Samuel Lichtenheld

10/1/2015

Assignment 2

Introduction.

The purpose of this assignment is to get hands on experience with an assembler and disassembler as well as start to make parts of the datapath that will be needed later on.

Setup.

I used C++ for the assembler/diassambler and VHDL and ModelSim for the remainder of the lab.

2.1.1.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Opcode | Rs | Rt | Rd | Shamt | funct |
| OR$4,$5,$2 | 000000 | 00101 | 00010 | 00100 | 00000 | 010101 |
| JR $3 | 00000 | 00011 | 00000 | 00000 | 00000 | 001000 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Opcode | Rs | Rt | Imm |
| BEQ $6, $7, loop | 000100 | 00110 | 00111 | 1111111111111110 |

2.1.2.

0000 0000 0000 00010

Same thing but unsigned

2.1.3.

You can branch 2^18 bytes away, so 2^16 instructions away. This is because there are 16 bits available in the immediate field and this value gets shifted left twice to give a farther range.

You can branch further away with a jump which gives you 25 bits of address space or even further away with jump register, which will jump to the address in a register.

2.2.1

2 separate labels. Thus, 21 instruction + 2 Labels = 23 lines

2.2.2

There are instructions that load only 1 byte, such as load byte. Therefore, it seems to be byte addressable.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | Mnemonic | Operation | Type | Opcode (hex) | Funct  (hex) | ALU action | ALU function code (bit) | Flags set (**Z**ero, **C**arry (unsigned overflow), **V** (signed overflow), **S**ign(+/-) |
| Add | add | R[rd] = R[rs] + R[rt] | R | 00 | 20 | add | 000 | CZSV |
| add imm. | addi | R[rt] = R[rs] + SignExtImm | I | 08 | - | add | 001 | CZSV |
| add imm. uns. | addiu | R[rt] = R[rs] + ZeroExtImm | I | 09 | - | add | 001 | CZSV |
| add uns. | addu | R[rd] = R[rs] + R[rt] | R | 00 | 21 | add | 000 | CZSV |
| and | and | R[rd] = R[rs] & R[rt] | R | 00 | 24 | and | 000 | SZ |
| and imm. | andi | R[rt] = R[rs] & ZeroExtImm | I | 0C | - | and | 011 | SZ |
| branch eq | beq | if (R[rs]==R[rt])      PC=PC+4+BranchAddr | I | 04 | - | subtract | 010 | CZSV |
| branch not eq | bne | if (R[rs]!=R[rt])      PC=PC+4+BranchAddr | I | 05 | - | subtract | 010 | CZSV |
| jump | j | PC=JumpAddr | J | 02 | - | add | 001 | CZSV |
| jump and link | jal | R[31]=PC+8;PC=JumpAddr | J | 03 | - | add | 001 | CZSV |
| jump reg | jr | PC=R[rs] | R | 00 | 08 | - | 000 |  |
| load byte uns. | lbu | R[rt] = {24’b0,M[R[rs]+SignExtImm](7:0)} | I | 24 | - | add | 001 | CZSV |
| load halfword uns. | lhu | R[rt] = {16’b0,M[R[rs]+SignExtImm](15:0)} | I | 25 | - | add | 001 | CZSV |
| load upper imm. | lui | R[rt] = {imm, 16’b0} | I | 0F | - | - |  |  |
| load word | lw | R[rt] = M[R[rs]+SignExtImm] | I | 23 | - | add | 001 | CZSV |
| Nor | nor | R[rd] = ~(R[rs] | R[rt]) | R | 00 | 27 | NOR | 000 | SZ |
| Or | or | R[rd] = R[rs] | R[rt] | R | 00 | 25 | OR | 000 | SZ |
| Or imm. | ori | R[rt] = R[rs] | ZeroExtImm | I | 0D | - | OR | 100 | SZ |
| set less than | slt | R[rd] = (R[rs]<R[rt]) ? 1:0 | R | 00 | 2A | slt | 000 | SZ |
| set less than imm. | slti | R[rt] = (R[rs]<SignExtImm) ? 1:0 | I | 0A | - | slt | 101 | SZ |
| set less than imm. uns. | sltiu | R[rt] = (R[rs]<SignExtImm) ? 1:0 | I | 0B | - | slt | 101 | SZ |
| set less than uns. | sltu | R[rd] = (R[rs]<R[rt]) ? 1:0 | R | 00 | 2B | slt | 000 | SZ |
| shift left logical | sll | R[rd] = R[rt] << shamt | R | 00 | 00 | sll | 000 | SZ |
| shift right logical | srl | R[rd] = R[rt] >>> shamt | R | 00 | 02 | srl | 000 | SZ |
| store byte | sb | M[R[rs]+SignExtImm}(7:0) = R[rt](7:0) | I | 28 | - | add | 001 |  |
| store halfword | sh | M[R[rs]+SignExtImm}(15:0) = R[rt](15:0) | I | 29 | - | add | 001 |  |
| store word | sw | M[R[rs]+SignExtImm} = R[rt] | I | 2B | -- | add | 001 |  |
| subtract | sub | R[rd] = R[rs] -R[rt] | R | 00 | 22 | subtract | 000 | CZSV |
| subtract uns. | subu | R[rd] = R[rs]= R[rt] | R | 00 | 23 | subtract | 000 | CZSV |

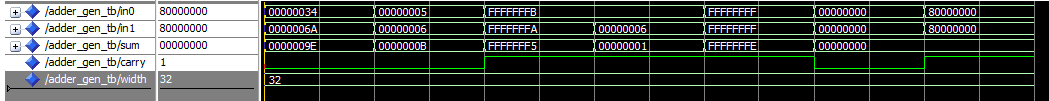


Figure 1: Adder TB

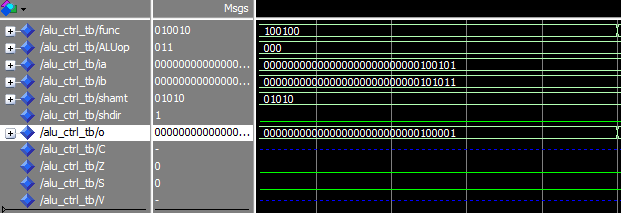


Figure 2: AND ALU + CTRL

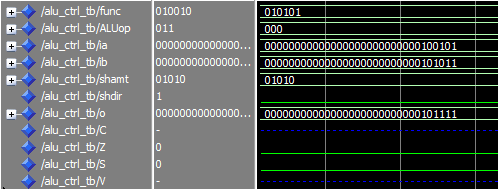


Figure 3: OR ALU + CTRL

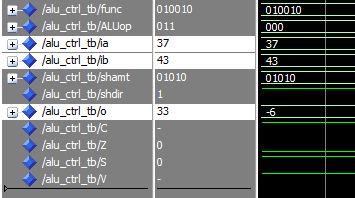


Figure 4: SUB ALU + CTRL

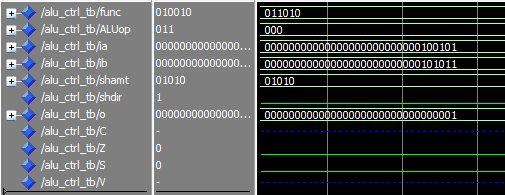


Figure 5: SLT ALU + CTRL

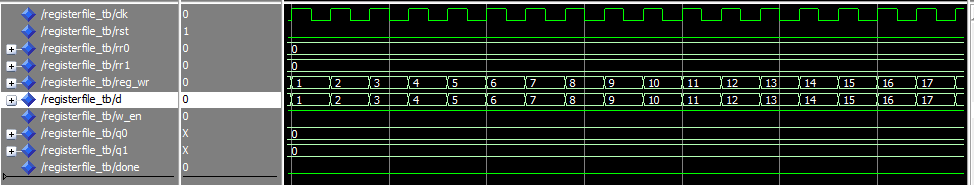


Figure 6: LOAD regfile

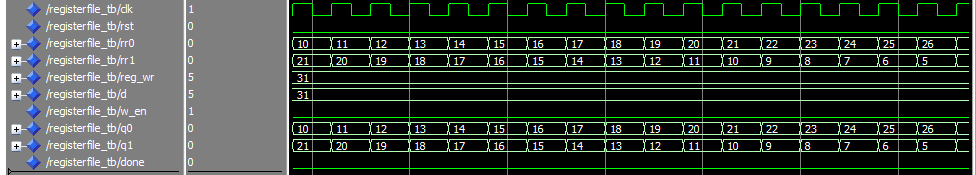


Figure 7: Read from regfile

# ADDER

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity adder\_gen is

generic (

width : positive := 32

);

port (

in0 : in std\_logic\_vector (width-1 downto 0);

in1 : in std\_logic\_vector (width-1 downto 0);

carry : out std\_logic;

sum : out std\_logic\_vector (width-1 downto 0)

);

end adder\_gen;

architecture BHV of adder\_gen is

begin

process(in0, in1)

variable temp : unsigned (width downto 0);

begin

temp := unsigned("0"&in0) + unsigned("0"&in1);

sum <= std\_logic\_vector(temp(width-1 downto 0));

carry <= std\_logic(temp(width));

end process;

end BHV;

# ADDER TB

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity adder\_gen\_tb is

end adder\_gen\_tb;

architecture TB of adder\_gen\_tb is

constant width : positive := 32;

signal in0, in1, sum : std\_logic\_vector(width-1 downto 0);

signal carry: std\_logic;

begin

UUT: entity work.adder\_gen

generic map (width => width)

port map (

in0 => in0,

in1 => in1,

carry => carry,

sum => sum

);

process

begin

-- test all input combinations

in0 <= std\_logic\_vector(to\_signed(52,width));

in1 <= std\_logic\_vector(to\_signed(106,width));

wait for 10 ns;

in0 <= std\_logic\_vector(to\_signed(5,width));

in1 <= std\_logic\_vector(to\_signed(6,width));

wait for 10 ns;

in0 <= std\_logic\_vector(to\_signed(-5,width));

in1 <= std\_logic\_vector(to\_signed(-6,width));

wait for 10 ns;

in0 <= std\_logic\_vector(to\_signed(-5,width));

in1 <= std\_logic\_vector(to\_signed(6,width));

wait for 10 ns;

in0 <= x"FFFFFFFF";

in1 <= x"FFFFFFFF";

wait for 10 ns;

in0 <= x"00000000";

in1 <= x"00000000";

wait for 10 ns;

in0 <= x"80000000";

in1 <= x"80000000";

wait for 10 ns;

wait;

report "Simulation Finished.";

wait;

end process;

end TB;

# ALU32

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity adder\_gen\_tb is

end adder\_gen\_tb;

architecture TB of adder\_gen\_tb is

constant width : positive := 32;

signal in0, in1, sum : std\_logic\_vector(width-1 downto 0);

signal carry: std\_logic;

begin

UUT: entity work.adder\_gen

generic map (width => width)

port map (

in0 => in0,

in1 => in1,

carry => carry,

sum => sum

);

process

begin

-- test all input combinations

in0 <= std\_logic\_vector(to\_signed(52,width));

in1 <= std\_logic\_vector(to\_signed(106,width));

wait for 10 ns;

in0 <= std\_logic\_vector(to\_signed(5,width));

in1 <= std\_logic\_vector(to\_signed(6,width));

wait for 10 ns;

in0 <= std\_logic\_vector(to\_signed(-5,width));

in1 <= std\_logic\_vector(to\_signed(-6,width));

wait for 10 ns;

in0 <= std\_logic\_vector(to\_signed(-5,width));

in1 <= std\_logic\_vector(to\_signed(6,width));

wait for 10 ns;

in0 <= x"FFFFFFFF";

in1 <= x"FFFFFFFF";

wait for 10 ns;

in0 <= x"00000000";

in1 <= x"00000000";

wait for 10 ns;

in0 <= x"80000000";

in1 <= x"80000000";

wait for 10 ns;

wait;

report "Simulation Finished.";

wait;

end process;

end TB;

# ALU32CONTROL

library ieee;

use ieee.std\_logic\_1164.all;

--ALUop

--R type 000

--add 001

--sub 010

--and 011

--or 100

--slt 101

entity alu32control is

port (

func : in std\_logic\_vector(5 downto 0);

ALUop : in std\_logic\_vector(2 downto 0);

control : out std\_logic\_vector(3 downto 0)

);

end alu32control;

architecture BHV of alu32control is

constant C\_ADD : std\_logic\_vector(3 downto 0) := "0010";

constant C\_SUB : std\_logic\_vector(3 downto 0) := "0110";

constant C\_AND : std\_logic\_vector(3 downto 0) := "0000";

constant C\_OR : std\_logic\_vector(3 downto 0) := "0001";

constant C\_NOR : std\_logic\_vector(3 downto 0) := "1100";

constant C\_SLT : std\_logic\_vector(3 downto 0) := "0111";

constant C\_SLTU : std\_logic\_vector(3 downto 0) := "1111";

constant C\_SHL : std\_logic\_vector(3 downto 0) := "0011";

constant rOp : std\_logic\_vector(2 downto 0) := "000";

constant addOp : std\_logic\_vector(2 downto 0) := "001";

constant subOp : std\_logic\_vector(2 downto 0) := "010";

constant andOp : std\_logic\_vector(2 downto 0) := "011";

constant orOp : std\_logic\_vector(2 downto 0) := "100";

constant sltOp : std\_logic\_vector(2 downto 0) := "101";

constant ADD\_F : std\_logic\_vector(5 downto 0) := "100000";

constant ADDU\_F : std\_logic\_vector(5 downto 0) := "100001";

constant AND\_F : std\_logic\_vector(5 downto 0) := "100100";

constant JR\_F : std\_logic\_vector(5 downto 0) := "001000";

constant NOR\_F : std\_logic\_vector(5 downto 0) := "010111";

constant OR\_F : std\_logic\_vector(5 downto 0) := "010101";

constant SLT\_F : std\_logic\_vector(5 downto 0) := "011010";

constant SLTU\_F : std\_logic\_vector(5 downto 0) := "011011";

constant SLL\_F : std\_logic\_vector(5 downto 0) := "000000";

constant SRL\_F : std\_logic\_vector(5 downto 0) := "000010";

constant SUB\_F : std\_logic\_vector(5 downto 0) := "010010";

constant SUBU\_F : std\_logic\_vector(5 downto 0) := "010011";

begin

process(func, ALUop)

begin

case ALUop is

when rOp =>

case func is

when ADD\_F =>

control <= C\_ADD;

when ADDU\_F =>

control <= C\_ADD;

when AND\_F =>

control <= C\_AND;

when JR\_F =>

control <= "----";

when NOR\_F =>

control <= C\_NOR;

when OR\_F =>

control <= C\_OR;

when SLT\_F =>

control <= C\_SLT;

when SLTU\_F =>

control <= C\_SLTU;

when SLL\_F =>

control <= C\_SHL;

when SRL\_F =>

control <= C\_SHL;

when SUB\_F =>

control <= C\_SUB;

when SUBU\_F =>

control <= C\_SUB;

when others =>

control <= "----";

end case;

when addOp =>

control <= C\_ADD;

when subOp =>

control <= C\_SUB;

when andOp =>

control <= C\_AND;

when orOp =>

control <= C\_OR;

when sltOp =>

control <= C\_SLT;

when others =>

control <= "----";

end case;

end process;

end BHV;

## ALU+CTRL

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity alu\_ctrl is

port(

func : in std\_logic\_vector(5 downto 0);

ALUop : in std\_logic\_vector(2 downto 0);

ia, ib : in std\_logic\_vector(31 downto 0);

shamt : in std\_logic\_vector(4 downto 0);

shdir : in std\_logic;

o : out std\_logic\_vector(31 downto 0);

C,Z,S,V : out std\_logic

);

end alu\_ctrl;

architecture STR of alu\_ctrl is

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

begin

U\_ALU32CONTROL : entity work.alu32control

port map (

func => func,

ALUop => ALUop,

control => ctrlSig

);

U\_ALU32: entity work.alu32

port map (

ia => ia,

ib => ib,

ctrl => ctrlSig,

shamt => shamt,

shdir => shdir,

o => o,

C => C,

Z => Z,

S => S,

V => V

);

end STR;

## ALU\_CTRL\_TB

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity alu\_ctrl\_tb is

end alu\_ctrl\_tb;

architecture TB of alu\_ctrl\_tb is

constant C\_ADD : std\_logic\_vector(3 downto 0) := "0010";

constant C\_SUB : std\_logic\_vector(3 downto 0) := "0110";

constant C\_AND : std\_logic\_vector(3 downto 0) := "0000";

constant C\_OR : std\_logic\_vector(3 downto 0) := "0001";

constant C\_NOR : std\_logic\_vector(3 downto 0) := "1100";

constant C\_SLT : std\_logic\_vector(3 downto 0) := "0111";

constant C\_SLTU : std\_logic\_vector(3 downto 0) := "1111";

constant C\_SHL : std\_logic\_vector(3 downto 0) := "0011";

constant rOp : std\_logic\_vector(2 downto 0) := "000";

constant addOp : std\_logic\_vector(2 downto 0) := "001";

constant subOp : std\_logic\_vector(2 downto 0) := "010";

constant andOp : std\_logic\_vector(2 downto 0) := "011";

constant orOp : std\_logic\_vector(2 downto 0) := "100";

constant sltOp : std\_logic\_vector(2 downto 0) := "101";

constant ADD\_F : std\_logic\_vector(5 downto 0) := "100000";

constant ADDU\_F : std\_logic\_vector(5 downto 0) := "100001";

constant AND\_F : std\_logic\_vector(5 downto 0) := "100100";

constant JR\_F : std\_logic\_vector(5 downto 0) := "001000";

constant NOR\_F : std\_logic\_vector(5 downto 0) := "010111";

constant OR\_F : std\_logic\_vector(5 downto 0) := "010101";

constant SLT\_F : std\_logic\_vector(5 downto 0) := "011010";

constant SLTU\_F : std\_logic\_vector(5 downto 0) := "011011";

constant SLL\_F : std\_logic\_vector(5 downto 0) := "000000";

constant SRL\_F : std\_logic\_vector(5 downto 0) := "000010";

constant SUB\_F : std\_logic\_vector(5 downto 0) := "010010";

constant SUBU\_F : std\_logic\_vector(5 downto 0) := "010011";

signal func : std\_logic\_vector(5 downto 0);

signal ALUop : std\_logic\_vector(2 downto 0);

signal ia, ib : std\_logic\_vector(31 downto 0);

signal shamt : std\_logic\_vector(4 downto 0);

signal shdir : std\_logic;

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

signal C,Z,S,V : std\_logic;

begin

UUT: entity work.alu\_ctrl

port map(

func => func,

ALUop => ALUop,

ia => ia,

ib => ib,

shamt => shamt,

shdir => shdir,

o => o,

C => C,

Z => Z,

S => S,

V => V

);

process

begin

func <= ADD\_F;

ALUop <= rOp;

ia <= std\_logic\_vector(to\_unsigned(37,32));

ib <= std\_logic\_vector(to\_unsigned(43,32));

shamt <= std\_logic\_vector(to\_unsigned(10,5));

shdir <= '0';

wait for 10 ns;

func <= AND\_F;

wait for 10 ns;

func <= OR\_F;

wait for 10 ns;

func <= SLT\_F;

wait for 10 ns;

func <= SLL\_F;

wait for 10 ns;

shdir <= '1';

wait for 10 ns;

func <= SUB\_F;

wait for 10 ns;

ALUop <= andOp;

wait for 10 ns;

wait;

end process;

end TB;

**REGISTERFILE**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity registerFile is

port (

clk, rst : in std\_logic;

rr0 : in std\_logic\_vector(4 downto 0); -- register file reads

rr1 : in std\_logic\_vector(4 downto 0);

reg\_wr : in std\_logic\_vector(4 downto 0); --which reg should be written to on rising edge of clk

d : in std\_logic\_vector(31 downto 0); -- input word

w\_en : in std\_logic; -- write register enable

q0, q1 : out std\_logic\_vector(31 downto 0)

);

end registerFile;

architecture STR of registerFile is

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

type rr\_array is array(31 downto 0) of std\_logic\_vector(31 downto 0);

signal q\_array: rr\_array;

--type registerarray is array(X downto 0) of std\_logic\_vector(X downto 0);

--signal ra: registerarray;

--The first line defines new a type that is an array of std\_logic\_vector's; the second line initializes a signal of type “registerarray” with name “ra”. The defined signal can be used in following manner “ra(0) <= 0xDEADBEE7”.

begin

U\_REG\_ARRAY: for i in 0 to 31 generate

U\_REG : entity work.reg\_gen

generic map(32)

port map(

clk => clk,

rst => rst,

ld => ldi(i),

d => d,

q => q\_array(i)

);

end generate U\_REG\_ARRAY;

process(clk,rst,rr0,rr1,reg\_wr,d,w\_en)

variable temp: unsigned (31 downto 0);

begin

q0 <= q\_array(to\_integer(unsigned(rr0)));

q1 <= q\_array(to\_integer(unsigned(rr1)));

temp := x"00000000";

if (w\_en = '1') then

temp(to\_integer(unsigned(reg\_wr))) := '1';

end if;

ldi <= std\_logic\_vector(temp);

end process;

end STR;

**REGISTERFILE TB**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity registerFile\_tb is

end registerFile\_tb;

architecture TB of registerFile\_tb is

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

signal rst : std\_logic;

signal rr0 : std\_logic\_vector(4 downto 0);

signal rr1 : std\_logic\_vector(4 downto 0);

signal reg\_wr : std\_logic\_vector(4 downto 0);

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

signal w\_en : std\_logic;

signal q0 : std\_logic\_vector(31 downto 0);

signal q1 : std\_logic\_vector(31 downto 0);

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

begin

clk <= not clk and not done after 10 ns;

UUT: entity work.registerFile

port map(

clk => clk,

rst => rst,

rr0 => rr0,

rr1 => rr1,

reg\_wr => reg\_wr,

d => d,

w\_en => w\_en,

q0 => q0,

q1 => q1

);

process

begin

rst <= '0';

rr0 <= std\_logic\_vector(to\_unsigned(0,5));

rr1 <= std\_logic\_vector(to\_unsigned(0,5));

reg\_wr <= std\_logic\_vector(to\_unsigned(0,5));

d <= std\_logic\_vector(to\_unsigned(0,32));

w\_en <= '0';

wait for 1 ns;

rst <= '1';

wait for 19 ns;

rst <= '0';

wait for 10 ns;

for i in 0 to 31 loop

w\_en <= '1';

d <= std\_logic\_vector(to\_unsigned(i,32));

reg\_wr <= std\_logic\_vector(to\_unsigned(i,5));

wait until rising\_edge(clk);

end loop;

w\_en <= '0';

wait until rising\_edge(clk);

for i in 0 to 31 loop

rr0 <= std\_logic\_vector(to\_unsigned(i,5));

rr1 <= std\_logic\_vector(to\_unsigned(31-i,5));

wait until rising\_edge(clk);

end loop;

done <='1';

wait;

end process;

end TB;