Sriharsha Singam

Lab #8 Report

ECE 2031 L08

30 October 2018

```
          ORG   0
Start:  CALL   CALC
        JUMP   Start
       ORG &H010
CALC:   LOAD   A
        AND    B
        XOR    C
        STORE  D
        RETURN
   ORG     &H030
A:  DW    &H00FF
B:  DW    &HA5A5
C:  DW    &H3300
D:  DW    &H0000
```

**Figure 1.** This is the LAB8PRELAB.asm file that contains the code for a program: D = (A AND B) XOR C and also makes use of the functions CALL and RETURN to make subroutines.

```
-- Altera Memory Initialization File (MIF)

DEPTH = 1024;
WIDTH = 16;

ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;

CONTENT
  BEGIN
    [000..3FF] : 0000;  -- Default to NOP

            000 : 4010;  -- Start:  CALL CALC
            001 : 1400;  --         JUMP      Start
            010 : 0430;  -- CALC:   LOAD      A
            011 : 2431;  --         AND       B
            012 : 2C32;  --         XOR     C
            013 : 0833;  --         STORE     D
            014 : 4400;  --         RETURN
            030 : 00FF;  -- A:      DW      &H00FF
            031 : A5A5;  -- B:      DW      &HA5A5
            032 : 3300;  -- C:      DW      &H3300
            033 : 0000;  -- D:      DW        &H0000
  END;
```

**Figure 2.** This is the LAB8PRELAB.mif file that contains the code for a program: D = (A AND B) XOR C and also makes use of the functions CALL and RETURN to make subroutines.

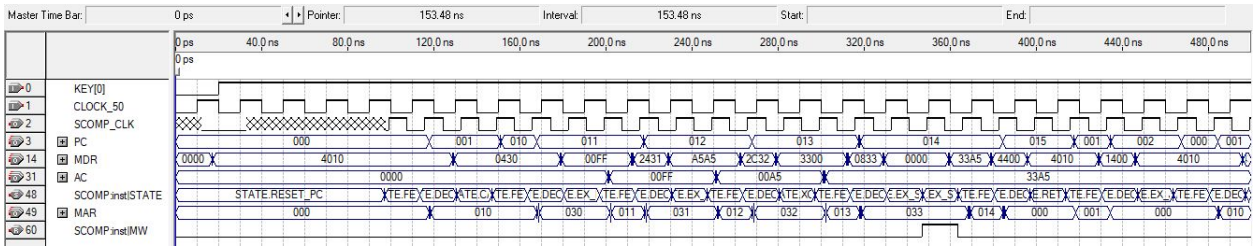



**Figure 3.** This is the timing simulation waveform for the program: LAB8PRELAB.mif file (which is describes a program: D = (A AND B) XOR C and also makes use of the functions CALL and RETURN to make subroutines).

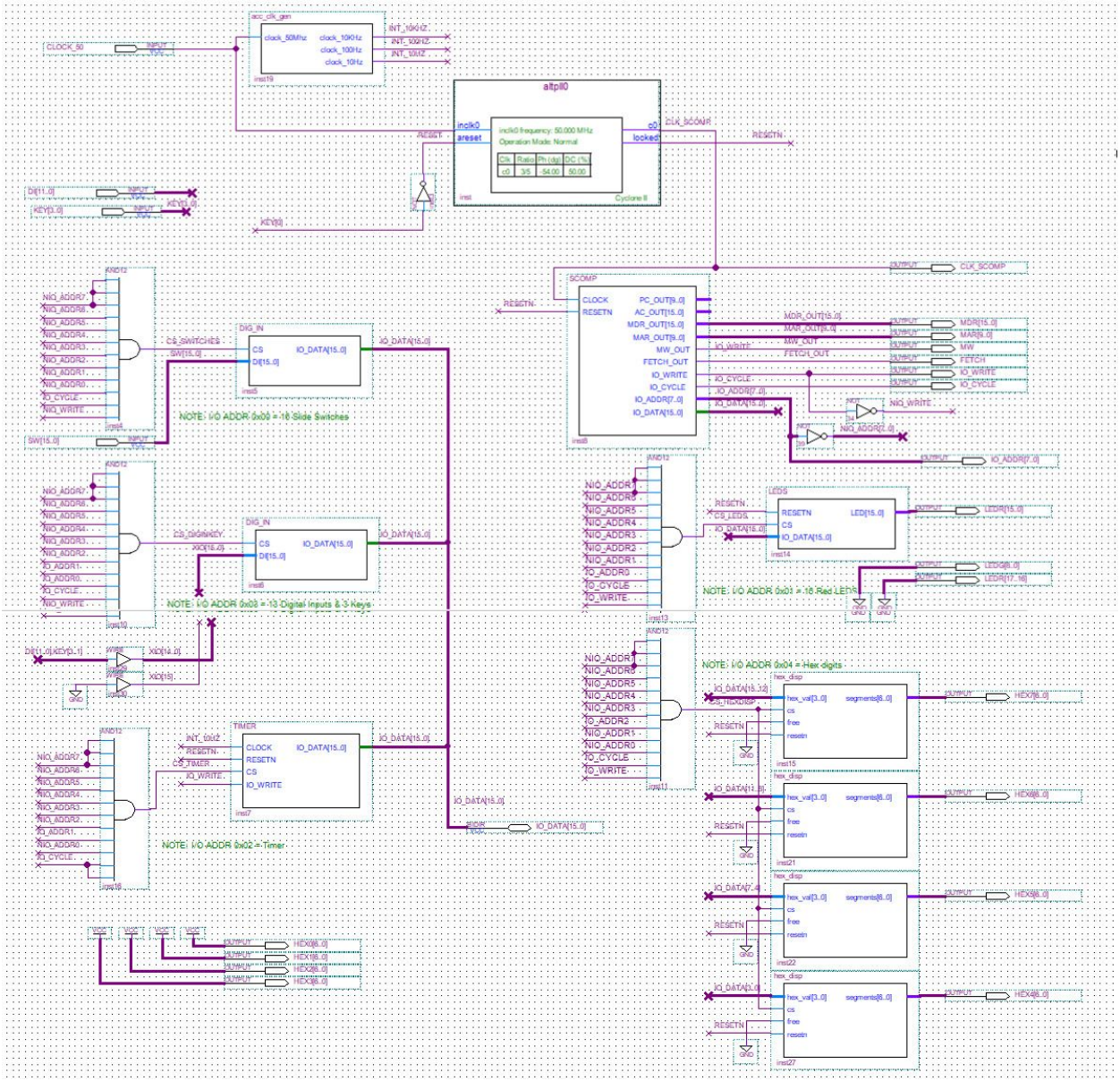



**Figure 4.** This is the schematic for the simple computer. It includes a clock signal and a schematic for the VHDL program that makes the simple computer's logic and all the input and output devices such as the switches, LEDs, and the timer.

```
SWITCHES: EQU  &H00
LEDS:          EQU &H01
TIMER:         EQU &H02
SEVENSEG: EQU &H04


Test:       IN  SWITCHES
            STORE INDATA

SHIFTNOW: LOAD INDATA
            OUT   LEDS
            OUT SEVENSEG
            SHIFT 1
            STORE INDATA
            OUT TIMER

TIME:       IN TIMER
            ADDI -20
            JPOS SHIFTNOW
            JUMP TIME

INDATA:   DW &H0000
```

**Figure 5.** This is the LAB8STEP6.asm file that contains the code for a program that gets an input number from the switches and then outputs the number using the LEDs and the Seven Segment Display and then every 2 seconds it does a logical shift on the number to left by 1.

```
-- Altera Memory Initialization File (MIF)

DEPTH = 1024;
WIDTH = 16;

ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;

CONTENT
  BEGIN
    [000..3FF] : 0000;  -- Default to NOP

           000 : 4800;  -- Test:        IN    SWITCHES
           001 : 080C;  --              STORE INDATA
           002 : 040C;  -- SHIFTNOW: LOAD INDATA
           003 : 4C01;  --              OUT   LEDS
           004 : 4C04;  --              OUT SEVENSEG
           005 : 3001;  --              SHIFT 1
           006 : 080C;  --              STORE INDATA
           007 : 4C02;  --              OUT TIMER
           008 : 4802;  -- TIME:      IN TIMER
           009 : 37EC;  --            ADDI -20
           00A : 1C02;  --            JPOS SHIFTNOW
           00B : 1408;  --            JUMP TIME
           00C : 0000;  -- INDATA:    DW &H0000
  END;
```

**Figure 6.** This is the LAB8STEP6.mif file that contains the code for a program that gets an input number from the switches and then outputs the number using the LEDs and the Seven Segment Display and then every 2 seconds it does a logical shift on the number to left by 1.

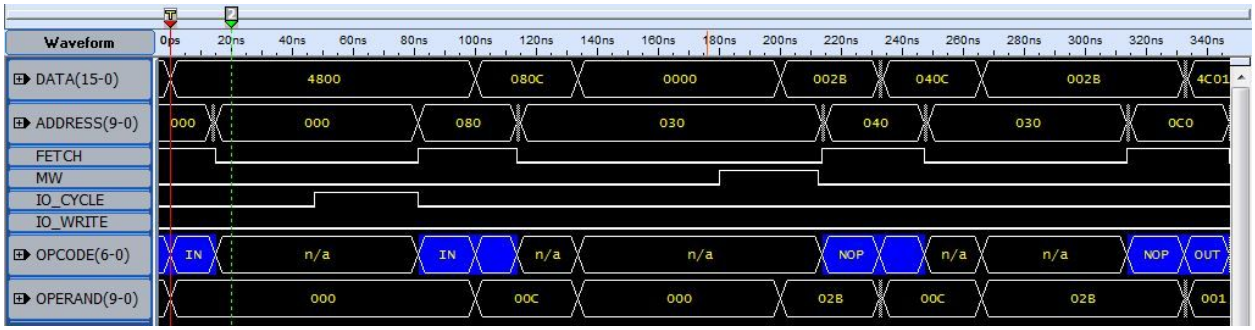
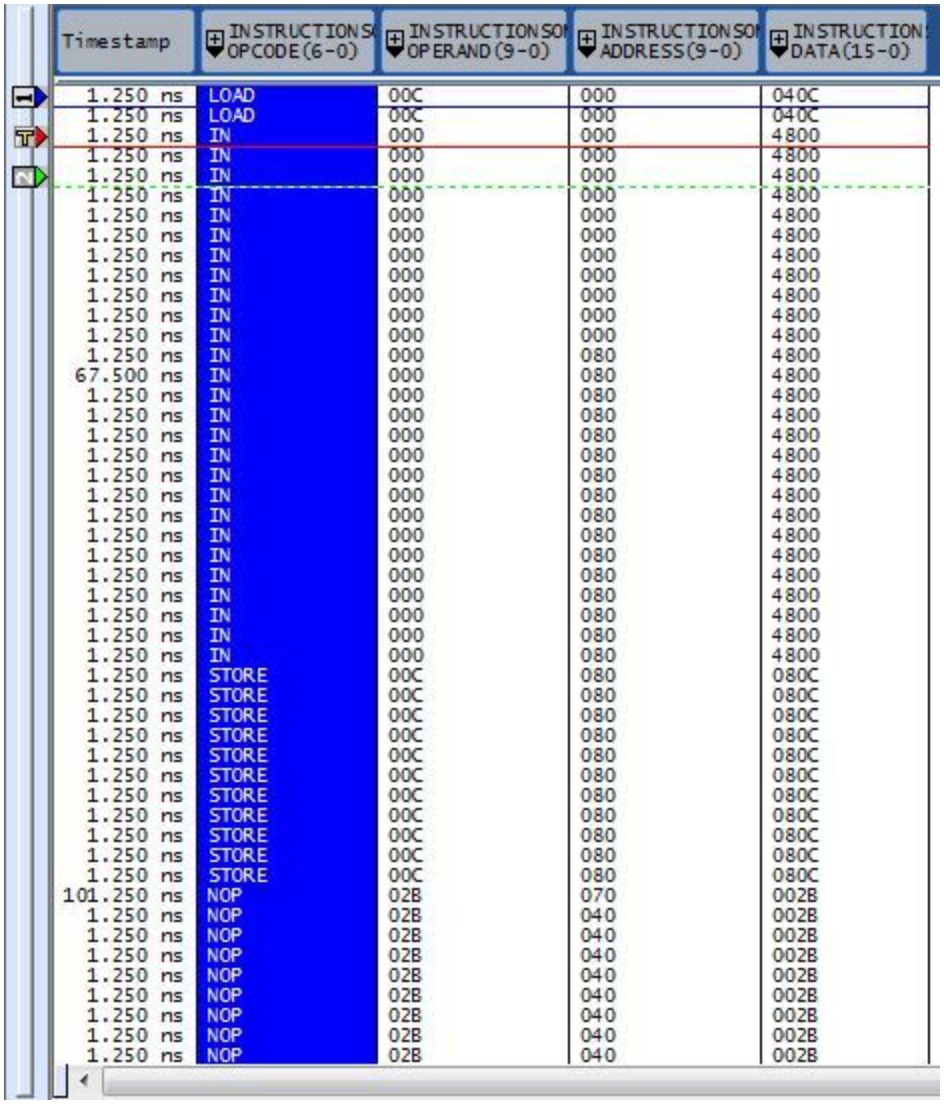



**Figure 7.** This is the Logic Analyzer signal recording graph along with the change in states for the Simple Computer that is running a program that gets an input number from the switches and then outputs the number using the LEDs and the Seven Segment Display and then every 2 seconds it does a logical shift on the number to left by 1.



| Timestamp | INSTRUCTION SI OPCODE(6-0) | INSTRUCTION SO OPERAND(9-0) | INSTRUCTION SO ADDRESS(9-0) | INSTRUCTION DATA(15-0) |
|---|---|---|---|---|
| 1.250 ns | LOAD | 00C | 000 | 040C |
| 1.250 ns | LOAD | 00C | 000 | 040C |
| 1.250 ns | IN | 000 | 000 | 4800 |
| 1.250 ns | IN | 000 | 000 | 4800 |
| 1.250 ns | IN | 000 | 000 | 4800 |
| 1.250 ns | IN | 000 | 000 | 4800 |
| 1.250 ns | IN | 000 | 000 | 4800 |
| 1.250 ns | IN | 000 | 000 | 4800 |
| 1.250 ns | IN | 000 | 000 | 4800 |
| 1.250 ns | IN | 000 | 000 | 4800 |
| 1.250 ns | IN | 000 | 000 | 4800 |
| 1.250 ns | IN | 000 | 000 | 4800 |
| 1.250 ns | IN | 000 | 080 | 4800 |
| 67.500 ns | IN | 000 | 080 | 4800 |
| 1.250 ns | IN | 000 | 080 | 4800 |
| 1.250 ns | IN | 000 | 080 | 4800 |
| 1.250 ns | IN | 000 | 080 | 4800 |
| 1.250 ns | IN | 000 | 080 | 4800 |
| 1.250 ns | IN | 000 | 080 | 4800 |
| 1.250 ns | IN | 000 | 080 | 4800 |
| 1.250 ns | IN | 000 | 080 | 4800 |
| 1.250 ns | IN | 000 | 080 | 4800 |
| 1.250 ns | IN | 000 | 080 | 4800 |
| 1.250 ns | IN | 000 | 080 | 4800 |
| 1.250 ns | IN | 000 | 080 | 4800 |
| 1.250 ns | IN | 000 | 080 | 4800 |
| 1.250 ns | IN | 000 | 080 | 4800 |
| 1.250 ns | STORE | 00C | 080 | 080C |
| 1.250 ns | STORE | 00C | 080 | 080C |
| 1.250 ns | STORE | 00C | 080 | 080C |
| 1.250 ns | STORE | 00C | 080 | 080C |
| 1.250 ns | STORE | 00C | 080 | 080C |
| 1.250 ns | STORE | 00C | 080 | 080C |
| 1.250 ns | STORE | 00C | 080 | 080C |
| 1.250 ns | STORE | 00C | 080 | 080C |
| 1.250 ns | STORE | 00C | 080 | 080C |
| 1.250 ns | STORE | 00C | 080 | 080C |
| 1.250 ns | STORE | 00C | 080 | 080C |
| 101.250 ns | NOP | 02B | 070 | 002B |
| 1.250 ns | NOP | 02B | 040 | 002B |
| 1.250 ns | NOP | 02B | 040 | 002B |
| 1.250 ns | NOP | 02B | 040 | 002B |
| 1.250 ns | NOP | 02B | 040 | 002B |
| 1.250 ns | NOP | 02B | 040 | 002B |
| 1.250 ns | NOP | 02B | 040 | 002B |
| 1.250 ns | NOP | 02B | 040 | 002B |
| 1.250 ns | NOP | 02B | 040 | 002B |

**Figure 8.** This is the Logic Analyzer signal recording table, that filters through the garbage to get only the states and the OPERAND, ADDRESS, and DATA values at the moment in time, for the Simple Computer that is running a program that gets an input number from the switches and then outputs the number using the LEDs and the Seven Segment Display and then every 2 seconds it does a logical shift on the number to left by 1.

# APPENDIX A
# VHDL CODE OF THE SIMPLE COMPUTER THE RUNS FUNCTIONS INCLUDING CALL, RETURN, IN, OUT, AND SHIFT.

```
-- SCOMP.vhd
-- Simple Computer Implementation with CALL, RETURN, IN, OUT,
and SHIFT functions
-- Sriharsha Singam
-- ECE 2031 L08
-- October 30, 2018

LIBRARY IEEE;
LIBRARY ALTERA_MF;
LIBRARY LPM;

USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE ALTERA_MF.ALTERA_MF_COMPONENTS.ALL;
USE LPM.LPM_COMPONENTS.ALL;

ENTITY SCOMP IS
  PORT(
    CLOCK    : IN    STD_LOGIC;
    RESETN   : IN    STD_LOGIC;
    PC_OUT   : OUT   STD_LOGIC_VECTOR( 9 DOWNTO 0);
    AC_OUT   : OUT   STD_LOGIC_VECTOR(15 DOWNTO 0);
    MDR_OUT  : OUT   STD_LOGIC_VECTOR(15 DOWNTO 0);
    MAR_OUT  : OUT   STD_LOGIC_VECTOR( 9 DOWNTO 0);
    MW_OUT      : OUT   STD_LOGIC;
    FETCH_OUT: OUT   STD_LOGIC;
    IO_WRITE : OUT   STD_LOGIC;
    IO_CYCLE : OUT   STD_LOGIC;
    IO_ADDR  : OUT   STD_LOGIC_VECTOR( 7 DOWNTO 0);
    IO_DATA  : INOUT STD_LOGIC_VECTOR(15 DOWNTO 0)
  );
END SCOMP;


ARCHITECTURE a OF SCOMP IS
  TYPE STATE_TYPE IS (
    RESET_PC,
    FETCH,
    DECODE,
    EX_LOAD,
    EX_STORE,
    EX_STORE2,
    EX_ADD,
    EX_JUMP,
```

```
    EX_AND,
    EX_SUB,
    EX_JNEG,
    EX_JPOS,
    EX_JZERO,
    EX_OR,
    EX_XOR,
    EX_ADDI,
    EX_SHIFT,
    EX_CALL,
    EX_RETURN,
    EX_IN,
    EX_OUT,
    EX_OUT2
  );

  SIGNAL STATE    : STATE_TYPE;
  SIGNAL AC       : STD_LOGIC_VECTOR(15 DOWNTO 0);
  SIGNAL IO_IN   : STD_LOGIC_VECTOR(15 DOWNTO 0);
  SIGNAL IR       : STD_LOGIC_VECTOR(15 DOWNTO 0);
  SIGNAL MDR      : STD_LOGIC_VECTOR(15 DOWNTO 0);
  SIGNAL PC       : STD_LOGIC_VECTOR(9 DOWNTO 0);
  SIGNAL MEM_ADDR : STD_LOGIC_VECTOR(9 DOWNTO 0);
  SIGNAL MW       : STD_LOGIC;
  SIGNAL AC_SHIFTED: STD_LOGIC_VECTOR(15 DOWNTO 0);
  SIGNAL PC_STACK: STD_LOGIC_VECTOR(9 DOWNTO 0);
  SIGNAL IO_WRITE_INT : STD_LOGIC;


  BEGIN
    -- Use altsyncram component for unified program and data
memory
    MEMORY : altsyncram
    GENERIC MAP (
    intended_device_family => "Cyclone",
    width_a          => 16,
    widthad_a        => 10,
    numwords_a       => 1024,
    operation_mode   => "SINGLE_PORT",
    outdata_reg_a    => "UNREGISTERED",
    indata_aclr_a    => "NONE",
    wrcontrol_aclr_a => "NONE",
    address_aclr_a   => "NONE",
    outdata_aclr_a   => "NONE",
    init_file        => "LABSTEP6LAB8.mif",
```

```
lpm_hint         => "ENABLE_RUNTIME_MOD=NO",
lpm_type         => "altsyncram"
)
PORT MAP (
wren_a    => MW,
clock0    => NOT(CLOCK),
address_a => MEM_ADDR,
data_a    => AC,
q_a       => MDR
);

-- Use LPN_CLSHIFT function to shift AC
SHIFTER: LPM_CLSHIFT
GENERIC MAP(
    lpm_width    => 16,
    lpm_widthdist => 4,
    lpm_shifttype => "LOGICAL"
)
PORT MAP(
    data    => AC,
    distance  => IR(3 DOWNTO 0),
    direction => IR(4),
    result    => AC_SHIFTED
);

 IO_BUS:   LPM_BUSTRI
 GENERIC MAP(
      lpm_width => 16
 )
 PORT MAP(
      data       => AC,
      enabledt => IO_WRITE_INT,
      tridata    => IO_DATA
 );
PC_OUT   <= PC;
MW_OUT      <= MW;
AC_OUT   <= AC;
MDR_OUT  <= MDR;
MAR_OUT  <= MEM_ADDR;
IO_ADDR  <= IR(7 DOWNTO 0);

WITH STATE SELECT
  MEM_ADDR <= PC WHEN FETCH,
              IR(9 DOWNTO 0) WHEN OTHERS;
```

```
    WITH STATE SELECT
          IO_WRITE  <=  '1' WHEN EX_OUT2,
                            '0' WHEN OTHERS;

     WITH STATE SELECT
          IO_CYCLE  <=  '1' WHEN EX_IN,
                            '1' WHEN EX_OUT2,
                            '0' WHEN OTHERS;

     WITH STATE SELECT
          FETCH_OUT <= '1' WHEN FETCH,
                           '0' WHEN OTHERS;

    PROCESS (CLOCK, RESETN)
      BEGIN
        IF (RESETN = '0') THEN          -- Active low,
asynchronous reset
          STATE <= RESET_PC;
        ELSIF (RISING_EDGE(CLOCK)) THEN
          CASE STATE IS
            WHEN RESET_PC =>
              MW        <= '0';          -- Clear memory write
flag
              PC        <= "0000000000"; -- Reset PC to the
beginning of memory, address 0x000
              AC        <= x"0000";      -- Clear AC register
              STATE     <= FETCH;

            WHEN FETCH =>
              MW    <= '0';             -- Clear memory write
flag
              IR    <= MDR;             -- Latch instruction
into the IR
              PC    <= PC + 1;          -- Increment PC to next
instruction address
              STATE <= DECODE;

            WHEN DECODE =>
              CASE IR(15 downto 10) IS
                WHEN "000000" =>       -- No Operation (NOP)
                  STATE <= FETCH;
                WHEN "000001" =>       -- LOAD
                  STATE <= EX_LOAD;
                WHEN "000010" =>       -- STORE
                  STATE <= EX_STORE;
```

```
                WHEN "000011" =>       -- ADD
                  STATE <= EX_ADD;
                WHEN "000101" =>       -- JUMP
                  STATE <= EX_JUMP;
                WHEN "001001" =>       -- AND
                  STATE <= EX_AND;
                    WHEN "000100" =>
                      STATE <= EX_SUB;
                    WHEN "000110" =>
                      STATE <= EX_JNEG;
                    WHEN "000111" =>
                      STATE <= EX_JPOS;
                    WHEN "001000" =>
                      STATE <= EX_JZERO;
                    WHEN "001010" =>
                      STATE <= EX_OR;
                    WHEN "001011" =>
                      STATE <= EX_XOR;
                    WHEN "001101" =>
                      STATE <= EX_ADDI;
                    WHEN "001100" =>
                      STATE <= EX_SHIFT;
                    WHEN "010000" =>
                      STATE <= EX_CALL;
                    WHEN "010001" =>
                      STATE <= EX_RETURN;
                    WHEN "010010" =>
                      STATE <= EX_IN;
                    WHEN "010011" =>
                      STATE <= EX_OUT;
                      IO_WRITE_INT <= '1';
                WHEN OTHERS =>
                  STATE <= FETCH;      -- Invalid opcodes
default to NOP
              END CASE;

            WHEN EX_LOAD =>
              AC    <= MDR;            -- Latch data from MDR
(memory contents) to AC
              STATE <= FETCH;

            WHEN EX_STORE =>
              MW    <= '1';            -- Raise MW to write AC
to MEM
              STATE <= EX_STORE2;
```

```
            WHEN EX_STORE2 =>
              MW    <= '0';             -- Drop MW to end write
cycle
              STATE <= FETCH;

            WHEN EX_ADD =>
              AC    <= AC + MDR;
              STATE <= FETCH;

            WHEN EX_JUMP =>
              PC    <= IR(9 DOWNTO 0);
              STATE <= FETCH;

            WHEN EX_AND =>
              AC    <= AC AND MDR;
              STATE <= FETCH;

            WHEN EX_SUB =>
                 AC      <= AC - MDR;
                 STATE <= FETCH;

               WHEN EX_JNEG =>
                    IF AC(15) = '1' THEN
                         PC   <= IR(9 DOWNTO 0);
                    END IF;
                    STATE <= FETCH;

               WHEN EX_JZERO =>
                    IF AC = "0000000000000000" THEN
                         PC   <= IR(9 DOWNTO 0);
                    END IF;
                    STATE <= FETCH;

               WHEN EX_JPOS =>
                    IF AC /= "0000000000000000" THEN
                         IF AC(15) = '0' THEN
                              PC   <= IR(9 DOWNTO 0);
                         END IF;
                    END IF;
                    STATE <= FETCH;

               WHEN EX_OR =>
                 AC      <= AC OR MDR;
                 STATE <= FETCH;
```

```
                WHEN EX_XOR =>
                  AC      <= AC XOR MDR;
                  STATE <= FETCH;

                WHEN EX_ADDI =>
                          AC   <= AC +
(IR(9)&IR(9)&IR(9)&IR(9)&IR(9)&IR(9)&IR(9 DOWNTO 0));

                     STATE <= FETCH;

                WHEN EX_SHIFT =>
                     AC <= AC_SHIFTED;
                     STATE <= FETCH;

                WHEN EX_CALL =>
                     PC_STACK  <= PC;
                     PC          <= IR(9 DOWNTO 0);
                     STATE <= FETCH;

                WHEN EX_RETURN =>
                     PC <= PC_STACK;
                     STATE <= FETCH;

                WHEN EX_IN =>
                     AC <= IO_DATA;
                     STATE <= FETCH;

                WHEN EX_OUT =>
                     IO_WRITE_INT  <= '1';
                     STATE <= EX_OUT2;

                WHEN EX_OUT2 =>
                     IO_WRITE_INT  <= '0';
                     STATE <= FETCH;


            WHEN OTHERS =>
              STATE <= FETCH;          -- If an invalid state is
reached, return to FETCH
          END CASE;
        END IF;
      END PROCESS;
  END a;
```