Andrew Smith
ECE 411 – Written Hand In
09/14/17

Test Description

Testing such a large system in such a small amount of time was initially a daunting task. I decided I would break the testing down into two parts, individual instruction tests as well as running the provided test code. If my individual unit tests passed then theoretically the whole system should work fully because instructions are independent of the previous instruction.

I developed test programs that would test only a small set of similar instructions being sure to test different base registers or immediate values. I then ran these small snippets of code to be sure that the LC3b behaved as expected and that the proper control signals were set.

After writing the small individual code segments that test different versions of single instructions I ran them in model sim. I looked at each individual instruction ensuring all of the control signals were as expected. I also verified that the test program did what it was supposed to by looking at the memory before and after.

After the unit tests were complete and I had verified that all the instructions were correct I ran the provided test code and saw that the output of that code matched running that program in the lc3sim tool thus I concluded that my design was correct.

A good example of my test code is the mp1_jsr_ret.asm test program. It tests both versions of the JSR instruction by jumping to a subroutine that has a bunch of shift operations then it returns and adds two registers. If the JSR instruction worked correctly I would observe on the wave trace that the previous PC would store to R7 and the new PC would be set to the location of the tag "SUBROUTINE". If all of the shifts and modifications to registers R2, R3, and R4 happened correctly then I would know the processor executed the code properly.

Datapath Changes

1. addr1mux – NEW – selects either the PC+2 or the SR1 register to be added with the output of addr2mux to generate a new PC.
2. addr2mux – NEW – selects either 0x0000, sext[IR[10:0] << 1], sext[IR[8:0] << 1], sext[IR[5:0] << 1] to calculate an offset for a JMP, JSR, JSRR, or BR instruction.
3. pc_add – NEW – adds the outputs of the addr1mux and addr2mux.
4. pcmux – MODIFIED – extended to load the pc_offset, alu_out, or the mem_wdata into the PC.
5. marmux – MODIFIED – extended to allow the zext8 of the trapvect to be loaded, and the mem_wdata to be loaded for the indirect accesses.
6. mdrmux – MODIFIED – extended to allow a byte to be written to memory
7. destmux – NEW – added so that for the JSR and JSRR instructions the R7 can be loaded with the old PC+2.
8. alumux – MODFIED – extended to allow for an imm4 and an offset6 to be added through the ALU. The imm4 is for the shift functionality and the offset6 is for calculating the address in LDB and STB instructions.
9. regfilemux – MODIFIED – extended to allow the byte contents of a memory address to be written and so that the pc_off value can be stored into registers. (LEA, JSR, JSRR)
10. Sign+Zero extensions – NEW – added a bunch of sign extensions and zero extension blocks to adjust signals and extend them to 16 bit width.

11. IR – MODIFIED – modified this block to support a plethora of output signals used throughout the datapath.