;LAB8\_PRELAB.ASM (ASSEMBLY)

;This code adds value to accumulator

;Vikram Jain

;ECE2031 L7

;10/31/2013

ORG 0

MAIN: CALL CALC

JUMP MAIN

ORG &H0010

CALC: LOAD B ;Load value stored in B

AND A ;And value stored in C

XOR C ;XOR value stored in D

STORE D ;Store value in D

RETURN

ORG &H030 ;set constants and variables

A: DW &H00FF ;Define A

B: DW &HA5A5 ;Define B

C: DW &H3300 ;Define C

D: DW &H0008 ;Define D

**Figure 1.** Incremental assembly code.

;LAB8\_PRELAB.MIF

;This code adds value to accumulator

;Vikram Jain

;ECE2031 L7

;10/31/2013

-- 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; -- MAIN: CALL CALC

001 : 1400; -- JUMP MAIN

010 : 0431; -- CALC: LOAD B ;Load value stored in B

011 : 2430; -- AND A ;And value stored in C

012 : 2C32; -- XOR C ;XOR value stored in D

013 : 0833; -- STORE D ;Store value in D

014 : 4400; -- RETURN

030 : 00FF; -- A: DW &H00FF ;Define A

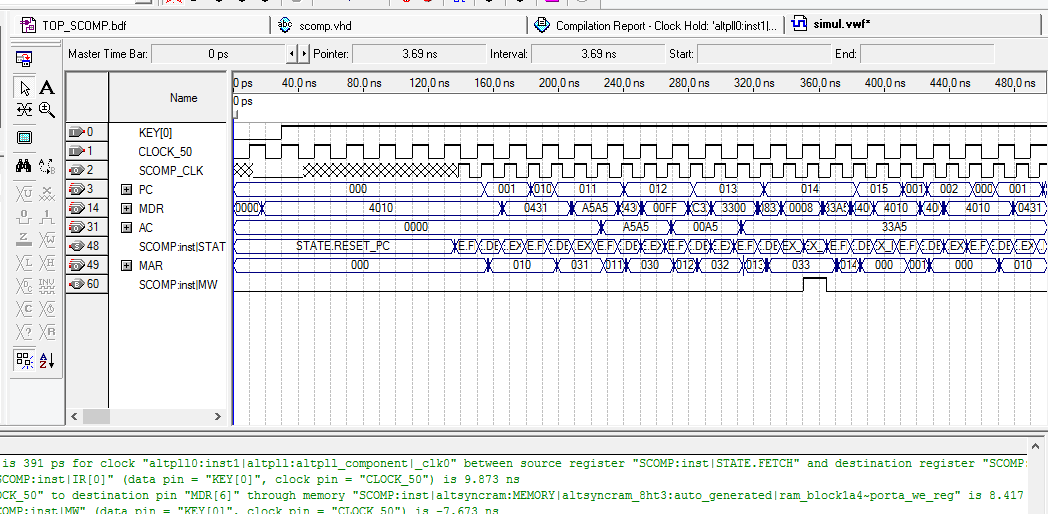
031 : A5A5; -- B: DW &HA5A5 ;Define B

032 : 3300; -- C: DW &H3300 ;Define C

033 : 0008; -- D: DW &H0008 ;Define D

END;

**Figure 2.** Incremental MIF code.



**Figure 3.** Timing Simulation of LAB8\_PRELAB.MIF running on simple computer.



**Figure 4.** Schematic of complete simple computer.

;LAB8.ASM (assembly)

;This code correctly modifies LEDs

;Vikram Jain

;ECE2031 L7

;10/31/2013

ORG 0

IN SWITCHES ;Load value stored in Switches

STORE INDATA ;Store value stored in INDATA

Start: LOAD INDATA ;Load INDATA

OUT LEDS ;write to leds

OUT SEVENSEG ;write to SEVENSEG

SHIFT 1 ;Shift it by 1

STORE INDATA ;Store new value in INDATA

LOAD ZERO ;Load a value of 0 into AC

OUT TIMER ;set timer to 0

Here: IN TIMER ;Read in timer forever

SUB TWOSECS ;Subtract 2

JNEG Here ;if timer is negative loop again

JUMP Start ;else change LEDS and SEVENSEG

ORG &H050 ;set constants and variables

SWITCHES: EQU &H00 ;constant value of Switches

LEDS: EQU &H01 ;constant value of LEDS

TIMER: EQU &H02 ;constant value of SEVENSEG

SEVENSEG: EQU &H04 ;constant value of TWOSECS

TWOSECS: DW &H0014 ;Define TWOSECS

INDATA: DW &H0000 ;Define INDATA

ZERO: DW &H0000 ;Define ZERO

**Figure 5.** Assembly code which lights up the next LED every 2 seconds.

;LAB8.MIF

;This code correctly modifies LEDs

;Vikram Jain

;ECE2031 L7

;10/31/2013

-- 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; --IN SWITCHES ;Load value stored in Switches

001 : 0851; --STORE INDATA ;Store value stored in INDATA

002 : 0451; --Start: LOAD INDATA ;Load INDATA

003 : 4C01; --OUT LEDS ;write to leds

004 : 4C04; --OUT SEVENSEG ;write to SEVENSEG

005 : 3001; --SHIFT 1 ;Shift it by 1

006 : 0851; --STORE INDATA ;Store new value in INDATA

007 : 0452; --LOAD ZERO ;Load a value of 0 into AC

008 : 4C02; --OUT TIMER ;set timer to 0

009 : 4802; --Here: IN TIMER ;Read in timer forever

00A : 1050; --SUB TWOSECS ;Subtract 2

00B : 1809; --JNEG Here ;if timer is negative loop again

00C : 1402; --JUMP Start ;else change LEDS and SEVENSEG

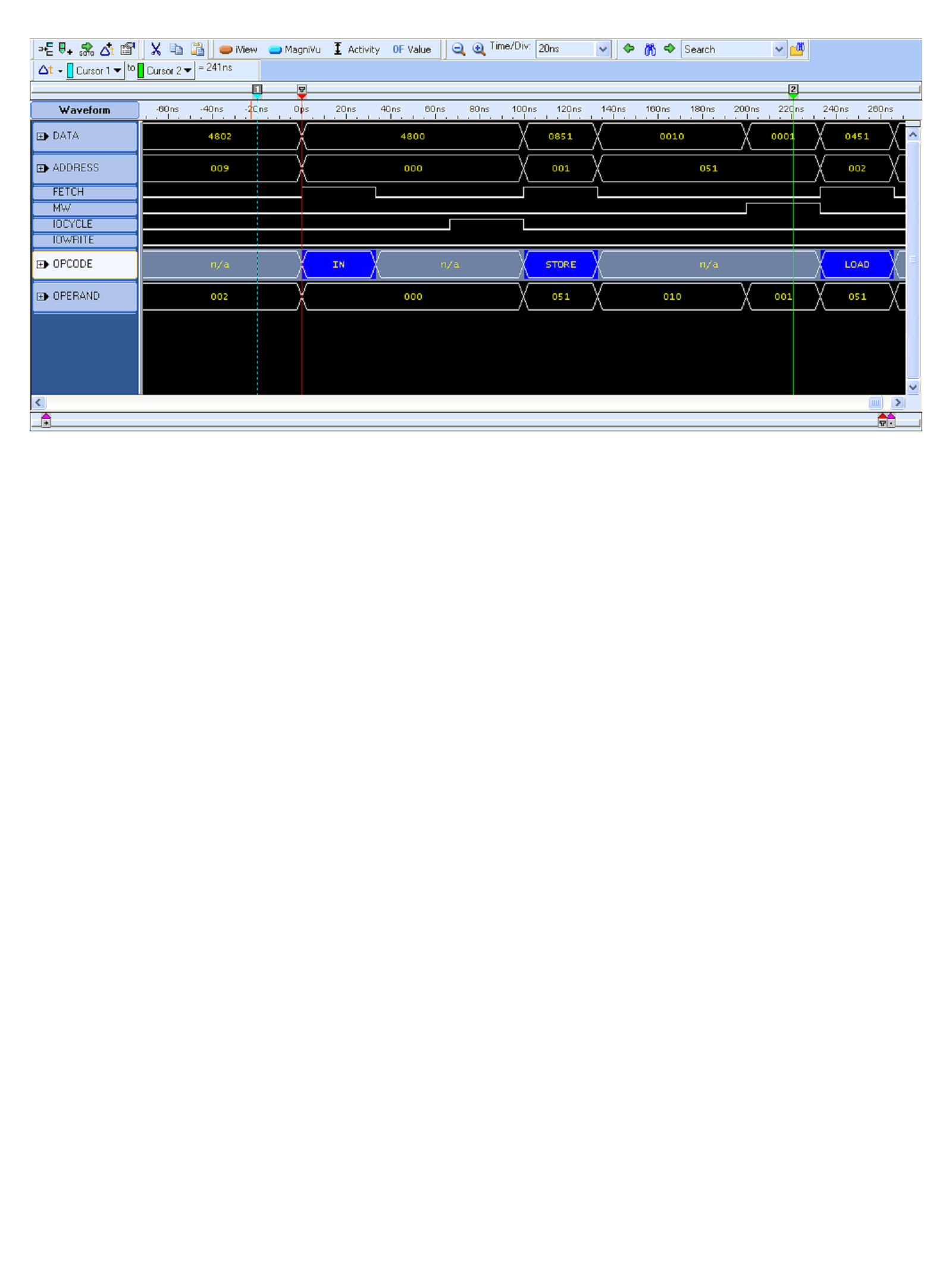
050 : 0014; --TWOSECS: DW &H0014 ;Define TWOSECS

051 : 0000; --INDATA: DW &H0000 ;Define INDATA

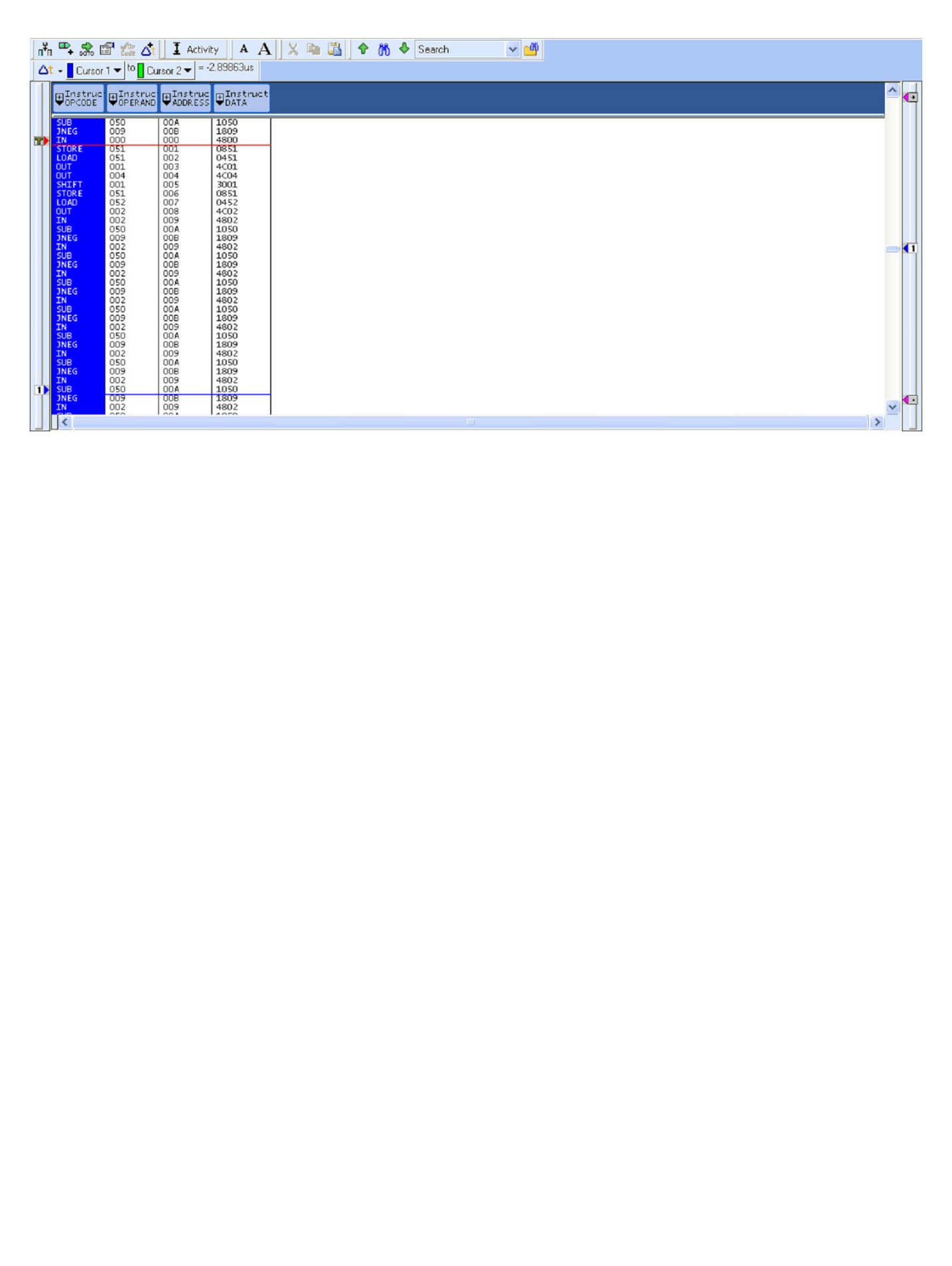
052 : 0000; --ZERO: DW &H0000 ;Define ZERO

END;

**Figure 6.** MIF code which lights up the next LED every 2 seconds.



**Figure 7.** Waveform depicting different states as LAB8.MIF is run on simple computer.



**Figure 8.** Filtered instruction list of LAB8.MIF running on simple computer.

**Appendix A: VHDL Code Implementing Simple Computer**

-- SCOMP.VHD (VHDL)

-- This code produces a simple computer

-- Vikram Jain

-- ECE2031 L09

-- 10/31/2013

-- State machine to simulate simple computer

LIBRARY IEEE;

LIBRARY ALTERA\_MF;

LIBRARY LPM;

USE LPM.LPM\_COMPONENTS.ALL;

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;

-- entity statement

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

ARCHITECTURE a OF SCOMP IS

-- declare states

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\_IN,

EX\_OUT,

EX\_OUT2,

EX\_RETURN

);

-- TYPE

SIGNAL STATE : STATE\_TYPE;

SIGNAL AC : STD\_LOGIC\_VECTOR(15 DOWNTO 0);

SIGNAL IR : STD\_LOGIC\_VECTOR(15 DOWNTO 0);

SIGNAL IO\_IN : 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 => "LAB8.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 LPM function to drive I/O bus

IO\_BUS: LPM\_BUSTRI

GENERIC MAP (

lpm\_width => 16

)

PORT MAP (

data => AC,

enabledt => IO\_WRITE\_INT,

tridata => IO\_DATA

);

-- Use LPM\_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

);

-- assignment statements

PC\_OUT <= PC;

MW\_OUT <= MW;

AC\_OUT <= AC;

MDR\_OUT <= MDR;

MAR\_OUT <= MEM\_ADDR;

IO\_ADDR <= IR(7 DOWNTO 0);

-- concurrent assignment statements

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

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\_OUT2, '1' WHEN EX\_OUT, '1' WHEN EX\_IN, '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;

IO\_WRITE\_INT <= '0';

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; -- SUB

WHEN "000110" =>

STATE <= EX\_JNEG; -- AC <0

WHEN "001000" =>

STATE <= EX\_JZERO; -- AC =0

WHEN "001010" =>

STATE <= EX\_OR; -- OR

WHEN "001101" =>

STATE <= EX\_ADDI; -- ADD IMMEDIATE

WHEN "001011" =>

STATE <= EX\_XOR; -- XOR

WHEN "000111" =>

STATE <= EX\_JPOS; -- AC > 0

WHEN "001100" => -- SHIFT (decode)

STATE <= EX\_SHIFT;

WHEN "010000" =>

STATE <= EX\_CALL; -- CALL

WHEN "010001" =>

STATE <= EX\_RETURN; -- RETURN

WHEN "010010" =>

STATE <= EX\_IN; -- IN

WHEN "010011" =>

IO\_WRITE\_INT <= '1';

STATE <= EX\_OUT;

WHEN OTHERS =>

STATE <= FETCH; -- Invalid opcodes default to NOP

END CASE;

WHEN EX\_LOAD =>

AC <= MDR; -- Latch data from MDR 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 => -- ADD (execute)

AC <= AC + MDR;

STATE <= FETCH;

WHEN EX\_JUMP => -- JUMP (execute)

PC <= IR(9 DOWNTO 0);

STATE <= FETCH;

WHEN EX\_AND => -- AND (execute)

AC <= AC AND MDR;

STATE <= FETCH;

WHEN EX\_SUB => -- SUB (execute)

AC <= AC - MDR;

STATE<=FETCH;

WHEN EX\_JNEG => -- JUMP if Negative (execute)

If (AC(15) = '1') then

PC<=IR(9 DOWNTO 0);

end if;

STATE<=FETCH;

WHEN EX\_JPOS => -- JUMP if Positive (execute)

If ((AC(15) = '0') AND (AC /=x"0000")) then

PC<=IR(9 DOWNTO 0);

end if;

STATE<=FETCH;

WHEN EX\_JZERO => -- JUMP if 0 (execute)

If (AC =x"0000") then

PC<=IR(9 DOWNTO 0);

end if;

STATE<=FETCH;

WHEN EX\_OR => -- OR (execute)

AC<=AC OR MDR;

STATE<=FETCH;

WHEN EX\_XOR => -- XOR (execute)

AC<=AC XOR MDR;

STATE<=FETCH;

WHEN EX\_ADDI => -- ADDI (execute)

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

STATE<=FETCH;

WHEN EX\_SHIFT => -- SHIFT (execute)

AC <= AC\_SHIFTED;

STATE <= FETCH;

WHEN EX\_CALL => -- CALL (execute)

PC\_STACK <= PC;

PC <= IR(9 DOWNTO 0);

STATE <= FETCH;

WHEN EX\_RETURN => -- RETURN (execute)

PC <= PC\_STACK;

STATE <= FETCH;

WHEN EX\_IN => -- IN (execute)

AC <= IO\_DATA;

STATE <= FETCH;

WHEN EX\_OUT => -- OUT (execute)

STATE <= EX\_OUT2;

WHEN EX\_OUT2 => -- OUT2 (execute)

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;