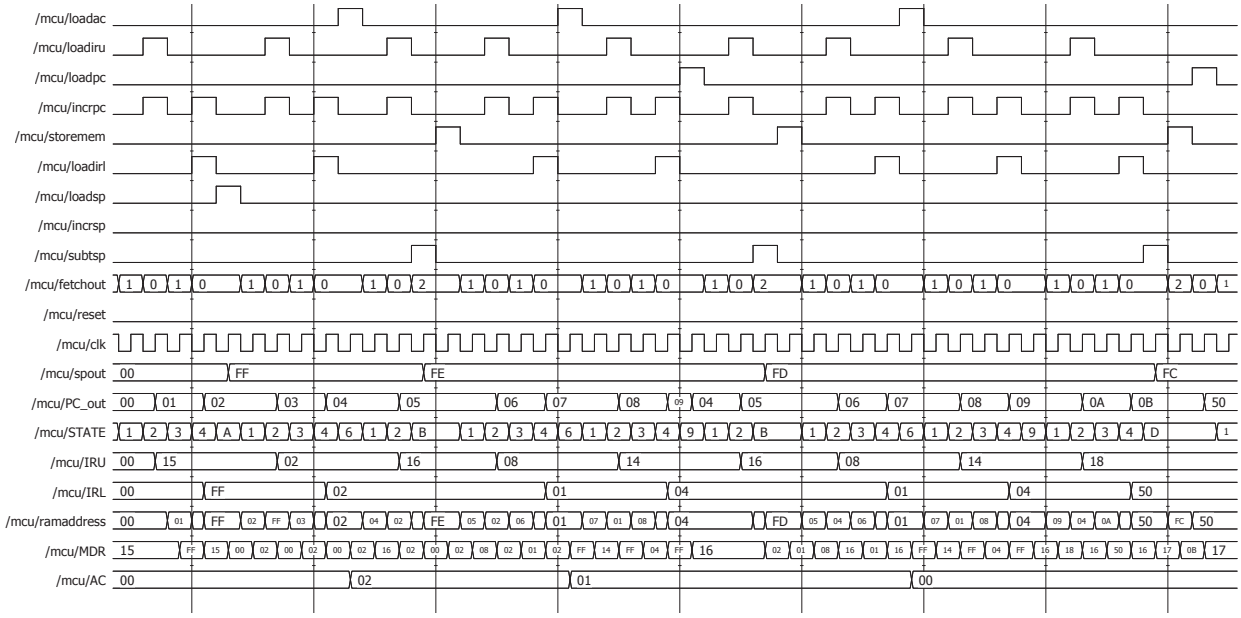

FIG. IV.1 Part A

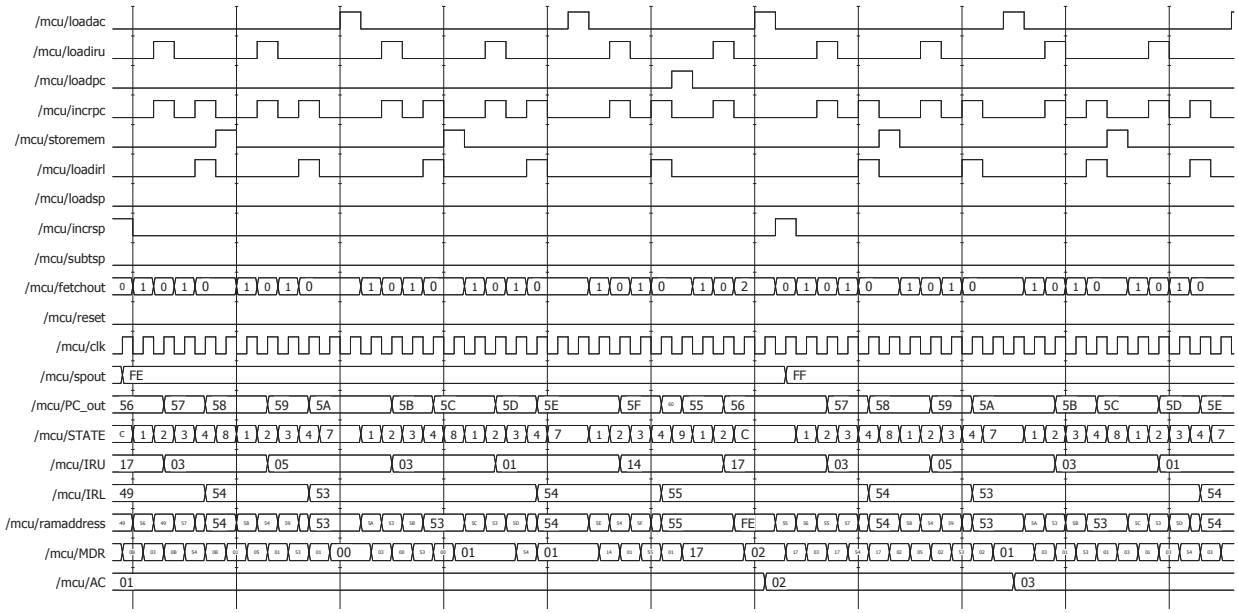

FIG. IV.2 Part B

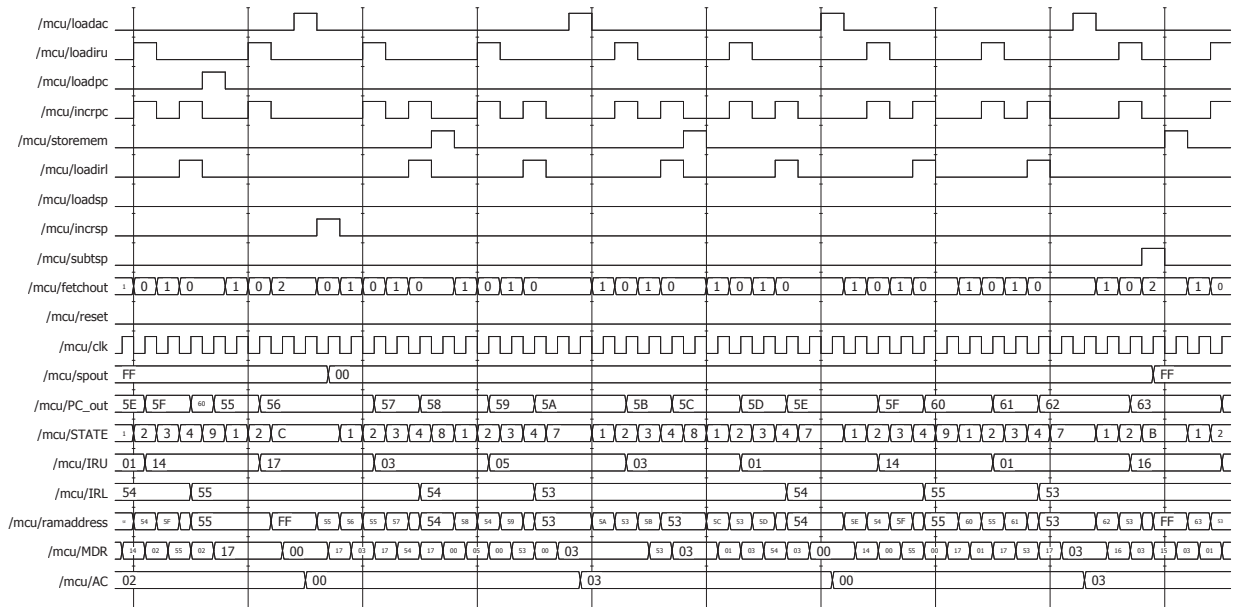

FIG. IV.3 Part C

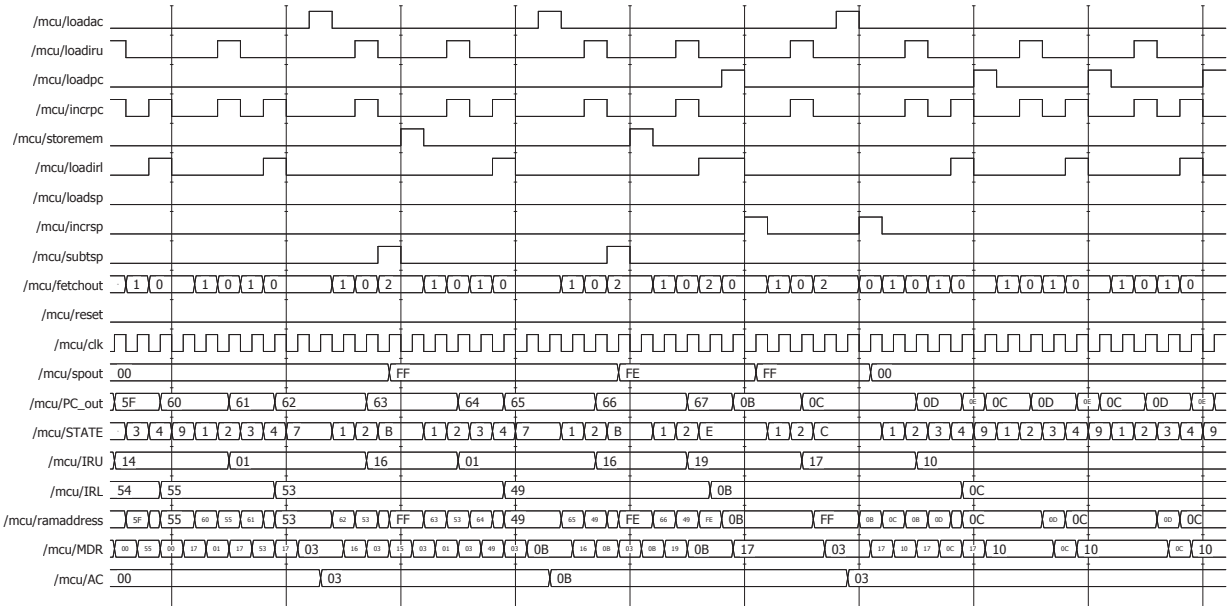

FIG. IV.4 Part D

V. ASSEMBLY LANGUAGE CODE AND MIF FILE

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
000	15	FF	02	02	16	08	01	14	.....
008	04	18	50	17	10	0C	00	00	..P.....
010	00	00	00	00	00	00	00	00	.....
018	00	00	00	00	00	00	00	00	.....
020	00	00	00	00	00	00	00	00	.....
028	00	00	00	00	00	00	00	00	.....
030	00	00	00	00	00	00	00	00	.....
038	00	00	00	00	00	00	00	00	.....
040	00	00	00	00	00	00	00	00	.....
048	00	00	00	00	00	00	00	00	.....
050	17	03	49	00	00	17	03	54	..I....T
058	05	53	03	53	01	54	14	55	.S.S.T.U
060	01	53	16	01	49	16	19	00	.S..I...
068	00	00	00	00	00	00	00	00	.....

FIG. V.1 mif file

Addr	+1	+2	Assembly language code		Explain
0	15	FF	LOADSP	FF	load sp to FF
2	02	02	LOADI	02	load N=2 to AC
4	16		PUSH		push N=2 to stack
5	08	01	SUBTI	01	AC=N=N-1
7	14	04	JNZER	04	if n!=0 ,then jump to address 4
9	18	50	JSR	50	jump to address 50
0B	17		POP		pop the result
0C	10	0C	JUMP	0C	stop instruction in 0C
49					
50	17		POP		POP the subroutines address
51	13	49	STORE	49	store the address in M[49]
53	00	00	NOP		for store the result in M[53]
					for store the N value in M[54]
55	17		POP		Push N value
56	03	54	STORE	54	store the N value in M[54]
58	05	53	ADD	53	add the N to M[53]
60	03	53	STORE	53	store the result in M[53]
62	01	54	LOAD	54	load N value in AC
64	14	55	JNZER	55	if N!=0,then jump to address 55
66	01	53	LOAD	53	load result in AC
68	16		PUSH		push result in stack
69	01	49	LOAD	49	load the subroutines address from M[49]
71	16		PUSH		push subroutines address into stack
72	19		RST		return to the main program