CompArch: Lab 1

Zoe Fiddler, Ezra Varady October 15, 2015

1 ALU Design

For the ALU we opted to implement 32-bit logic rather than a bit slice. While the design is less flexible the implementation is similarly performant, and it allows scaling to widths that are multiples of 32 bits. This required us to implement 32-bit versions of each of the operations we support in the ALU. For the logical operations, XOR, NAND etc. this was accomplished by attaching 32 instances of the two input version of the gate, connecting the two bits in each place of the operands. The adder we used was a generalized version of the 4-bit adder we wrote previously. Subtraction was implemented as an adder with an inverter on top. SLT is calculated as XOR of the most significant bit of the result of subtraction and its overflow. All of these inputs connect to a 3-bit MUX where all eight inputs and the output are 32 bit values. Flags are activated only when the opcode is zero or one, with carryout and overflow being drawn from the proper adder. Zero is NOR of all the bits output by the MUX.

2 Test Cases

2.1 Adder Subtractor

We aimed to create test cases for each of the different black box behaviors as well as propagation through the adder. This includes testing proper sum is found in cases with no carryout or overflow, carryout and no overflow, overflow and no carryout and both carryout and overflow. The test bench produced in Modelsim can be seen in Figure 1 on the following page. The same bench was used in subtraction, with the second operand negated. This produced the exact same test bench, seen in Figure 2 on the next page. One problem we debugged using this test bench was that the flags were not being set on subtraction. The error turned out to be due to an indexing error in our code.

2.2 Logic

For the logic outputs we tested each of the different modes AND, NAND, OR, NOR, XOR, using the same 2 operands. By doing this we could see that each function was behaving properly. No errors were encountered in this process. The test bench can be seen in Figure 3 on the following page

<testing ad<="" th=""><th>ider</th><th></th><th></th><th></th><th></th><th></th><th></th></testing>	ider						
Testing Addition gives e	expected value for small values (no	са	rryout or overflow)				
Operand 1	Operand 2	1	Result	-1	Zero	Carryout	Overflow
000000000000000000000000000000000000000	00000000000000000000000000000110	1	000000000000000000000000000001011	-1	0	0	0
000000000000000000000000000000000000000	00000000000000000000000000000011	1	00000000000000000000000000000111	-1	0	0	0
00000000000000000000000001100100	00000000000000000000000001100100	1	00000000000000000000000011001000	- 1	0	0	0
Testing cases with carry	yout but no overflow						
Operand 1	Operand 2	1	Result	-1	Zero	Carryout	Overflow
111111111111111111110110001111000	00000000000000000010011100010000	1	00000000000000000001001110001000	-1	0	1	0
00000000000000010011100010000000	1111111111111111100111110010110000	1	0000000000000000111010100110000	- 1	0	1	0
1111111111111111101100011111000000	111111111111111110001010110100000	1	11111111111111100111100101100000	-1	0	1	0
Testing cases with overf	flow but no carryout						
Operand 1	Operand 2	1	Result	-1	Zero	Carryout	Overflow
010000000000000000000000000000000	010000000000000000000000000000000000000	1	100000000000000000000000000000000000000	-1	0	0	1
01000000000000000000111010010011	01000000000000110100001111110000	1	10000000000000110101001010000011	-1	0	0	1
Testing cases with both	overflow and carryout						
Operand 1	Operand 2	1	Result	-1	Zero	Carryout	Overflow
101111111111111111111000101101101	1011111111111110010111110000010000	1	01111111111111100101011011011111101	-1	0	1	1
10001000110010110100000010101101	10000100011000011111001000010000	1	00001101001011010011001010111101	-1	0	1	1
Testing inverses sum to	0, Zero flag set to 1, carryout but	t n	o overflow				
Operand 1	Operand 2	1	Result	-1	Zero	Carryout	Overflow
11111111111111111111111110110110	0000000000000000000000000100101010	Ť.	000000000000000000000000000000000000000	- 1	1	1	0
1111111111111111111011000111110000	00000000000000000010011100010000	Ť.	000000000000000000000000000000000000000	i	1	1	0

Figure 1: Test bench for 32-bit adder functionality of an ALU

ı	<testing s<="" th=""><th>ubtraction</th><th></th><th></th><th></th><th></th><th></th><th>></th></testing>	ubtraction						>
	Testing Subtraction giv	es expected value for small values	(no	carryout or overflow)				
1	Operand 1	Operand 2	- 1	Result	Ze	ero	Carryout	Overflow
1	000000000000000000000000000000000000000	111111111111111111111111111111111111	- 1	000000000000000000000000000001011	1	0	0	0
1	000000000000000000000000000000000000000	11111111111111111111111111111111111111	- 1	00000000000000000000000000000111	1	0	0	0
	00000000000000000000000001100100	111111111111111111111111110011100	- 1	00000000000000000000000011001000	1	0	0	0
ł	Testing cases with carr	yout but no overflow						
1	Operand 1	Operand 2	- 1	Result	Ze	ero	Carryout	Overflow
ı	1111111111111111111101100011111000	1111111111111111111011000111110000	- 1	0000000000000000001001110001000	1	0	1	0
1	00000000000000010011100010000000	0000000000000001100001101010000	- 1	0000000000000000111010100110000	1	0	1	0
1	1111111111111111101100011111000000	0000000000000001110101001100000	- 1	1111111111111111001111100101100000	1	0	1	0
ı	Testing cases with over	flow but no carryout						
ı	Operand 1	Operand 2	- 1	Result	Ze	ero	Carryout	Overflow
ı	010000000000000000000000000000000000000	110000000000000000000000000000000000000	- 1	100000000000000000000000000000000000000	1	0	0	1
ı	0100000000000000000111010010011	1011111111111110010111110000010000	- 1	1000000000000011010101001010000011	1	0	0	1
ı	Testing cases with both	overflow and carryout						
1	Operand 1	Operand 2	- 1	Result	Ze	ero	Carryout	Overflow
ı	10111111111111111111000101101101	0100000000000110100001111110000	- 1	0111111111111100101010110101111101	1	0	1	1
1	10001000110010110100000010101101	011110111001111000001101111110000	- 1	00001101001011010011001010111101	1	0	1	1
1	Testing the difference	between a number and itself is 0, 2	ero	flag set to 1				
1	Operand 1	Operand 2	- 1	Result	Ze	ero	Carryout	Overflow
1	111111111111111111111111111111111111111	11111111111111111111111111011010110	- 1	000000000000000000000000000000000000000	1	1	1	0
1	00000000000000000010011100010000	0000000000000000010011100010000	- 1	000000000000000000000000000000000000000	1	1	1	0

Figure 2: Test bench for 32-bit subtractor functionality of an ALU

<testing subtrac<="" th=""><th>tion</th><th></th><th></th><th></th><th></th><th>></th></testing>	tion					>
Testing Subtraction gives exp	ected value for small values	(no	carryout or overflow)			
Operand 1 0	perand 2		Result	Zero	Carryout	Overflow
000000000000000000000000000000101 1111	11111111111111111111111111111010		000000000000000000000000000001011	1 0	0	0
000000000000000000000000000000000000000	111111111111111111111111111111		00000000000000000000000000000111	1 0	0	0
0000000000000000000000001100100 1111	11111111111111111111110011100		00000000000000000000000011001000	1 0	0	0
Testing cases with carryout b	out no overflow					
Operand 1 0	perand 2	1	Result	Zero	Carryout	Overflow
1111111111111111111101100011111000 11111	111111111111111011000111110000	1	00000000000000000001001110001000	1 0	1	0
00000000000000010011100010000000 0000	00000000000001100001101010000	1	0000000000000000111010100110000	1 0	1	0
1111111111111111101100011111000000 0000	00000000000001110101001100000	1	1111111111111111001111100101100000	1 0	1	0
Testing cases with overflow b	out no carryout					
Operand 1	perand 2	1	Result	Zero	Carryout	Overflow
010000000000000000000000000000000000000	000000000000000000000000000000000000000		100000000000000000000000000000000000000	1 0	0	1
01000000000000000000111010010011 1011	1111111111001011110000010000	1	10000000000000110101001010000011	1 0	0	1
Testing cases with both overf	low and carryout					
Operand 1	perand 2	1	Result	Zero	Carryout	Overflow
10111111111111111111000101101101 0100	0000000000110100001111110000	1	0111111111111110010101101011111101	1 0	1	1
10001000110010110100000010101101 0111	1011100111100000110111110000	1	00001101001011010011001010111101	1 0	1	1
Testing the difference betwee	en a number and itself is 0, Ze	ero	flag set to 1			
Operand 1 0	perand 2	1	Result	Zero	Carryout	Overflow
111111111111111111111111011010110 1111	111111111111111111110110110	1	000000000000000000000000000000000000000	1 1	1	0
00000000000000000010011100010000 0000	000000000000000010011100010000		000000000000000000000000000000000000000			_
	Testing Subtraction gives expression of the set of the	Testing cases with overflow but no carryout Operand 1 Operand 2 11111111111111111111111111111111111	Testing cases with overflow but no carryout Operand 1 Operand 2 Operand 2 Operand 2 Operand 2 Operand 3 Operand 4 Operand 5 Operand 5 Operand 5 Operand 6 Operand 6 Operand 7 Operand 7 Operand 7 Operand 7 Operand 7 Operand 8 Operand 8 Operand 9 Op	Testing Subtraction gives expected value for small values (no carryout or overflow) Operand 1	Testing Subtraction gives expected value for small values (no carryout or overflow) Operand 1	Operand 1 Operand 2 Result Zero Carryout 000000000000000000000000000000000000

Figure 3: Test bench for 32-bit logic functionalities of an ${\rm ALU}$

2.3 SLT

For the set less than function we tested a variety of cases with both positive and negative numbers, which can be seen in Figure 4. In all of these cases, the flags should be set to 0 at all times. Using this test bench we discovered a wiring problem which caused an error in the lsb of the output.

# <testing si<="" th=""><th>LT</th><th></th><th></th><th></th><th></th><th></th><th>></th><th></th><th></th></testing>	LT						>		
# Operand 1	Operand 2	1	Result	-1	Expected	1	Zero	Carryout	Overflow
# 0000000000000000000000000000000000000	000000000000000000000000000000000000000	1	000000000000000000000000000000000000000	- 1	1	l i	0	0	0
# 0000000000000000000000000000000000000	000000000000000000000000000000000000000	1	000000000000000000000000000000000000000	- 1	0	l.	0	0	0
# 0000000000000000000000000000000000000	000000000000000000000000000000000000000	1	000000000000000000000000000000000000000	- 1	0	i.	0	0	0
# 11111111111111111111111111111111111	11111111111111111111111111111111111111	1	000000000000000000000000000000000000000	-1	0	i.	0	0	0
# 1111111111111111111111111111111111	11111111111111111111111111111111111111	1	000000000000000000000000000000000000000	- 1	1	i.	0	0	0

Figure 4: Test bench for 32-bit SLT functionality of ALU

Table 1: Worst case propagation delays

Operation	Propagation Delay						
Logical operations	160ns						
Add	1820						
Sub	1830						
SLT	1470						

3 Work Plan Reflection

We followed our work plan exceedingly well. The day before the lab, we sat down and did most of it. We then, as planned, smoked several (3-7) cigarettes. After a brief nap, as the kids are calling it, we wrote this report. Some may attribute our success to the nonexistance of a work plan before the writing of this report, however we *vehemently* deny that this was the case.