library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.std\_logic\_unsigned.all;

entity main is

port (clk: in std\_logic;

reset: in std\_logic:='0'

);

end main;

architecture top of main is

component alu is

port(

rx:in std\_logic\_vector(7 downto 0); -- input 1

ry: in std\_logic\_vector(7 downto 0); -- input 2

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

sele: in std\_logic\_vector (3 downto 0); --select line

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

--branchen: out std\_logic-- branch enable line

--- zero: out std\_logic

);

end component alu;

component datamem IS

PORT

(

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

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

aluop: in std\_logic\_vector(3 downto 0); -- opcode coming in

q : OUT std\_logic\_vector(7 DOWNTO 0);-- output

clk : IN std\_logic

);

END component datamem;

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 mux;

component regbank is

port( clk : in std\_logic;

rst : in std\_logic;

write\_en : in std\_logic;--\ enables for

rx\_en : in std\_logic;---> Write,Read rx 1 and ry 2.

ry\_en : in std\_logic;--/ comes from the decoder

rx\_addr : in std\_logic\_vector(3 downto 0);--Register addresses for Rx to be read

ry\_addr : in std\_logic\_vector(3 downto 0);--Register addresses for RY to be read

write\_addr : in std\_logic\_vector(3 downto 0);--Regbank address for where write\_data will be written

write\_data : in std\_logic\_vector(7 downto 0);--data being written into the regbank

rx\_data : out std\_logic\_vector(7 downto 0);--

ry\_data : out std\_logic\_vector(7 downto 0)

);

end component;

component decoder is

port (

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

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;

--PC counter lines

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

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

-- Will need interupt lines

);

end component decoder;

component imem is

port(

inst\_addr : in std\_logic\_vector(7 downto 0); -- input from PC

clk : in std\_logic;

instruction : out std\_logic\_vector(15 downto 0) -- output to go to decoder

);

end component imem;

signal clk\_sig: std\_logic;

component pcbranch is

port( rx\_in : in std\_logic\_vector(7 downto 0); -- connects to rx from regbank from regbank

opcode : in std\_logic\_vector(3 downto 0); --connects to alu\_opcode not select lines from decoder

offset : in std\_logic\_vector(7 downto 0); -- connects to branch from decoder

PC\_jump : in std\_logic\_vector(7 downto 0); --jump address for the PC from the Decoder.

clk : in std\_logic;

pcout : out std\_logic\_vector(7 downto 0) --outputs to instruction mem

);

end component;

------------------------signals for ALU-------------------------------------

signal reg\_alu: std\_logic\_vector(7 downto 0); -- Regbank to ALU rx

--signal alu\_pc: std\_logic; -- ALU - Branch Enable to PC- ALU OUT

signal mux1\_alu: std\_logic\_vector(7 downto 0); -- mux1 RY to ALU ry

signal alu\_mux3: std\_logic\_vector(7 downto 0); -- Alu output to Mux 3

signal deop\_alu: std\_logic\_vector(3 downto 0); -- decoder opcode to alu opcode

signal desel\_alu: std\_logic\_vector(3 downto 0);-- decoder sel to alu sel

--------------------------signals for dmem---------------------------------

-- output q signal defined

--(deop\_alu) decoder opcode to dmem aluop

signal mux2\_dmem: std\_logic\_vector(7 downto 0); -- mux2 output to datamem address

--signal reg\_dmem: std\_logic\_vector(7 downto 0); -- reg rx to dmem input

-- (clk\_sig)

---------------------------signals for mux1 ------------------------------

--input from reg RY defined in reg(reg\_mux1) A

signal decode\_mux1: std\_logic\_vector(7 downto 0); -- decoder add\_imm to B input of mux1

signal ry\_sel: std\_logic; -- decoder sel\_rymux to sel of mux 1

--output to ALU defined in alu(mux1\_alu)

--------------------------signals for mux2---------------------------------

signal decode\_mux2: std\_logic\_vector (7 downto 0); -- decoder rl\_addr to Mux2 A input

--signal ry\_mux2: std\_logic\_vector(7 downto 0); -- reg ry\_data to Mux2 B input

signal rlsmux\_sel: std\_logic; -- decoder (sel\_rlsmux) to mux2 sel

-- mux2 output defined

--------------------------signals for mux3---------------------------------

-- alu\_mux3 already defined

signal mux3\_reg: std\_logic\_vector(7 downto 0); -- mux3 output writeback to Reg write\_data

signal dmem\_mux3: std\_logic\_vector(7 downto 0); -- dmem q to mux3 input

signal mux3\_selectline: std\_logic; -- select line for mux3 dmem to mux3

------------------------------signals for Regbank----------------------

--clock defined already

signal rst\_sig : std\_logic; -- reset from decoder to register bank

signal writen\_sig: std\_logic; -- decoder wr\_en to register bank write\_en

signal rxen\_sig: std\_logic; -- decoder rdx enable to regbank rx\_en

signal ryen\_sig: std\_logic; -- decoder rdy enable to regbank ry\_en

signal rdx\_rx: std\_logic\_vector(3 downto 0); -- decoder rdx to regbank rx\_addr

signal rdy\_ry: std\_logic\_vector(3 downto 0); -- decoder rdy to regbank ry\_addr

signal wrd\_waddr: std\_logic\_vector(3 downto 0); --decoder wrd to regbank write\_addr

--(mux3\_reg)

-- Rx output declared in alu(reg\_alu)

signal reg\_mux1: std\_logic\_vector(7 downto 0); -- reg RY output to mux1

---------------------------signals for imem------------------------

signal iaddress: std\_logic\_vector(7 downto 0); -- PC to imem

-- (clk\_sig)

-- (instr\_sig) imem to decoder

----------------------------signals for decoder---------------------

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

--decoder opcode(deop\_alu) to branch, alu, dmem

-- select line from decoder to alu ( desel\_alu)

-- (decode\_mux1)

-- (decode\_mux2)

-- (rst\_sig)

-- (rxen\_sig)

-- (ryen\_sig)

-- (rdx\_rx)

-- (rdy\_ry)

-- (writen\_sig)

-- (wrd\_waddr)

-- (ry\_sel)

-- (rlsmux\_sel)

-- (mux3\_selectline)

-- offest (offset\_sig) inside branch

-- pcjump (pcjump\_sig) inside branch

---------------------signals for PC/BRANCH------------------------

-- (reg\_alu) regbank Rx output to pcbranch (rx\_in)

-- opcode is (deop\_alu)

signal offset\_sig: std\_logic\_vector(7 downto 0); --offset value from decoder to pcbranch

signal pcjump\_sig: std\_logic\_vector(7 downto 0); -- pc jump address from decoder to PCbranch

--clk\_sig

--iaddress (PC to imem)

begin

alu1: alu port map(reg\_alu,mux1\_alu,deop\_alu,desel\_alu,alu\_mux3);

mux1: mux port map(reg\_mux1,decode\_mux1,ry\_sel,mux1\_alu);

mux2: mux port map(decode\_mux2,reg\_mux1,rlsmux\_sel,mux2\_dmem);

mux3: mux port map(alu\_mux3,dmem\_mux3,mux3\_selectline,mux3\_reg);

mem1: datamem port map(reg\_alu,mux2\_dmem,deop\_alu,dmem\_mux3,clk\_sig);

reg1: regbank port map(clk\_sig,rst\_sig,writen\_sig,rxen\_sig,ryen\_sig,rdx\_rx,rdy\_ry,wrd\_waddr,mux3\_reg,reg\_alu,reg\_mux1);

decoder1: decoder port map(instr\_sig,deop\_alu,desel\_alu,decode\_mux1,decode\_mux2,rst\_sig,rxen\_sig,ryen\_sig,rdx\_rx,rdy\_ry,writen\_sig,wrd\_waddr,ry\_sel,rlsmux\_sel,mux3\_selectline,offset\_sig,pcjump\_sig);

imem1: imem port map(iaddress,clk\_sig,instr\_sig);

brnch: pcbranch port map(reg\_alu,deop\_alu,offset\_sig,pcjump\_sig,clk\_sig,iaddress);

end architecture top;