Assignment 3

Ch4 textbook problems, bugspim
VHDL: add32, alu32, alu32control, registerFile

Andrew Fowler

2/13/2012

Introduction:

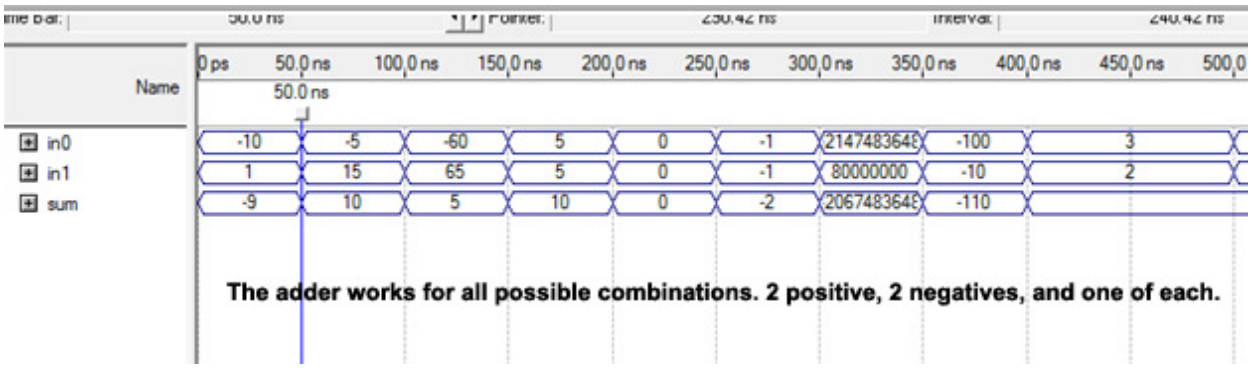
This assignment was split into two main parts. The assignment consisted of a MIPS simulator that had bugs in it (called bugspim) and VHDL design. The MIPS simulator had 5 bugs in it that the user had to find and report on. The VHDL design consisted of designing a 32 bit adder, an ALU, ALUcontroller, registerFile which will all be used in the final MIPS processor data path.

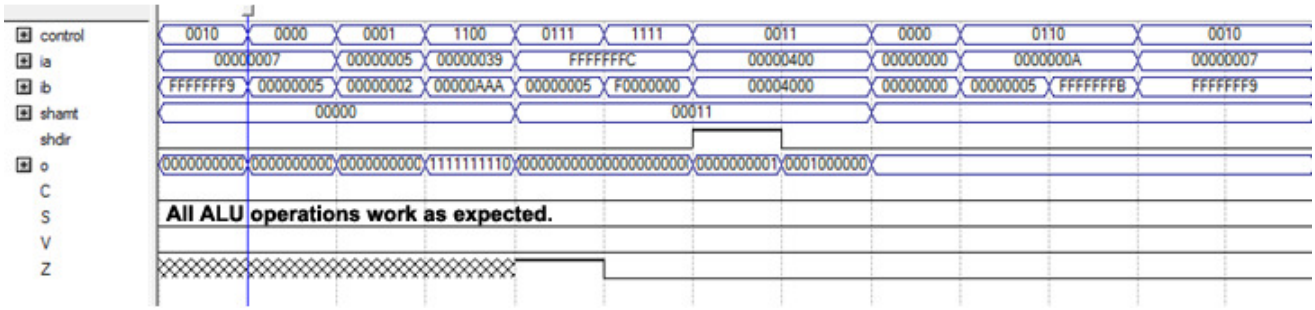
Design and Testing:

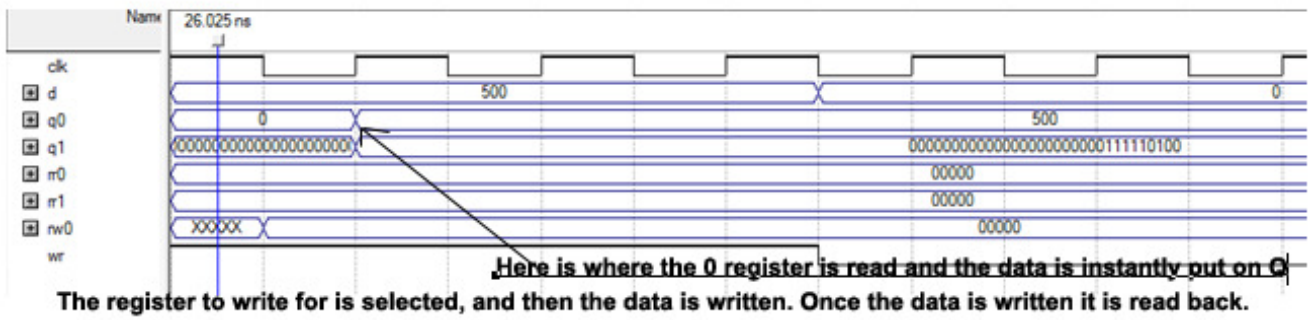
VHDL code:

All VHDL is included in the zip file where this document was found.

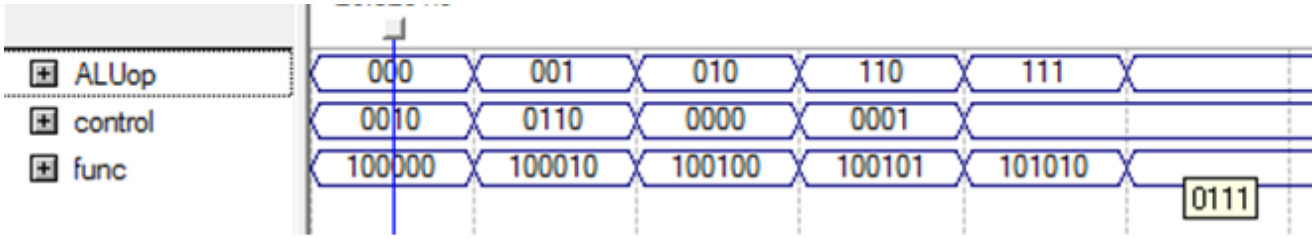
VHDL simulations:







The register to write for is selected, and then the data is written. Once the data is written it is read back.

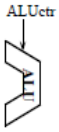


The control signals follow the table found in the assignment PDF. When ALUop and func are the correct signals then the correct control is outputted.

the sum of "ia" and "ib" when "control" = "0010"
the difference of "ia" and "ib" when "control" = "0110"
the logical AND of "ia" and "ib" when "control" = "0000"
the logical OR of "ia" and "ib" when "control" = "0001"
the logical NOR of "ia" and "ib" when "control" = "1100"
the slt operation of signed "ia" and "ib" when "control" = "0111"
the slt of unsigned "ia" and "ib" when "control" = "1111"
the logical shift of "ib", in the direction indicated by "shdir" ('0' left amount "shamt" when "control" = "0011"

funct<5:0>	Instruction Operation
10 0000	add
10 0010	subtract
10 0100	and
10 0101	or
10 1010	set-on-less-than

EEL-4713 Renato Figueiredo



ALUctr<2:0>	ALU Operation
000	And
001	Subtract
010	Add
110	Or
111	Set-on-less-than

Bugspim problems:

Lui – larger.s file – went to a bad location in memory and created a runtime error.
BEQ - test.s – branched when the 2 values were not equal.
JAL – test.s – does not work properly.
SH – test.s – compile errors when tried to do code that would work.
ADDU – test.s – works as a normal add would work with negative numbers.

Textbook Questions – old edition of the book

Chapter 4 questions: 4.2.1-3, 4.8.1-3

Question 4.2.1

a. It could reuse the ALU, instruction memory and the Registers black (although additions would need to be made to add the third register read in) and the data path between the ALU and registers block.
b. It would reuse the ALU, instruction memory, and the registers block and the data path between immediate values and ALU.

Question 4.2.2

a. The registers block needs to be expended to include reading 3 registers. The alu also needs to be extended to accept one more register. The instruction memory might have to be modified to

hold 3 registers values and any control logic that would know to look for 3 registers would also need to be changed.

b. Nothing needs to be added, the control logic is there and the registers block will accept the instruction. The ALU would be in charge of the shift.

Question 4.2.3

a. New control signals would need to be added to the ALU to decide when the third register needs to be added.

b. No change needed, the SLL instruction is already a mips instruction and should have the needed control logic to control the ALU.

Question 4.8.1

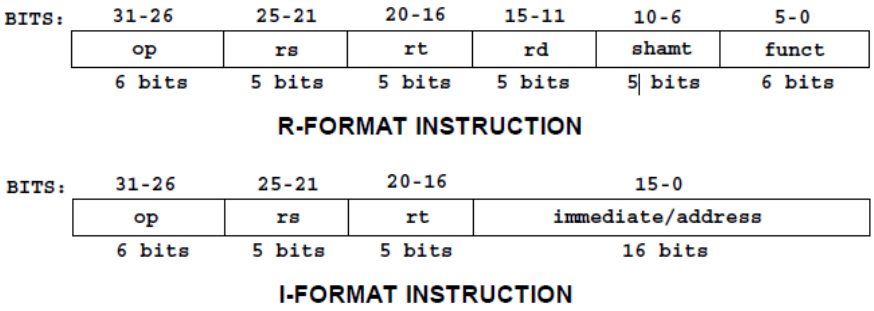


Figure 1 Credit: http://jjc.hydrus.net/cs61c/handouts/formats4.pdf

a. If bit 7 is effected then the immediate/address part will be altered. Any add immediate instruction will show the problem. ADDI $t1, $zero, 0xFFFF – if $t1 is 0xFFFF then the bit is not stuck at 0, if it is any other output there is a problem.

b. Having the bit set to 0 would make data be read from the ALU instead of the memory. Pick an address that with an offset from a certain location (since the ALU controls offset) and write a value to the memory location that is different from the offset. Now try to read that memory location and see if the offset is the data that is produced, or if the actual data from the memory location is read.

Question 4.8.2

There is no way that we can set a bit to 1 for the 0 test, and 0 for the 1 test at the same time. So it is not possible to test both combinations at the same time.

a. ADDI $t1, $zero, 0x0000 – if $t1 is 0x0000 then the bit is not stuck at 1, if it is any other output there is a problem.

b. There is no 100% reliable test

Question 4.8.3

a. When the bit was in the immediate field then the instructions following would have to add (or subtract) and extra 127 and then 1 from the immediate value. When it is an offset offset it an extra 128 when needed.

b. This problem is incurable since there is no other way to store most results into registers (unless they are coming from memory).