ECGR 3183 Project 6: Single-Cycle Processor

By: Christopher Lowe

5/3/2022

**Processor Code**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

use IEEE.std\_logic\_signed.all;

use work.all;

entity processor is

Port ( clk : in STD\_LOGIC);

end processor;

architecture Behavioral of processor is

component reg\_file is

Port ( readRegA : in STD\_LOGIC\_VECTOR (4 downto 0);

readRegB : in STD\_LOGIC\_VECTOR (4 downto 0);

writeReg : in STD\_LOGIC\_VECTOR (4 downto 0);

writeData : in STD\_LOGIC\_VECTOR (63 downto 0);

writeEnable : in STD\_LOGIC;

clk : in STD\_LOGIC;

readDataA : out STD\_LOGIC\_VECTOR (63 downto 0);

readDataB : out STD\_LOGIC\_VECTOR (63 downto 0));

end component;

for all : reg\_file use entity work.register\_file(Behavioral);

component main\_ALU is

Port ( alu\_input\_a : in STD\_LOGIC\_VECTOR (63 downto 0);

alu\_input\_b : in STD\_LOGIC\_VECTOR (63 downto 0);

alu\_control : in STD\_LOGIC\_VECTOR (3 downto 0);

alu\_result : inout STD\_LOGIC\_VECTOR (63 downto 0);

zero : out STD\_LOGIC);

end component;

for all : main\_ALU use entity work.ALU\_main(Behavioral);

component data\_mem is

Port ( mem\_in : in STD\_LOGIC\_VECTOR (63 downto 0);

mem\_address : in STD\_LOGIC\_VECTOR (63 downto 0);

mem\_write : in STD\_LOGIC;

mem\_read : in STD\_LOGIC;

mem\_out : out STD\_LOGIC\_VECTOR (63 downto 0));

end component;

for all : data\_mem use entity work.data\_memory(Behavioral);

component instr\_mem is

Port ( read\_addr : in STD\_LOGIC\_VECTOR (63 downto 0);

instruction : out STD\_LOGIC\_VECTOR (31 downto 0));

end component;

for all : instr\_mem use entity work.instruction\_memory(Behavioral);

component control is

Port ( opcode : in STD\_LOGIC\_VECTOR (10 downto 0);

Reg2Loc : inout STD\_LOGIC;

ALUSrc : out STD\_LOGIC;

MemtoReg : inout STD\_LOGIC;

RegWrite : out STD\_LOGIC;

MemRead : out STD\_LOGIC;

MemWrite : out STD\_LOGIC;

Branch : out STD\_LOGIC;

UncondBranch : out STD\_LOGIC;

ALUOp : out STD\_LOGIC\_VECTOR (1 downto 0));

end component;

for all : control use entity work.main\_control(Behavioral);

component pc is

Port ( pcIn : in STD\_LOGIC\_VECTOR (63 downto 0);

clk : in std\_logic;

pcOut : out STD\_LOGIC\_VECTOR (63 downto 0));

end component;

for all : pc use entity work.program\_counter(Behavioral);

component control\_ALU is

Port ( ALUOp : in STD\_LOGIC\_VECTOR (1 downto 0);

opcode : in STD\_LOGIC\_VECTOR (10 downto 0);

ALU\_control\_out : out STD\_LOGIC\_VECTOR (3 downto 0));

end component;

for all : control\_ALU use entity work.alu\_control(Behavioral);

component s\_ext is

Port ( s\_ext\_in : in STD\_LOGIC\_VECTOR (31 downto 0);

s\_ext\_out : out STD\_LOGIC\_VECTOR (63 downto 0));

end component;

for all : s\_ext use entity work.sign\_extend(Behavioral);

signal pIn : std\_logic\_vector (63 downto 0) := x"0000000000000000";

signal pOut : std\_logic\_vector (63 downto 0);

signal pClock : std\_logic;

signal instr\_mem\_in : std\_logic\_vector (63 downto 0);

signal instr\_mem\_out : std\_logic\_vector (31 downto 0);

signal regASel, regBSel, writeRegSel : std\_logic\_vector (4 downto 0);

signal regWriteEnable, regClock : std\_logic;

signal regIn, regAOut, regBOut : std\_logic\_vector (63 downto 0);

signal contIn : std\_logic\_vector (10 downto 0);

signal Reg2Loc, UncondBranch, Branch, MemRead, MemtoReg, MemWrite, ALUSrc, RegWrite : std\_logic;

signal ALUOp : std\_logic\_vector (1 downto 0);

signal contOp : std\_logic\_vector (10 downto 0);

signal ALUContID : std\_logic\_vector (1 downto 0);

signal AContOut : std\_logic\_vector (3 downto 0);

signal ALUInA, ALUInB, ALUOut : std\_logic\_vector (63 downto 0);

signal ALUOpcode : std\_logic\_vector (3 downto 0);

signal z : std\_logic;

signal data\_mem\_in, data\_mem\_addr, data\_mem\_out : std\_logic\_vector (63 downto 0);

signal data\_mem\_write, data\_mem\_read : std\_logic;

signal sIn : std\_logic\_vector (31 downto 0);

signal sOut : std\_logic\_vector (63 downto 0);

signal orOut : std\_logic := '0';

signal andOut : std\_logic;

begin

p : pc port map

(pcIn => pIn,

clk => pClock,

pcOut => pOut);

im : instr\_mem port map

(read\_addr => instr\_mem\_in,

instruction => instr\_mem\_out);

rf : reg\_file port map

(readRegA => regASel,

readRegB => regBSel,

writeReg => writeRegSel,

writeData => regIn,

writeEnable => regWriteEnable,

clk => regClock,

readDataA => regAOut,

readDataB => regBOut);

c : control port map

(opcode => contIn,

Reg2Loc => Reg2Loc,

ALUSrc => ALUSrc,

MemtoReg => MemtoReg,

RegWrite => RegWrite,

MemRead => MemRead,

MemWrite => MemWrite,

Branch => Branch,

UncondBranch => UncondBranch,

ALUOp => ALUOp);

ac : control\_ALU port map

(ALUOp => ALUContID,

opcode => contOp,

ALU\_control\_out => AContOut);

am : main\_ALU port map

(alu\_input\_a => ALUInA,

alu\_input\_b => ALUInB,

alu\_control => ALUOpcode,

alu\_result => ALUOut,

zero => z);

dm : data\_mem port map

(mem\_in => data\_mem\_in,

mem\_address => data\_mem\_addr,

mem\_write => data\_mem\_write,

mem\_read => data\_mem\_read,

mem\_out => data\_mem\_out);

se : s\_ext port map

(s\_ext\_in => sIn,

s\_ext\_out => sOut);

pClock <= clk;

pIn <= (pOut + x"0000000000000004") when orOut = '0' else (pOut + (sOut(61 downto 0) & "00"));

instr\_mem\_in <= pOut;

contIn <= instr\_mem\_out (31 downto 21);

ALUContID <= ALUOp;

contOp <= instr\_mem\_out (31 downto 21);

regClock <= clk;

regASel <= instr\_mem\_out (9 downto 5);

regBSel <= instr\_mem\_out (20 downto 16) when Reg2Loc = '0' else instr\_mem\_out (4 downto 0);

writeRegSel <= instr\_mem\_out (4 downto 0);

regWriteEnable <= RegWrite;

sIn <= instr\_mem\_out;

ALUInA <= regAOut;

ALUInB <= regBOut when ALUSrc = '0' else sOut;

ALUOpcode <= AContOut;

andOut <= z and Branch;

orOut <= UncondBranch or andOut;

data\_mem\_in <= regBOut;

data\_mem\_addr <= ALUOut;

data\_mem\_read <= MemRead;

data\_mem\_write <= MemWrite;

regIn <= ALUOut when MemtoReg = '0' else data\_mem\_out;

end Behavioral;

Graphical user interface

Description automatically generated with low confidence

**Figure 1: LDUR**

The Main Control, Sign Extend, and Data Memory function to perform the operation “LDUR X2, [X10, #0]”. The above image shows that the operation was performed successfully

Timeline

Description automatically generated

**Figure 2: STUR**

The Main Control, Sign Extend, and Data Memory function to perform the operation “STUR X4, [X7, #8]”. The above image shows that the operation was performed successfully

Graphical user interface

Description automatically generated

**Figure 3: SUB**

The Register File, ALU Control, and ALU function to perform the operation “SUB X4, X3, X2”. The above image shows that the operation was performed successfully

Timeline

Description automatically generated with medium confidence

**Figure 4: ADD**

The Register File, ALU Control, and ALU function to perform the operation “ADD X5, X3, X2”. The above image shows that the operation was performed successfully

Graphical user interface

Description automatically generated

**Figure 5: ORR**

The Register File, ALU Control, and ALU function to perform the operation “ORR X6, X2, X3”. The above image shows that the operation was performed successfully

A picture containing graphical user interface

Description automatically generated

**Figure 6: AND**

The Register File, ALU Control, and ALU function to perform the operation “AND X7, X2, X3”. The above image shows that the operation was performed successfully

Graphical user interface

Description automatically generated with medium confidence

**Figure 7: CBZ**

The Program Counter, Sign Extend, and Main Control function to perform the operation “CBZ X1, #2”. The above image shows that the operation was performed successfully

A picture containing timeline

Description automatically generated

**Figure 8: B**

The Program Counter, Sign Extend, and Main Control function to perform the operation “B #2”. The above image shows that the operation was performed successfully