Descriptions of Highlighted Blocks Except Muxes:

IR: Added

a: For use in the shf instruction.

d: For use in the shf instruction.

Jsr_trigger: to differentiate between jsr and jsrr.

immediate: to differentiate between immediate and non immediate arithmetic operations.

Imm4: extended imm4 for shf instruction.

Imm5: extended imm5 for arithmetic operations.

Trapvector: extended trap vector for trap instruction.

Adj6 & Adj9: I added an output for sign extended offset without the shift left for use in the ldb and stb instruction.

Description of Testing:

In order to adequately test the design of the extended lc3b architecture. I initially ran it against the checkpoint 1 code. I noticed that all the registers were the what I expected them to be and “don’t cares” do not appear at any time in the execution. To verify my register values, I used the lc3bIDE to check at each instruction that the register values were accurate. Next, I used the mp1-final.asm to test my new architecture to give a more thorough test than simply the cp1 code. Once I was satisfied that all the register values were correct in the execution, I needed to verify further that all store instructions were valid. Although the test code provided loosely tested that the stored values were accurate, I wanted to verify each store and load instruction in isolation. To accomplish this, I modified, and later reverted, the given code to store some information into memory, cleared the register and then loaded the value back into it. For example, I would use sti to store some value in memory, clear the source register and then ldi with the same address to verify the value I stored was not corrupted. For ldb and stb, I would store a byte at low and high bytes in a single 2byte memory slot and load the same address separately using ldb. If I could reconstruct the two values in a predictable way, then I could conclude that the ldb and stb were acting correctly.

To test the arithmetic operations on the new architecture. I found it sufficient to use the sample code provided because it extensively uses the immediate and non immediate operations as well as all the possible shifts. In addition to this, shifts were accomplished merely by changing the aluop and making sure sign was preserved. Therefore, extensive testing, in my mind, was not required.

Finally, to test instructions like jsr, jsrr and trap, I used modelsim closely to verify that all goals listed in the RTL of the instruction were accomplished. This included the changing pc, extension of trap vector if applicable and storage of the non incremented pc into r7 and all of this being done in the correct order and correctly returning upon ret or end of trap etc.