Bob Minchin

EEL4713C

Assignment 3

February 13, 2012

#### INTRODUCTION

The purposes of this assignment were (1) to become further acquainted with the MIPS architecture by going through a bug-finding exercise in processor simulation and (2) to reinforce some of the basic principles covered in class by guiding the student through designing some more basic components that will be needed to build into a microprocessor pipeline later in the semester. A basic adder, a MIPS ALU, an ALU controller, and a register file were assigned, and each was designed and simulated according to the directions.

## MIPS BUG SIMULATION ("BUGSPIM")

The purpose of this exercise was to identify five bugs in a MIPS simulation program based on knowledge of how the MIPS architecture operates. Although the directions seemed straightforward, several problems were encountered when trying to complete this part of the assignment. First was my difficulty getting Bugspim to start without the instruction to load it directly into the SPIM directory. Then there was my complete lack of familiarity with how to work within Grid Appliance/Linux. I had to figure out everything from how to edit text to how the file path structure worked. Finally, it took me a long time to figure out that the same lines of code are executed at the beginning of every program and that my attempts to load custom code into the simulator were actually successful.

After overcoming all of these problems, I did not have much time left to actually perform the simulation. In order to determine which instructions created problems, I ran the following instructions numerous times in various different contexts: ADD, ADDI, ADDU, ADDUI, AND, ANDI, BNE, DIV, DIVU, J, JR, LB, LUI, LW, MFHI, MFLO, MULT, MULTU, OR, ORI, SB, SLL, SLT, SLTI, SLTIU, SLTU, SRL, SUB, SUBU, SW, XOR, and XORI. However, I was able to locate only one bug, which was as follows.

The instruction "load upper immediate" (lui) does not work. It is designed to take in a 16-bit number and place it in the upper half of the register. The following code was executed:

### lui \$t0, 65535

This should have resulted in t0 = 0xFFFF0000. Instead, it resulted in t0 = 0x0000FFFF, which means that the load was accomplished but the shift was not.

### DESIGN AND TESTING

The first device to design was a simple 32-bit adder that was named "add32." It was designed by simply taking two signals and applying an arithmetic add operation, which was accomplished using the following code.

```
-- Bob Minchin

-- EEL 4713C

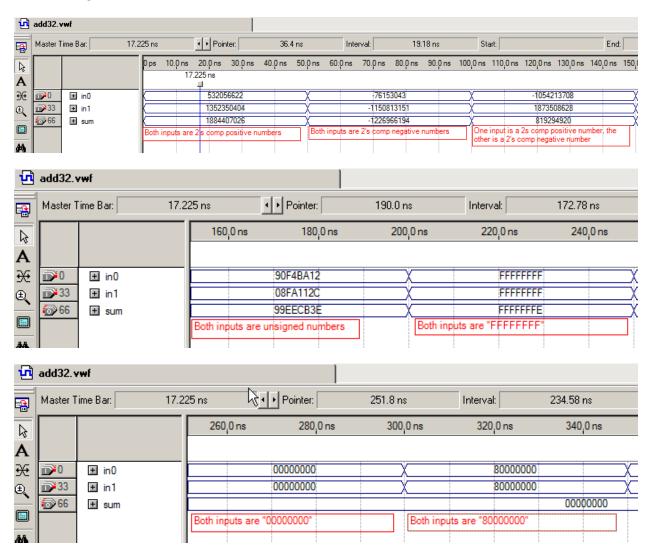
-- Homework 3

-- Part 3.1: 32-Bit Adder

LIBRARY ieee ;
```

END Structure ;

The following are screenshots of the simulation of this device.



The device performed as expected.

The next device was a 32-bit ALU, which eventually will be at the heart of all MIPS operations. The purpose of this device is to perform basic arithmetic and logic operations on numbers in internal registers that will be used to execute more complex operations in the computer architecture.

The code used for the overflow flag was quick and dirty—sort of a "brute force" solution that requires an additional add every time an add or subtract operation is requested. This is an inefficient design that works for our current purposes but will likely need to be replaced by another solution, such as a ripplecarry adder, once timing and efficiency become important.

Furthermore, the ALU was implemented as an "all-in-one" device. No modularity was implemented. All functions were performed internally. As the device is refined, more refined methods of implementation may be built in to help facilitate greater efficiency.

This ALU, "alu32," was accomplished with the following code.

```
-- Bob Minchin
-- EEL 4713C
-- Homework 3
-- Part 3.2: 32-Bit ALU
LIBRARY ieee ;
USE ieee.std logic 1164.all ;
USE ieee.std_logic_arith.all ;
USE ieee.std_logic_unsigned.all ;
ENTITY alu32 IS
     --32-bit inputs, output
     --4-bit control input
     --5-bit ALUOp input, shift amount input
     --1-bit shift direction input, output flags
                           IN STD_LOGIC_VECTOR(31 DOWNTO 0) ;
IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
     PORT (
                ia, ib :
                control :
                shamt :
                                IN STD_LOGIC_VECTOR(4 DOWNTO 0) ;
                shdir :
                                IN STD_LOGIC ;
                o :
                                BUFFER STD_LOGIC_VECTOR(31 DOWNTO 0) ;
                C, Z, S, V : OUT STD_LOGIC ) ;
END alu32 ;
ARCHITECTURE Structure OF alu32 IS
     SIGNAL temp33 : STD_LOGIC_VECTOR(32 DOWNTO 0) ;
BEGIN
     PROCESS(ia, ib, control, shamt, shdir, temp33, o)
           VARIABLE temp32 : unsigned(31 DOWNTO 0) ;
     BEGIN
           C <= '0' ; --initialize flags
           V <= '0';
           IF o = x"0000000" THEN
                Z <= '1' ;
```

```
ELSE Z <= '0' ;
           END IF ;
           IF \circ (31) = '1' THEN
                S <= '1' ;
           ELSE S <= '0' ;
           END IF ;
           CASE control IS
                WHEN "0000" => --and
                      o <= ia AND ib ;
                 WHEN "0001" => --or
                      o <= ia OR ib ;
                 WHEN "0010" => --add
                      temp33 <= ('0' & ia) + ('0' & ib) ;
                      temp32 := UNSIGNED(('0' & ia(30 DOWNTO 0))) +
                            UNSIGNED(('0' & ib(30 DOWNTO 0)));
                      V <= temp33(32) XOR temp32(31) ;
                      C <= temp33(32) ;
                      o <= temp33(31 DOWNTO 0) ;</pre>
                 WHEN "0011" => --shift logical
                      IF (shdir = '0') THEN
                            temp32 := SHL(UNSIGNED(ib),
                                 UNSIGNED(shamt)) ;
                      ELSE
                            temp32 := SHR(UNSIGNED(ib),
                                 UNSIGNED(shamt)) ;
                      END IF ;
                      o <= CONV_STD_LOGIC_VECTOR(temp32, 32) ;</pre>
                 WHEN "0110" => --subtract
                      temp33 <= ('0' & ia) + (NOT ('1' & ib) ) + 1;
                      temp32 := UNSIGNED(('0' & ia(30 DOWNTO 0))) +
                            UNSIGNED(('0' & ib(30 DOWNTO 0)));
                      V <= temp33(32) XOR temp32(31) ;
                      C <= temp33(32) ;
                      o <= temp33(31 DOWNTO 0) ;</pre>
                 WHEN "0111" => --set less than
                      IF (SIGNED(ia) < SIGNED(ib) ) THEN
                            o <= x"0000001" ;</pre>
                      ELSE o <= x"00000000" ;
                      END IF ;
                 WHEN "1100" => --nor
                      o <= NOT (ia OR ib) ;</pre>
                 WHEN "1111" => --set less than unsigned
                      IF (UNSIGNED(ia) < UNSIGNED(ib) ) THEN
                            o <= x"0000001" ;</pre>
                      ELSE o <= x"00000000" ;
                      END IF ;
                 WHEN OTHERS =>
           END CASE ;
     END PROCESS ;
END Structure ;
```

Master Time Bar:	17.225 ns	•	Pointe	er:	5.2 r	ns	Interva	al:	-12.03	) ns	Start:				End:
	0 ps			30.0 ns	40.0 ns	50.0 ns	60.0 ns	70.0 ns	80.0 ns	90.0 ns	100,0 ns	110,0 ns	120 <sub>,</sub> 0 ns	130,0 ns	140.0 ns
		17.22													
		-	000	0		X_		00	01		X.		00	10	
■>5 shdir															
mile shamt			0111					11(					111		
12		0111111000				—X—	11001100				<b>_</b> ↓			010011110	
		0110111000				=;⊱	11001110				;			01011010	
I11 V															
•@>113 C															
•@>114 S	Whe	en "control" =	0000 the	e ALU per	rforms	14/	en "control"	- 0001 +	he Alline	rforme	Mba	n "control	" - 0010 H	he ALU pe	forme
		gical AND.	0000, 110		lonna	al	ogical OR. I	Vote that	the sign fla			rithmetic s		ne ALO per	noms
						en	abled becau	se the MS	oB = 1.						
			i	i	i				i	1				i	i
alu32.vhd				🖸 a	alu32. vwf										
Master Time Bar:	17.225 ns	• • P	ointer:		49.54 ns	1	nterval:		.32 ns	Star	t			End:	
	150.0 ns	S		1	60 <sub>,</sub> 0 ns			170	0 ns			180,0 r	าร		
iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii	D1X								0011						
■>5 shdir									0011			_			
→6   shamt	10	01	1011		X	(	)1010			110				11111	
12									101110101						
								01010111							
	10X 10X0000	001101011100	10000100	00000000	00X100000	11010111		01010111				000000	000000000000000000000000000000000000000	000000000000000000000000000000000000000	000000000
	10X 10X0000	001101011100	10000100	00000000	00 <u>×100000</u>	11010111						000000	000000000	000000000000000000000000000000000000000	00000000
<ul> <li> <sup>6</sup> 78         <ul> <li></li></ul></li></ul>	10,0000	001101011100	10000100	00000000	<u>00×100000</u>	11010111							000000000	000000000	00000000
<ul> <li>              √78             ●             0      </li> <li>             111             V         </li> </ul>		001101011100	10000100	00000000	00 <u>×100000</u>	11010111							000000000	000000000	00000000
<ul> <li>2078</li> <li>20111</li> <li>2</li> <li>20112</li> <li>2</li> <li>20113</li> <li>C</li> </ul>		001101011100 en "control" = 0					00100001000	0000000	(01000010	00000000	000000000			000000000	00000000
<ul> <li>2078</li> <li>20111</li> <li>20112</li> <li>20113</li> <li>20113</li> </ul>							00100001000	0000000	(01000010	00000000	000000000	n in "shamt		Z flag pope	
<ul> <li> <sup>™</sup> <sup>™   <sup>™</sup></sup></li></ul>							in the directi	on given i	(01000010 n "shdir" ar	00000000	000000000	n in "shamt			
€278							in the directi	0000000	(01000010 n "shdir" ar	00000000	000000000	n in "shamt			
€ 78 ■ 111 V 112 C 113 C 114 S			0011, the d	device pe		gical shift	in the directi	on given i	(01000010) n "shdir" ar wf	nd by the a	000000000	n in "shamt		Z flag pops	
		en "control" = 0 17.225 ns	0011, the d	device pe	rforms a lo	gical shift	in the directi	on given i lu <b>32. v</b> 32.19 na	(01000010) n "shdir" ar wf	nd by the a	mount giver	n in "shamt Nu re	ote that the sult is 0.	Z flag pops	s up when t
		en "control" = 0 17.225 ns	0011, the (	device pe	rforms a lo	gical shift	in the directi	on given i lu <b>32. v</b> 32.19 na	(01000010) n "shdir" ar wf	nd by the a	mount giver	n in "shamt Nu re	tote that the sult is 0.	Z flag pops	s up when t
		en "control" = 0 17.225 ns	0011, the (	device pe	rforms a lo	gical shift ter:	in the directi	on given i lu <b>32. v</b> 32.19 na	(01000010) n "shdir" ar wf	nd by the a	mount giver	n in "shamt Nr re	tote that the sult is 0.	Z flag pops	s up when t
	control	en "control" = 0 17.225 ns	0011, the (	device pe	rforms a lo	gical shift	in the directi	on given i lu <b>32. v</b> 32.19 na	(01000010) n "shdir" ar wf	nd by the a	mount giver	n in "shamt Nu re	tote that the sult is 0.	Z flag pops	s up when t
	control	en "control" = 0 17.225 ns	0011, the (	device pe	Point 320,0 ns 01	gical shift ter:	in the directi	on given i lu <b>32. v</b> 32.19 na	(01000010) n "shdir" ar wf	nd by the a	mount giver	n in "shamt Ne re 38 0111	te that the sult is 0. 364.97	Z flag pops	s up when t
	control	en "control" = 0 17.225 ns	0011, the .0 ns	device pe	► Point 320,0 ns 01 00	gical shift ter:	in the direction of the	on given i lu32. v 32.19 ns	(01000010) n "shdir" ar wf	ad by the a	erval:	100111	364.97	Z flag pope	s up when t
	control	en "control" = 0 17.225 ns	0011, the .0 ns	device pe	► Point 320,0 ns 01 00	gical shift ter:	in the directi	on given i lu32. v 32.19 ns	(01000010) n "shdir" ar wf	ad by the a	mount giver	100111	364.97	Z flag pope	s up when t
	control hdir hamt	en "control" = 0 17.225 ns	0011, the o	device pe	Point 320,0 ns 01 00 1010111	gical shift ter:	in the direction of the	on given i lu 32. v 32.19 ns 0 ns	(01000010) n "shdir" ar wf	ad by the a	erval:	01111 10011 0110101	364.97 30,0 ns	Z flag pops ns	s up when t s up when t s 400,0
	control hdir hamt a	en "control" = 0 17.225 ns	0011, the o	device pe 3 00011 10110	Forms a lo	gical shift ter:	in the direction of the	on given i lu32. v 32.19 ns 0 ns 0000 100	(01000010) n "shdir" ar wf	ad by the a	erval:	0111 01100101 01100101	364.97 30,0 ns 0,010011 0010011	Z flag pops ns 110000 <sup>-1</sup>	s up when t s up when t s 400,0
	control hdir hamt	en "control" = 0 17.225 ns	0011, the o	device pe 3 00011 10110	Forms a lo	gical shift ter:	in the direction in the direction 340, 010011111 11110110	on given i lu32. v 32.19 ns 0 ns 0000 100	(01000010) n "shdir" ar wf	ad by the a	erval:	0111 01100101 01100101	364.97 30,0 ns 0,010011 0010011	Z flag pops ns 110000 <sup>-1</sup>	s up when t s up when t s 400,0
	control hdir hamt b	en "control" = 0 17.225 ns	0011, the o	device pe 3 00011 10110	Forms a lo	gical shift ter:	in the direction in the direction 340, 010011111 11110110	on given i lu32. v 32.19 ns 0 ns 0000 100	(01000010) n "shdir" ar wf	ad by the a	erval:	0111 01100101 01100101	364.97 30,0 ns 0,010011 0010011	Z flag pops ns 110000 <sup>-1</sup>	s up when t s up when t s 400,0
<ul> <li>         778         90         111         V         111         V         111         C         111         C         111         C         111         S         </li> <li>         alu32.vhd     </li> <li>         Master Time Bar:     </li> <li>         Master Time Bar:     </li> <li>         Master Time Bar:     </li> <li>         12         till     </li> <li>         5         s     </li> <li>         12         till         <ptillit< p=""> <ptill< p=""> <ptill< p=""></ptill<></ptill<></ptillit<></li></ul>	control hdir hamt a	en "control" = 0 17.225 ns	0011, the o	device pe 3 00011 10110	Forms a lo	gical shift ter:	in the direction in the direction 340, 010011111 11110110	on given i lu32. v 32.19 ns 0 ns 0000 100	(01000010) n "shdir" ar wf	ad by the a	erval:	0111 01100101 01100101	364.97 30,0 ns 0,010011 0010011	Z flag pops ns 110000 <sup>-1</sup>	s up when t s up when t s 400,0
<sup>2</sup> 78 <sup>2</sup> 7 <sup>2</sup> 711 <sup>2</sup> 7 <sup>2</sup>	control hdir hamt	en "control" = 0 17.225 ns	0011, the o	device pe 3 00011 10110	Forms a lo	gical shift ter:	in the direction in the direction 340, 010011111 11110110	on given i lu32. v 32.19 ns 0 ns 0000 100	(01000010) n "shdir" ar wf	ad by the a	erval:	0111 01100101 01100101	364.97 30,0 ns 0,010011 0010011	Z flag pops ns 110000 <sup>-1</sup>	s up when t s up when t s 400,0
	control hdir hamt	en "control" = ( 17.225 ns 300.	0011, the o	device pe	Point 320,0 ns 01 01010111 01111100 0011011	gical shift ter: 10 110 111111( 01000) 10110(	in the direction in the direc	on given i lu32. v 32.19 ns 0 ns 0000 100	oriococito	ad by the a	erval: 10111110 10000000000000000000000000000	0111 0110010 0000000	364.97 30,0 ns 0,0011 0000000	Z flag pops ns 110000 <sup>-1</sup> 1111000	s up when t
<sup>2</sup> 78 <sup>2</sup> 7 <sup>2</sup> 711 <sup>2</sup> 7 <sup>2</sup>	control hdir hamt	en "control" = ( 17.225 ns 300.	0011, the 0	device pe	Point 320,0 ns 01 01010111 01111100 0011011	gical shift ter: 10 110 1111110 101000 101100 101100 the dev	in the direction in the direction 340, 010011111 11110110	on given i lu32. v 32.19 ns 0 ns 0000 100	oriococito n "shdir" ar wf s V V V V V V V V V V V V V	ad by the a Inte 360,0 r 100111 01000000	erval:	0111 0000000 0000000	364.97 364.97 30,0 ns 0,00011 0010011 0000000 0000000	Z flag pops ns 110000 1111000 000000 erforms	s up when t

# The following are screenshots of the simulation that illustrate each of the ALU operations.

\$	alu32.vhd				ับปี alu32.vwf			
<b>.</b>	Master Tin	ne Bar:	17.225 ns	<ul> <li>Image: Pointer:</li> </ul>	742.79 ns		I	
R			600.0 ns	620,0 ns	640,0 ns	- 11Ý	1111	ý—
A						- P^		^
Æ	<b>i</b>		D1X	1100	X		00010	¥=
Ð	■>5	shdir				— 蔽	00010010100010010000000011110011	Ŷ
	<b>6</b>	🛨 shamt		01000	X		00100000011100001010101111101010	ŷ=
	12 → 45	ia ∃ib		010001000000111110110 0000100010011110110		= 10	000000000000000000000000000000000000000	Ŷ
<b>#4</b>	78	E D E o		10110011011000000000		=[~`		<u> </u>
<b>≜</b> ∦B	111	V						
$\langle \overline{\sigma}  $	112	Z						
×.	113	С						
<u>.</u>	114	S				Wh	en "control" = 1111 the device performs an	
1			When " NOR or	control" = 1100, the devic peration.	e performs a logical	luns	igned set-on-less than operation. Because "ia" i s than "ib," "o" is set to 1.	s
z								

The device performed as expected.

The next part was a controller for this ALU. The job of the controller is to decode the instruction operand into a set of instructions for the ALU. Because there are a number of instructions that require the ALU to perform the same operation, this device simply needs to convert the relevant parts of the machine code into an internal signal that will tell the ALU what it needs to do. Basically, it is a glorified look-up table.

The first step was to create a physical look-up table that included all the MIPS instructions and their corresponding ALU operations. Once the instructions and operations were delineated, three-bit ALU operation codes were assigned to each instruction. If multiple instructions needed the same function performed in the ALU, they were given the same ALU operation code.

Instruction Opcode	ALUop	Instruction Operand	Funct Field	Desired ALU Action	ALU Control Input
R-type	100	add	100000	add	0010
001000	000	add imm		add	0010
001001	000	add imm uns		add	0010
R-type	100	add uns	100001	add	0010
R-type	100	and	100100	and	0000
001100	011	and imm		and	0000
000100	001	branch on equal		subtract	0110
000101	001	branch on not equal		subtract	0110
000010		jump			
000011		jump and link			
R-type	100	jump reg	001000		
100100	000	load byte uns		add	0010

The table that was created can be seen below.

100101	000	load hw uns		add	0010
110000	000	load linked		add	0010
001111	000	load upper imm		add	0010
100011	000	load word		add	0010
R-type	100	nor	100111	nor	1100
R-type	100	or	100101	or	0001
001101	010	or imm		or	0001
R-type	100	set less than	101010	set less than	0111
001010	101	set less than imm		set less than	0111
001011	110	set less than imm uns		set less than uns	1111
R-type	100	set less than uns	101011	set less than uns	1111
R-type	100	shift left log	000000	shift log	0011
R-type	100	shift right log	000010	shift log	0011
101000	000	store byte		add	0010
111000	000	store cond		add	0010
101001	000	store hw		add	0010
101011	000	store word		add	0010
R-type	100	subtract	100010	subtract	0110
R-type	100	subtract uns	100011	subtract	0110

This table was referenced constantly in creating the code for the entity, which was named "alu32control." In all cases except for that of "R-type" instructions, the ALU opcode was all that was needed to determine the ALU control signal. For "R-type" instructions, the instruction opcode was needed as well.

The following code was used to implement "alu32contol."

```
-- Bob Minchin
-- EEL 4713C
-- Homework 3
-- Part 3.3: 32-Bit ALU Controller
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY alu32control IS
      --6-bit instruction opcode input
      --5-bit ALU opcode input
      --4-bit ALU control signal output
                 func : IN STD_LOGIC_VECTOR(5 DOWNTO 0) ;
ALUop : IN STD_LOGIC_VECTOR(2 DOWNTO 0) ;
      PORT (
                  control : OUT STD_LOGIC_VECTOR(3 DOWNTO 0) );
END alu32control ;
ARCHITECTURE Structure OF alu32control IS
BEGIN
```

```
PROCESS(func, ALUop)
     BEGIN
           CASE ALUOP IS
                 WHEN "100" => --r-type
                       CASE func IS
                             WHEN "100000" => --add
                                   control <= "0010" ;
                             WHEN "100001" => --add unsigned
                                   control <= "0010" ;</pre>
                             WHEN "100100" => --and
                                   control <= "0000" ;
                             WHEN "100111" => --nor
                                   control <= "1100" ;
                             WHEN "100101" => --or
                                   control <= "0001" ;</pre>
                             WHEN "101010" =>--set less than
                                   control <= "0111" ;</pre>
                             WHEN "101011" => --set less than unsigned
                                   control <= "1111" ;</pre>
                             WHEN "000000" => -- shift left logical
                                   control <= "0011" ;</pre>
                             WHEN "000010" => --shift right logical
                                   control <= "0011" ;</pre>
                             WHEN "100010" => --subtract
                                   control <= "0110" ;</pre>
                             WHEN "100011" => --subtract unsigned
                                   control <= "0110" ;</pre>
                             WHEN OTHERS =>
                       END CASE ;
                 WHEN "000" => --add immediate
                       control <= "0010" ;</pre>
                 WHEN "011" => --and immediate
                       control <= "0000" ;
                 WHEN "001" => --branch
                       control <= "0110" ;</pre>
                 WHEN "010" => --or immediate
                       control <= "0001" ;</pre>
                 WHEN "101" => --set less than
                       control <= "0111" ;</pre>
                 WHEN "110" => --set less than unsigned
                       control <= "1111" ;</pre>
                 WHEN OTHERS =>
           END CASE ;
     END PROCESS ;
END Structure ;
```

The simulation waveforms can be seen as follows.

alu32control.vwf			🕹 alu	J32control.vhd				
Master Time Bar:	17.525 ns	• • Pointe	er: 17	74.7 ns	Interval:	157.18 ns	9	Start:
A	0 ps 17.5	40.0 ns 25 ns	80.0 ns	120,0 ns	160,0 ns	200,0 ns	240,0 ns	
			001 X 0110 X op = 001 rs to a ract.	010 0001 ALUop = 010 refers to an or.	011 000000 		101 0111 = 101 to an slt	X 110 ALUop = 110 refers to an sltu.
A.2 II I								
▲ 2 II IIII IIIIIIIIIIIIIIIIIIIIIIIIIII			💱 alu32control.v					
_	17.525 ns 350.0 ns 1X 1000	Pointer: 390,0 ns 430,0	787.48 ns	Interval:	769.96 ns	590,0 ns	630,0 ns	670,0 ns

The device performed as expected.

The final device for this project was a register file called "registerFile.". The register file contains all 32 internal registers for the MIPS microprocessor. Each register in the register file is multiplexed at the input and the output to determine which muxes are written to and/or read from on any given operation. Two registers can be read from at once, while only one register can be written to. This allows register-register operations, where both inputs come simultaneously from separate registers, and the output is written to a third. Because the zero register cannot be written to, no control logic was created for its modification. Any attempt to write over its default value of zero will be ignored.

The device was implemented using the 32-bit register module, "reg32," created in Assignment 2. The device was multiplexed internally, and the control logic for the write enables also was handled internally. Thus, the only modular component used was "reg32." This was not an inefficient implementation in terms of architecture, but whether the coding would have gone faster had I used a 2-D array is subject to debate. But I'll leave that one to the philosophers.

One other thing worth noting is that in its current form, the device performs on a purely synchronous basis. In future use, the rising-clock synchronization of the register file outputs will probably be replaced by an asynchronous realization that changes the output as soon as the read register control input changes.

Due to its length, the code for "registerFile" can be found in Appendix B. The simulation waveforms are as follows.

æ	registerFile.	vhd				🖸 registe	rFile.vwf					
	Master Tim	e Bar:	17.225 ns	• • P	ointer:	260.89 r	IS	Interval:	243.67 ns	Star	rt:	End:
R			0 ps	20.0 ns	40.0 ns	60.0 ns	80.0 ns	100,0	0 ns 120,0 ns	140,0 ns	160 <sub>,</sub> 0 ns 180	.0 ns 200,0 ns
Α			1	7.225 ns								
Æ	∎>0	clk										
Ð,	<b>™</b> 1	± m0			1	2		X	0	X	12	
_	<b>₽</b> 7	± m1			1	9		X	0	X	19	
	13	+ rw					12			X	0	
<b>#</b> \$	■> 19	WF										
¢.₀	20	± d			4F9/	C335		X	F6904AB/	۸ X	000000	10 X
	53	± q0					000	00000			X	F6904ABA
XΨ	86	⊞ q1								0000		
XX 라			is acco		0" and "q1" r	write enable eflect the defa ely.		ate	The write enable goe which allows F6904, written to register 12 Meanwhile, register read on both outputs	ABA tobe 0 is being	Registers 12 and 1 again read, but this register 12 contains that was written to clock cycle.	time s the number

۰	registerFile	registerFile.vhd								
	Master Tir	ne Bar: 17.2	25 ns •• Pointer:			459.6 ns	Interval:	44	12.38	
<b>₽</b>			210,0 ns	230,0 ns	250,0 ns	270 <sub>,</sub> 0 ns	290 <sub>,</sub> 0 ns	310 <sub>,</sub> 0 ns	3	
Α										
Æ	▶0	clk								
€	<b>₽</b> 1	0m \pm		0	X	19	X			
	₫7	± π1		0	X	12	X			
	<b>₽</b> 13	± rw		19	X					
ĝĝ.	■> 19	wr								
<b>≜</b> ,₀	<b>₽</b> 20	<b>Ξ</b> d	2	4875DFC						
	53	± q0	F6904ABA		00000000		24875	DFC	$\Box$	
$\overline{\Delta a}$	<b>6</b> 86	⊞ q1		00000	000		F6904	ABA		
×				4875DFC is no		sters 19 and 12	2, now read in			
- <mark>0</mark> -			write enable	egister 19 as goes high.		rse the previou es written to the				
1										

The device performs as expected.

APPENDIX A: Textbook Problems (From the UNrevised Edition)

**4.2.1)** a) From the components in Figure 4.2, this instruction will use the PC, the instruction memory, the register file (both read ports and the write port), and the ALU.

**b)** Again from Figure 4.2, this instruction will use the PC, the instruction memory, the register file (one read port and the write port), and the mux between the instruction memory and the ALU.

**4.2.2)** a) This instruction will need a third register read port and a third ALU input port.

**b)** This instruction will need a shifting module, which probably should be placed in the ALU.

**4.2.3)** a) This instruction will need an additional ALU control signal that would correspond to a 3-input addition.

**b)** This instruction will need a ALU new control signal as well.

**4.8.1)** a) This bit is used only for shift amount (R-type) or constant/address value (I-type). One way would be to load a known value into a register and write it to address 0x110000000. If this value appears instead in address 0x100000000, then the bit is stuck at zero.

**b)** This signal is used only when loading data from memory. If it is stuck at zero, then the ouput of the ALU will be read instead of the memory. So one solution would be to write a certain number to memory and then attempt to load a register from that location in memory, referenced by a lower memory value with an offset, which must be different from the value in memory. The output of the ALU will be the offset, so if the register is loaded with this value after execution, then the signal is stuck on zero.

**4.8.2)** a) The same test as above could be used (using the memory values in opposite roles), or an ORI with 0000 and the zero register could be executed. If bit 7 of the result is 1, then the signal is stuck at 1.

**b)** The test above can't be used because there would be no way to know whether anything read from memory was already there or not. However, if this signal is stuck at 1, then it will not be possible to read from the ALU output. Any simple ALU operation would not assert this signal, so if a register's value does not reflect the output of the ALU, then the signal is stuck.

**4.8.3)** a) Yes. There is always more than one way to skin a cat when it comes to constants and offsets. This would require some more shifting around and some creative use of registers, but you could make it work.

**b)** No. If a register can't read from the ALU output, then nothing useful can be done.

```
APPENDIX B: Code for "registerFile"
-- Bob Minchin
-- EEL 4713C
-- Homework 3
-- Part 3.4: 32-Bit, 32-Register File
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;
ENTITY registerFile IS
      --32-bit input, output
      --5-bit mux signals
      --1-bit clock, write enable
      PORT (
                  rr0, rr1, rw :
                                    IN STD_LOGIC_VECTOR(4 DOWNTO 0) ;
                  d :
                                    IN STD LOGIC VECTOR(31 DOWNTO 0) ;
                                   IN STD_LOGIC ;
                  clk, wr :
                  q0, q1 :
                                    OUT STD_LOGIC_VECTOR(31 DOWNTO 0) );
END registerFile ;
ARCHITECTURE Structure OF registerFile IS
      --Temporary signals between output of registers and output of device
      --Arrays would have been better
      SIGNAL out00, out01, out02, out03, out04, out05, out06, out07, out08,
            out09, out10, out11, out12, out13, out14, out15, out16, out17,
            out18, out19, out20, out21, out22, out23, out24, out25, out26,
            out27, out28, out29, out30, out31 : STD LOGIC VECTOR(31 DOWNTO
                  0);
      --Individual write enables for the 32 muxes
     SIGNAL wr01, wr02, wr03, wr04, wr05, wr06, wr07, wr08, wr09, wr10, wr11,
            wr12, wr13, wr14, wr15, wr16, wr17, wr18, wr19, wr20, wr21, wr22,
            wr23, wr24, wr25, wr26, wr27, wr28, wr29, wr30, wr31 :
STD LOGIC ;
      --"reg32" component from Assignment 2
     COMPONENT reg32 IS
            PORT (
                                          IN STD LOGIC VECTOR(31 DOWNTO 0);
                        D :
                        Clk, Wr, Clr :
                                          IN STD LOGIC ;
                                          OUT STD_LOGIC_VECTOR(31 DOWNTO
                        Q:
                              0));
     END COMPONENT ;
BEGIN
     PROCESS(rr0, clk)
     BEGIN
            --Control logic for output 1
            IF clk'EVENT AND clk = '1' THEN
                  CASE rr0 IS
                        WHEN "00000" => q0 <= out00 ;
                        WHEN "00001" => q0 <= out01 ;
                        WHEN "00010" => q0 <= out02 ;
                        WHEN "00011" => q0 <= out03 ;
                        WHEN "00100" => q0 <= out04 ;
                        WHEN "00101" => q0 <= out05 ;
                        WHEN "00110" => q0 <= out06 ;
                        WHEN "00111" => q0 <= out07 ;
                        WHEN "01000" => q0 <= out08 ;
```

```
WHEN "01001" => q0 <= out09 ;
                  WHEN "01010" => q0 <= out10 ;
                  WHEN "01011" => q0 <= out11 ;
                  WHEN "01100" => q0 <= out12 ;
                  WHEN "01101" => q0 <= out13 ;
                  WHEN "01110" => q0 <= out14 ;
                  WHEN "01111" => q0 <= out15 ;
                  WHEN "10000" => q0 <= out16 ;
                  WHEN "10001" => q0 <= out17 ;
                  WHEN "10010" => q0 <= out18 ;
                  WHEN "10011" => q0 <= out19 ;
                  WHEN "10100" => q0 <= out20 ;
                  WHEN "10101" => q0 <= out21 ;
                  WHEN "10110" => q0 <= out22 ;
                  WHEN "10111" => q0 <= out23 ;
                  WHEN "11000" => q0 <= out24 ;
                  WHEN "11001" => q0 <= out25 ;
                  WHEN "11010" => q0 <= out26 ;
                  WHEN "11011" => q0 <= out27 ;
                  WHEN "11100" => q0 <= out28 ;
                  WHEN "11101" => q0 <= out29 ;
                  WHEN "11110" => q0 <= out30 ;
                  WHEN "11111" => q0 <= out31 ;
                  WHEN OTHERS =>
            END CASE ;
      END IF ;
END PROCESS ;
PROCESS(rr1, clk)
BEGIN
      --Control logic for output 2
      IF clk'EVENT AND clk = '1' THEN
            CASE rr1 IS
                  WHEN "00000" => q1 <= out00 ;
                  WHEN "00001" => q1 <= out01 ;
                  WHEN "00010" => q1 <= out02 ;
                  WHEN "00011" => q1 <= out03 ;
                  WHEN "00100" => q1 <= out04 ;
                  WHEN "00101" => q1 <= out05 ;
                  WHEN "00110" => q1 <= out06 ;
                  WHEN "00111" => q1 <= out07 ;
                  WHEN "01000" => q1 <= out08 ;
                  WHEN "01001" => q1 <= out09 ;
                  WHEN "01010" => q1 <= out10 ;
                  WHEN "01011" => q1 <= out11 ;
                  WHEN "01100" => q1 <= out12 ;
                  WHEN "01101" => q1 <= out13 ;
                  WHEN "01110" => q1 <= out14 ;
                  WHEN "01111" => q1 <= out15 ;
                  WHEN "10000" => q1 <= out16 ;
                  WHEN "10001" => q1 <= out17 ;
                  WHEN "10010" => q1 <= out18 ;
                  WHEN "10011" => q1 <= out19 ;
                  WHEN "10100" => q1 <= out20 ;
                  WHEN "10101" => q1 <= out21 ;
                  WHEN "10110" => q1 <= out22 ;
                  WHEN "10111" => q1 <= out23 ;
                  WHEN "11000" => q1 <= out24 ;
```

```
WHEN "11001" => q1 <= out25 ;
                  WHEN "11010" => q1 <= out26 ;
                  WHEN "11011" => q1 <= out27 ;
                  WHEN "11100" => q1 <= out28 ;
                  WHEN "11101" => q1 <= out29 ;
                  WHEN "11110" => q1 <= out30 ;
                  WHEN "11111" => q1 <= out31 ;
                  WHEN OTHERS =>
            END CASE ;
      END IF ;
END PROCESS ;
PROCESS(rw, wr)
BEGIN
      --all write enable initialized to 0
      wr01 <= '0' ;
      wr02 <= '0' ;
      wr03 <= '0' ;
      wr04 <= '0' ;
      wr05 <= '0' ;
     wr06 <= '0' ;
      wr07 <= '0' ;
     wr08 <= '0' ;
     wr09 <= '0' ;
      wr10 <= '0' ;
      wr11 <= '0' ;
     wr12 <= '0' ;
      wr13 <= '0' ;
      wr14 <= '0' ;
      wr15 <= '0' ;
      wr16 <= '0' ;
      wr17 <= '0' ;
      wr18 <= '0' ;
      wr19 <= '0' ;
      wr20 <= '0' ;
     wr21 <= '0' ;
     wr22 <= '0' ;
      wr23 <= '0' ;
      wr24 <= '0' ;
     wr25 <= '0' ;
     wr26 <= '0' ;
     wr27 <= '0' ;
     wr28 <= '0' ;
      wr29 <= '0' ;
      wr30 <= '0' ;
      wr31 <= '0' ;
      CASE rw IS
            --control logic for write enables
            WHEN "00001" =>
                  IF wr = '1' THEN wr01 <= '1' ;
                  END IF ;
            WHEN "00010" =>
                  IF wr = '1' THEN wr02 <= '1' ;
                  END IF ;
            WHEN "00011" =>
                  IF wr = '1' THEN wr03 <= '1' ;
                  END IF ;
            WHEN "00100" =>
```

IF wr = '1' THEN wr04 <= '1' ; END IF ; WHEN "00101" => IF wr = '1' THEN wr05 <= '1' ; END IF ; WHEN "00110" => IF wr = '1' THEN wr06 <= '1'; END IF ; WHEN "00111" => IF wr = '1' THEN wr07 <= '1'; END IF ; WHEN "01000" => IF wr = '1' THEN wr08 <= '1'; END IF ; WHEN "01001" => IF wr = '1' THEN wr09 <= '1' ; END IF ; WHEN "01010" => IF wr = '1' THEN wr10 <= '1'; END IF ; WHEN "01011" => IF wr = '1' THEN wr11 <= '1'; END IF ; WHEN "01100" => IF wr = '1' THEN wr12 <= '1' ; END IF ; WHEN "01101" => IF wr = '1' THEN wr13 <= '1'; END IF ; WHEN "01110" => IF wr = '1' THEN wr14 <= '1'; END IF ; WHEN "01111" => IF wr = '1' THEN wr15 <= '1' ; END IF ; WHEN "10000" => IF wr = '1' THEN wr16 <= '1' ; END IF ; WHEN "10001" => IF wr = '1' THEN wr17 <= '1' ; END IF ; WHEN "10010" => IF wr = '1' THEN wr18 <= '1' ; END IF ; WHEN "10011" => IF wr = '1' THEN wr19 <= '1' ; END IF ; WHEN "10100" => IF wr = '1' THEN wr20 <= '1' ; END IF ; WHEN "10101" => IF wr = '1' THEN wr21 <= '1' ; END IF ; WHEN "10110" => IF wr = '1' THEN wr22 <= '1'; END IF ; WHEN "10111" =>

IF wr = '1' THEN wr23 <= '1' ; END IF ; WHEN "11000" => IF wr = '1' THEN wr24 <= '1'; END IF ; WHEN "11001" => IF wr = '1' THEN wr25 <= '1'; END IF ; WHEN "11010" => IF wr = '1' THEN wr26 <= '1'; END IF ; WHEN "11011" => IF wr = '1' THEN wr27 <= '1'; END IF ; WHEN "11100" => IF wr = '1' THEN wr28 <= '1' ; END IF ; WHEN "11101" => IF wr = '1' THEN wr29 <= '1'; END IF ; WHEN "11110" => IF wr = '1' THEN wr30 <= '1'; END IF ; WHEN "11111" => IF wr = '1' THEN wr31 <= '1'; END IF ; WHEN OTHERS => END CASE ; END PROCESS ; reg32 PORT MAP (d, clk, '0', '1', out00); Req00 : reg32 PORT MAP (d, clk, wr01, '1', out01) ; Reg01 : reg32 PORT MAP (d, clk, wr02, '1', out02) ; Reg02 : Reg03 : reg32 PORT MAP (d, clk, wr03, '1', out03) ; reg32 PORT MAP (d, clk, wr04, '1', out04) ; Reg04 : reg32 PORT MAP (d, clk, wr05, '1', out05) ; Reg05 : reg32 PORT MAP (d, clk, wr06, '1', out06) ; Req06 : reg32 PORT MAP (d, clk, wr07, '1', out07) ; Req07 : reg32 PORT MAP (d, clk, wr08, '1', out08) ; Reg08 : reg32 PORT MAP (d, clk, wr09, '1', out09) ; Req09 : reg32 PORT MAP (d, clk, wr10, '1', out10); Real0 : reg32 PORT MAP (d, clk, wr11, '1', out11) ; Reg11 : reg32 PORT MAP (d, clk, wr12, '1', out12) ; Reg12 : reg32 PORT MAP (d, clk, wr13, '1', out13); Reg13 : Reg14 : reg32 PORT MAP (d, clk, wr14, '1', out14) ; reg32 PORT MAP (d, clk, wr15, '1', out15) ; Reg15 : reg32 PORT MAP (d, clk, wr16, '1', out16) ; Reg16 : reg32 PORT MAP (d, clk, wr17, '1', out17) ; Req17 : reg32 PORT MAP (d, clk, wr18, '1', out18) ; Reg18 : reg32 PORT MAP (d, clk, wr19, '1', out19) ; Reg19 : reg32 PORT MAP (d, clk, wr20, '1', out20) ; Reg20 : reg32 PORT MAP (d, clk, wr21, '1', out21) ; Reg21 : Reg22 : reg32 PORT MAP (d, clk, wr22, '1', out22) ; Reg23 : reg32 PORT MAP (d, clk, wr23, '1', out23) ; Req24 : reg32 PORT MAP (d, clk, wr24, '1', out24) ; reg32 PORT MAP (d, clk, wr25, '1', out25) ; Req25 : reg32 PORT MAP (d, clk, wr26, '1', out26) ; Reg26 :

Reg27	:	reg32	PORT	MAP	(d,	clk,	wr27,	'1',	out27)	;
Reg28	:	reg32	PORT	MAP	(d,	clk,	wr28,	'1',	out28)	;
Reg29	:	reg32	PORT	MAP	(d,	clk,	wr29,	'1',	out29)	;
Reg30	:	reg32	PORT	MAP	(d,	clk,	wr30,	'1',	out30)	;
Reg31	:	reg32	PORT	MAP	(d,	clk,	wr31,	'1',	out31)	;

END STRUCTURE ;