Paul Kafka

10/21/13

Embedded System Design

Homework 5

**VHDL**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

USE IEEE.STD\_LOGIC\_ARITH.ALL;

USE IEEE.STD\_LOGIC\_UNSIGNED.ALL;

LIBRARY altera\_mf;

USE altera\_mf.altera\_mf\_components.all;

ENTITY SCOMP IS

PORT( clock, reset : IN STD\_LOGIC;

program\_counter\_out : OUT STD\_LOGIC\_VECTOR( 7 DOWNTO 0 );

register\_AC\_out : OUT STD\_LOGIC\_VECTOR(15 DOWNTO 0 );

memory\_data\_register\_out : OUT STD\_LOGIC\_VECTOR(15 DOWNTO 0 );

memory\_address\_register\_out : OUT STD\_LOGIC\_VECTOR(7 DOWNTO 0 ));

END SCOMP;

ARCHITECTURE a OF scomp IS

TYPE STATE\_TYPE IS (

*------------ADDED-----------------------------------------------*

execute\_subt, execute\_xor, execute\_or, execute\_and, execute\_addi,

*-----------------------------------------------------------------*

reset\_pc, fetch, decode, execute\_add, execute\_load,

execute\_store, execute\_store3, execute\_store2, execute\_jump );

SIGNAL state: STATE\_TYPE;

SIGNAL instruction\_register, memory\_data\_register : STD\_LOGIC\_VECTOR(15 DOWNTO 0 );

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

SIGNAL program\_counter : STD\_LOGIC\_VECTOR( 7 DOWNTO 0 );

SIGNAL memory\_address\_register : STD\_LOGIC\_VECTOR( 7 DOWNTO 0 );

SIGNAL memory\_write : STD\_LOGIC;

BEGIN

*-- Use Altsyncram function for computer's memory (256 16-bit words)*

memory: altsyncram

GENERIC MAP (

operation\_mode => "SINGLE\_PORT",

width\_a => 16,

widthad\_a => 8,

lpm\_type => "altsyncram",

outdata\_reg\_a => "UNREGISTERED",

*-- Reads in mif file for initial program and data values*

init\_file => "program.mif",

intended\_device\_family => "Cyclone")

PORT MAP ( wren\_a => memory\_write, clock0 => clock,

address\_a => memory\_address\_register, data\_a => Register\_AC,

q\_a => memory\_data\_register );

*-- Output major signals for simulation*

program\_counter\_out <= program\_counter;

register\_AC\_out <= register\_AC;

memory\_data\_register\_out <= memory\_data\_register;

memory\_address\_register\_out <= memory\_address\_register;

PROCESS ( CLOCK, RESET )

BEGIN

IF reset = '1' THEN

state <= reset\_pc;

ELSIF clock'EVENT AND clock = '1' THEN

CASE state IS

*-- reset the computer, need to clear some registers*

WHEN reset\_pc =>

program\_counter <= "00000000";

*-- memory\_address\_register <= "00000000";*

register\_AC <= "0000000000000000";

memory\_write <= '0';

state <= fetch;

*-- Fetch instruction from memory and add 1 to PC*

WHEN fetch =>

instruction\_register <= memory\_data\_register;

program\_counter <= program\_counter + 1;

memory\_write <= '0';

state <= decode;

*-- Decode instruction and send out address of any data operands*

WHEN decode =>

CASE instruction\_register( 15 DOWNTO 8 ) IS

WHEN X"00" =>

state <= execute\_add;

WHEN X"01" =>

state <= execute\_store;

WHEN X"02" =>

state <= execute\_load;

WHEN X"03" =>

state <= execute\_jump;

*----------Added--------------------------*

WHEN X"05" =>

state <= execute\_subt;

WHEN X"06" =>

state <= execute\_xor;

WHEN X"07" =>

state <= execute\_or;

WHEN X"08" =>

state <= execute\_and;

WHEN X"0B" =>

state <= execute\_addi;

*-----------------------------------------*

WHEN OTHERS =>

state <= fetch;

END CASE;

*-----------------ADDED------------------------------------------------*

WHEN execute\_subt=>

register\_ac <= register\_ac - memory\_data\_register;

state <= fetch;

WHEN execute\_xor=>

register\_ac <= register\_ac XOR memory\_data\_register;

state <= fetch;

WHEN execute\_or=>

register\_ac <= register\_ac OR memory\_data\_register;

state <= fetch;

WHEN execute\_and=>

register\_ac <= register\_ac AND memory\_data\_register;

state <= fetch;

WHEN execute\_addi=>

if memory\_data\_register =X"FF" THEN

register\_ac <= register\_ac - 1;

else

register\_ac <= register\_ac + memory\_data\_register;

state <= fetch;

END IF;

*---------------------------------------------------------------------*

*-- Execute the ADD instruction*

WHEN execute\_add =>

register\_ac <= register\_ac + memory\_data\_register;

state <= fetch;

*-- Execute the STORE instruction*

*-- (needs three clock cycles for memory write)*

WHEN execute\_store =>

*-- write register\_AC to memory*

memory\_write <= '1';

state <= execute\_store2;

*-- This state ensures that the memory address is*

*-- valid until after memory\_write goes inactive*

WHEN execute\_store2 =>

memory\_write <= '0';

state <= execute\_store3;

WHEN execute\_store3 =>

state <= fetch;

*-- Execute the LOAD instruction*

WHEN execute\_load =>

register\_ac <= memory\_data\_register;

state <= fetch;

*-- Execute the JUMP instruction*

WHEN execute\_jump =>

program\_counter <= instruction\_register( 7 DOWNTO 0 );

state <= fetch;

WHEN OTHERS =>

state <= fetch;

END CASE;

END IF;

END PROCESS;

*-- memory address register is stored inside synchronous memory unit*

*-- need to send it's outputs based on current state*

WITH state SELECT

memory\_address\_register <= "00000000" WHEN reset\_pc,

program\_counter WHEN fetch,

instruction\_register(7 DOWNTO 0) WHEN decode,

program\_counter WHEN execute\_add,

*---------------ADDED--------------------------------*

program\_counter WHEN execute\_subt,

program\_counter WHEN execute\_xor,

program\_counter WHEN execute\_or,

program\_counter WHEN execute\_and,

program\_counter WHEN execute\_addi,

*----------------------------------------------------*

instruction\_register(7 DOWNTO 0) WHEN execute\_store,

instruction\_register(7 DOWNTO 0) WHEN execute\_store2,

program\_counter WHEN execute\_store3,

program\_counter WHEN execute\_load,

instruction\_register(7 DOWNTO 0) WHEN execute\_jump;

END a;

**MIF File**

DEPTH = 256; % Memory depth and width are required %

WIDTH = 16; % Enter a decimal number %

ADDRESS\_RADIX = HEX; % Address and value radixes are optional %

DATA\_RADIX = HEX; % Enter BIN, DEC, HEX, or OCT; unless %

% otherwise specified, radixes = HEX %

-- Specify values for addresses, which can be single address or range

CONTENT

BEGIN

[00..FF] : 0000; % Range--Every address from 00 to FF = 0000 (Default) %

---------Test for the ADD instruction -------------------------------------

00 :0260; % LOAD AC with MEM(60)=B=AAAA %

01 :0061; % ADD MEM(61)=C=5555 to AC %

02 :0162; % STORE AC in MEM(62)=A %

03 :0262; % LOAD AC with MEM(62) check for new value of FFFF %

---------Test for the SUBI instruction ------------------------------------

04 :0261; % Load AC with MEM(61)=C %

05 :0563; % Test C-D = 2222. 05 is minus opcode. 63 is D's Address %

06 :0164; % STORE AC in MEM(64)=E %

07 :0264; % LOAD AC with MEM(64) check for new value of 2222 %

---------Test for the XOR instruction -------------------------------------

08 :0261; % Load AC with MEM(61)=C %

09 :0663; % Test C XOR D = 6666. 06 is XOR opcode. 63 is D's Address %

0A :0165; % STORE AC in MEM(65)=F %

0B :0265; % LOAD AC with MEM(65) check for new value of 6666 %

---------Test for the OR instruction --------------------------------------

0C :0261; % Load AC with MEM(61)=C %

0D :0763; % Test C OR D= 7777. 07 is OR opcode. 63 is D's Address %

0E :0166; % STORE AC in MEM(66)=G %

0F :0266; % LOAD AC with MEM(66) check for new value of 7777 %

---------Test for the AND instruction -------------------------------------

10 :0261; % Load AC with MEM(61)=C %

11 :0863; % Test C AND D = 1111. 08 is AND opcode. 63 is D's Address %

12 :0167; % STORE AC in MEM(67)=H %

13 :0267; % LOAD AC with MEM(67) check for new value of 1111 %

---------Test for the ANDI instruction ------------------------------------

14 :0261; % Load AC with MEM(61)=C %

15 :0BFF; % Test C ANDi FF = 5554. 0B is ANDI opcode. FF is the integer %

16 :0168; % STORE AC in MEM(68)=I %

17 :0268; % LOAD AC with MEM(68) check for new value of 5554 %

---------------------------------------------------------------------------

18 :0318; % JUMP to same line (loop forever) 03 is jump opcode. 18 is same line %

---------------------------------------------------------------------------

60 :AAAA; % Data Value of B %

61 :5555; % Data Value of C %

62 :0000; % Data Value of A - should be FFFF after running program %

63 :3333; % Data Value of D %

64 :0000; % Data value of E - should be 2222 after running program %

65 :0000; % Data value of F - should be 6666 after running program %

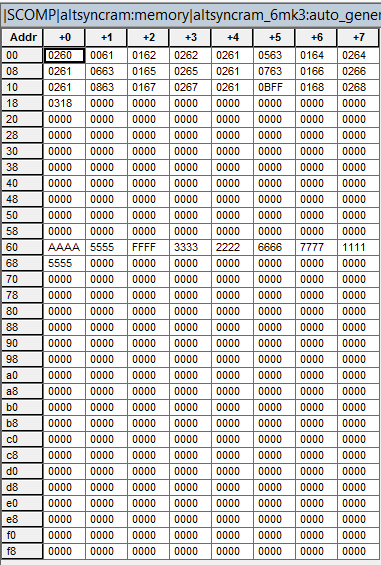
66 :0000; % Data value of G - should be 7777 after running program %

67 :0000; % Data value of H - should be 1111 after running program %

68 :0000; % Data value of I - should be 5554 after running program %

END ;

**Memory After Simulation**



**Simulation**

