library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.std\_logic\_unsigned.all;

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

-------------------------------------- CONNECTING ALL STAGES TO REGISTERS--------------------------------------------

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

entity pipeline is

port(clk: in std\_logic);

end entity;

architecture top of pipeline is

component fetch is

port( clk: in std\_logic;

fetch\_reg\_alu: in std\_logic\_vector(7 downto 0); --

fetch\_deop\_alu: in std\_logic\_vector(3 downto 0);--

fetch\_offset\_sig: in std\_logic\_vector(7 downto 0);--

fetch\_pcjump\_sig: in std\_logic\_vector(7 downto 0);--

fetch\_instructions: out std\_logic\_vector(15 downto 0)--

);

end component;

component decoder is

port( instructions: in std\_logic\_vector( 15 downto 0); -- input from the instruction memory.

--Opperation Codes-----------------

opcode : out std\_logic\_vector(3 downto 0);

sel : out std\_logic\_vector(3 downto 0);

imm\_val : out std\_logic\_vector(7 downto 0); -- immediate value being added to RX. Goes to RY Mux

--Load and Store line

rl\_addr: out std\_logic\_vector( 7 downto 0); --

---to decoder

rst : out std\_logic; -- reset for the Register bank

rdx\_en: out std\_logic; -- enables the read process for the Register bank

rdy\_en: out std\_logic; -- enables the read process for the Register bank

rdx : out std\_logic\_vector(3 downto 0); -- Rx output from Decoder to the register bank

rdy : out std\_logic\_vector(3 downto 0); -- Ry output from Decoder to the register bank

wr\_en : out std\_logic; -- enable for the right process to the Register bank

wrd : out std\_logic\_vector(3 downto 0); -- Write address to the Register bank

-- Select line for the MUXs

sel\_rymux : out std\_logic;

sel\_rlsmux: out std\_logic;

sel\_wbmux : out std\_logic;

offset: out std\_logic\_vector(7 downto 0);

pcjump: out std\_logic\_vector(7 downto 0)

-- Will need interupt lines

);

end component;

component execute is

port( clk: in std\_logic;

execute\_fetch\_reg\_alu : out std\_logic\_vector(7 downto 0);

execute\_rlsmux\_sel: in std\_logic;

execute\_deop\_alu\_in: in std\_logic\_vector(3 downto 0);

execute\_dsel\_alu\_in:in std\_logic\_vector(3 downto 0);

execute\_decode\_mux2:in std\_logic\_vector(7 downto 0);

execute\_rst\_sig : in std\_logic;

execute\_rxen\_sig : in std\_logic;

execute\_rdx\_rx : in std\_logic\_vector(3 downto 0);

execute\_ryen\_sig : in std\_logic;

execute\_rdy\_ry : in std\_logic\_vector(3 downto 0);

execute\_writen\_sig: in std\_logic;

execute\_wrd\_waddr : in std\_logic\_vector(3 downto 0);

execute\_decode\_mux1:in std\_logic\_vector(7 downto 0);

execute\_ry\_sel : in std\_logic;

execute\_deop\_alu\_out: out std\_logic\_vector(3 downto 0);

execute\_reg\_alu : out std\_logic\_vector(7 downto 0);

execute\_mux2\_dmem : out std\_logic\_vector(7 downto 0);

execute\_alu\_mux3 : out std\_logic\_vector(7 downto 0);

execute\_mux3\_reg : in std\_logic\_vector(7 downto 0)

);

end component;

component writeback is

PORT

(

writeback\_reg\_alu : IN std\_logic\_vector(7 DOWNTO 0); --memd from Rx (register bank)

writeback\_mux2\_dmem : IN std\_logic\_vector(7 DOWNTO 0);--dAddress from Mux

writeback\_deop\_alu: in std\_logic\_vector(3 downto 0); -- opcode coming in

writeback\_clk\_sig : IN std\_logic;

--PC : IN std\_logic\_vector ( DATA\_WIDTH - 1 DOWNTO 0); -- PC COUNTER INPUT

writeback\_alu\_mux3: in std\_logic\_vector(7 downto 0);

writeback\_mux3\_selectline: in std\_logic;

writeback\_mux3\_reg: out std\_logic\_vector( 7 downto 0)

);

end component;

component reg1 is

port( clk : in std\_logic;

reg\_alu\_in: in std\_logic\_vector(7 downto 0);

reg\_alu\_out: out std\_logic\_vector(7 downto 0);

deop\_alu\_in: in std\_logic\_vector(3 downto 0);

deop\_alu\_out: out std\_logic\_vector(3 downto 0);

offset\_sig\_in: in std\_logic\_vector(7 downto 0);

offset\_sig\_out: out std\_logic\_vector(7 downto 0);

pcjump\_sig\_in: in std\_logic\_vector(7 downto 0);

pcjump\_sig\_out: out std\_logic\_vector(7 downto 0);

instructions\_in: in std\_logic\_vector(15 downto 0);

instructions\_out: out std\_logic\_vector(15 downto 0)

);

end component ;

component reg2 is

port( sel\_rlsmux\_in: in std\_logic;

sel\_rlsmux\_out: out std\_logic;

opcode\_in:in std\_logic\_vector(3 downto 0);

opcode\_out:out std\_logic\_vector(3 downto 0);

sel\_in: in std\_logic\_vector(3 downto 0);

sel\_out: out std\_logic\_vector(3 downto 0);

rl\_addr\_in: in std\_logic\_vector(7 downto 0);

rl\_addr\_out: out std\_logic\_vector(7 downto 0);

rst\_in: in std\_logic;

rst\_out: out std\_logic;

rdx\_in: in std\_logic\_vector(3 downto 0);

rdx\_out: out std\_logic\_vector(3 downto 0);

rdx\_en\_in: in std\_logic;

rdx\_en\_out: out std\_logic;

rdy\_in: in std\_logic\_vector(3 downto 0);

rdy\_out: out std\_logic\_vector( 3 downto 0);

rdy\_en\_in: in std\_logic;

rdy\_en\_out: out std\_logic;

wr\_en\_in: in std\_logic;

wr\_en\_out: out std\_logic;

wrd\_in: in std\_logic\_vector(3 downto 0);

wrd\_out: out std\_logic\_vector(3 downto 0);

imm\_val\_in: in std\_logic\_vector( 7 downto 0);

imm\_val\_out: out std\_logic\_vector(7 downto 0);

sel\_rymux\_in: in std\_logic;

sel\_rymux\_out: out std\_logic;

sel\_wbmux\_in: in std\_logic;

sel\_wbmux\_out: out std\_logic;

clk: in std\_logic

);

end component;

component reg3 is

port( clk : in std\_logic;

deop\_alu\_in3 : in std\_logic\_vector(3 downto 0);

deop\_alu\_out3 : out std\_logic\_vector(3 downto 0);

reg\_alu\_in3 : in std\_logic\_vector(7 downto 0);

reg\_alu\_out3 : out std\_logic\_vector(7 downto 0);

mux2\_dmem\_in : in std\_logic\_vector(7 downto 0);

mux2\_dmem\_out : out std\_logic\_vector(7 downto 0);

alu\_mux3\_in : in std\_logic\_vector(7 downto 0);

alu\_mux3\_out : out std\_logic\_vector(7 downto 0);

mux3\_selectline\_in : in std\_logic;

mux3\_selectline\_out : out std\_logic;

mux3\_reg\_in : in std\_logic\_vector(7 downto 0);

mux3\_reg\_out : out std\_logic\_vector(7 downto 0)

);

end component;

-----------------------Reg1 SIGNALS-------------------------------

signal reg\_alu\_sig\_out: std\_logic\_vector(7 downto 0);

signal reg\_alu\_sig\_in: std\_logic\_vector(7 downto 0);

signal deop\_alu\_sig\_out: std\_logic\_vector(3 downto 0);

-- signal declared as opcode\_sig\_in

signal offset\_sig\_out: std\_logic\_vector(7 downto 0);

signal offset\_sig\_in: std\_logic\_vector(7 downto 0);

signal pcjump\_sig\_in: std\_logic\_vector(7 downto 0);

signal pcjump\_sig\_out: std\_logic\_vector(7 downto 0);

signal instructions: std\_logic\_vector(15 downto 0);

signal instr\_sig: std\_logic\_vector (15 downto 0);

-----------------------reg2 SIGNALS-------------------------------

signal sel\_rlsmux\_sig\_in: std\_logic;

signal sel\_rlsmux\_sig\_out:std\_logic;

signal opcode\_sig\_in: std\_logic\_vector(3 downto 0);

signal opcode\_sig\_out: std\_logic\_vector(3 downto 0);

signal sel\_sig\_in: std\_logic\_vector(3 downto 0);

signal sel\_sig\_out: std\_logic\_vector(3 downto 0);

signal rl\_addr\_sig\_in: std\_logic\_vector(7 downto 0);

signal rl\_addr\_sig\_out: std\_logic\_vector(7 downto 0);

signal rst\_sig\_in: std\_logic;

signal rst\_sig\_out: std\_logic;

signal rdx\_sig\_in: std\_logic\_vector(3 downto 0);

signal rdx\_sig\_out: std\_logic\_vector(3 downto 0);

signal rdx\_en\_sig\_in: std\_logic;

signal rdx\_en\_sig\_out: std\_logic;

signal rdy\_sig\_in: std\_logic\_vector(3 downto 0);

signal rdy\_sig\_out: std\_logic\_vector( 3 downto 0);

signal rdy\_en\_sig\_in: std\_logic;

signal rdy\_en\_sig\_out: std\_logic;

signal wr\_en\_sig\_in: std\_logic;

signal wr\_en\_sig\_out: std\_logic;

signal wrd\_sig\_in: std\_logic\_vector(3 downto 0);

signal wrd\_sig\_out: std\_logic\_vector(3 downto 0);

signal imm\_val\_sig\_in: std\_logic\_vector( 7 downto 0);

signal imm\_val\_sig\_out: std\_logic\_vector(7 downto 0);

signal sel\_rymux\_sig\_in: std\_logic;

signal sel\_rymux\_sig\_out: std\_logic;

signal sel\_wbmux\_sig\_in: std\_logic;

signal sel\_wbmux\_sig\_out: std\_logic;

----------------------------reg3 SIGNALS--------------------------

Signal deop\_alu\_sig\_in3 :std\_logic\_vector(3 downto 0);

Signal deop\_alu\_sig\_out3 :std\_logic\_vector(3 downto 0);

Signal reg\_alu\_sig\_in3 : std\_logic\_vector(7 downto 0);

Signal reg\_alu\_sig\_out3 : std\_logic\_vector(7 downto 0);

Signal mux2\_dmem\_sig\_in :std\_logic\_vector(7 downto 0);

Signal mux2\_dmem\_sig\_out :std\_logic\_vector(7 downto 0);

Signal alu\_mux3\_sig\_in :std\_logic\_vector(7 downto 0);

Signal alu\_mux3\_sig\_out :std\_logic\_vector(7 downto 0);

Signal mux3\_selectline\_sig\_in :std\_logic;

Signal mux3\_selectline\_sig\_out :std\_logic;

Signal mux3\_reg\_sig\_in : std\_logic\_vector(7 downto 0);

Signal mux3\_reg\_sig\_out :std\_logic\_vector(7 downto 0);

component mux is

port(

a: in std\_logic\_vector(7 downto 0);

b: in std\_logic\_vector(7 downto 0);

sel: in std\_logic;

output: out std\_logic\_vector(7 downto 0)

) ;

end component;

signal a: std\_logic\_vector(7 downto 0);

signal b: std\_logic\_vector(7 downto 0);

signal sel: std\_logic;

signal output: std\_logic\_vector(7 downto 0) ;

begin

fetch1:fetch port map(clk,reg\_alu\_sig\_out,deop\_alu\_sig\_out,offset\_sig\_out,pcjump\_sig\_out,instructions);

reg11:reg1 port map(clk,reg\_alu\_sig\_in, reg\_alu\_sig\_out,opcode\_sig\_in,deop\_alu\_sig\_out,offset\_sig\_in,offset\_sig\_out,pcjump\_sig\_in,pcjump\_sig\_out,instructions,instr\_sig);

decode1: decoder port map(instr\_sig,opcode\_sig\_in,sel\_sig\_in,imm\_val\_sig\_in,rl\_addr\_sig\_in, rst\_sig\_in,rdx\_en\_sig\_in, rdy\_en\_sig\_in,

rdx\_sig\_in,rdy\_sig\_in,wr\_en\_sig\_in,wrd\_sig\_in,sel\_rymux\_sig\_in,sel\_rlsmux\_sig\_in,sel\_wbmux\_sig\_in,offset\_sig\_in,pcjump\_sig\_in);

reg22: reg2 port map(sel\_rlsmux\_sig\_in,sel\_rlsmux\_sig\_out,opcode\_sig\_in,opcode\_sig\_out,sel\_sig\_in,sel\_sig\_out,rl\_addr\_sig\_in,rl\_addr\_sig\_out,rst\_sig\_in,rst\_sig\_out,

rdx\_sig\_in,rdx\_sig\_out,rdx\_en\_sig\_in,rdx\_en\_sig\_out,rdy\_sig\_in,rdy\_sig\_out,rdy\_en\_sig\_in,rdy\_en\_sig\_out,wr\_en\_sig\_in,wr\_en\_sig\_out,wrd\_sig\_in,wrd\_sig\_out,imm\_val\_sig\_in,imm\_val\_sig\_out,sel\_rymux\_sig\_in,

sel\_rymux\_sig\_out,sel\_wbmux\_sig\_in,sel\_wbmux\_sig\_out,clk);

execute1: execute port map(clk,reg\_alu\_sig\_in,sel\_rlsmux\_sig\_out,opcode\_sig\_out,sel\_sig\_out,rl\_addr\_sig\_out,rst\_sig\_out,rdx\_en\_sig\_out,rdx\_sig\_out,rdy\_en\_sig\_out,rdy\_sig\_out,wr\_en\_sig\_out,wrd\_sig\_out,imm\_val\_sig\_out,sel\_rymux\_sig\_out,deop\_alu\_sig\_in3,reg\_alu\_sig\_in3,mux2\_dmem\_sig\_in,alu\_mux3\_sig\_in,mux3\_reg\_sig\_out);

reg33: reg3 port map(clk,deop\_alu\_sig\_in3,deop\_alu\_sig\_out3,reg\_alu\_sig\_in3,reg\_alu\_sig\_out3,mux2\_dmem\_sig\_in,mux2\_dmem\_sig\_out,alu\_mux3\_sig\_in,alu\_mux3\_sig\_out,sel\_wbmux\_sig\_out,mux3\_selectline\_sig\_out,mux3\_reg\_sig\_in,mux3\_reg\_sig\_out);

writeback1: writeback port map(reg\_alu\_sig\_out3,mux2\_dmem\_sig\_out,deop\_alu\_sig\_out3,clk,alu\_mux3\_sig\_out,mux3\_selectline\_sig\_out,mux3\_reg\_sig\_in);

end architecture top;