Jesse Chen, Ojan Thonrycroft

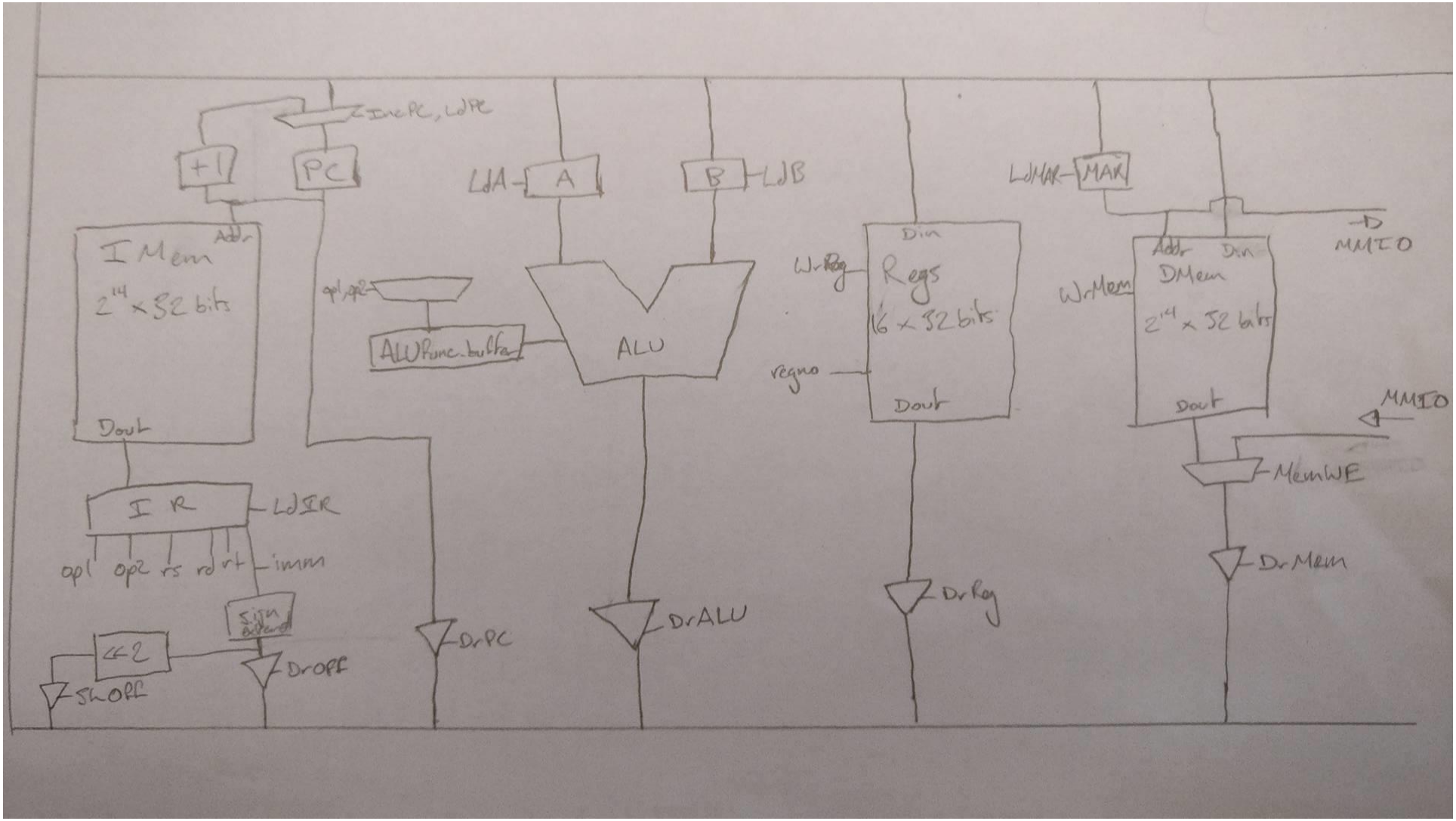
Assignment 2

There were a number of design options to choose from in the lecture slides, but for this project, we decided to mostly stick with the final design that was agreed upon in class. More specifically, we chose a design (pictured below) that most closely resembled what we were already most familiar with from CS 2110 and CS 2200, with a few minor modifications to fit the design given in class. As such, a majority of the code given to us was left unchanged. The only code that had to be added involved filling in the logic within certain components of the design that were already outlined for us. This includes decoding the instructions, implementing the ALU, implementing the FPGA board's memory-mapped I/O, defining the processor states and their transitions (also pictured below), and implementing the sign extender.

Decoding instructions was simple enough, since we just used the prescribed ISA to decode the 32-bit instructions into each of the six possible components: op1 [31:26], op2 [25:18], imm [23:8], rd [11:8], rs [7:4], and rt [3:0]. The ALU was mostly just a matter of defining each of its 14 different operations and mapping its output depending on what function it was currently set to use. Implementing our FPGA board's memory-mapped I/O involved overriding the LW and SW instructions when the MAR contained certain addresses that mapped to the board's I/O. In these cases, instead of reading and writing to data memory, it read and wrote to the board's I/O. In terms of processor states, our processor had a total of 36 different states, including an error state. Each instruction has a series of states that it goes through before restarting to fetch the next instruction, and each state defines which signals are to be turned on at any point in an instruction's execution. Finally, a sign extender module was already given to us, but we simply had to define its parameters and add it to our design.

The biggest issue that I ran into was that our ALU didn't seem to do anything but add. After a very long debugging process, I found that ALUfunc was getting reset to 0 at every clock cycle, which is mapped to addition. This was a problem because we had initially set ALUfunc during FETCH2, before the ALU was ready to drive its output in ALUI3/ALUR3. To fix this, I added an ALUfunc_buffer as a new register to save ALUfunc until the ALU was ready. Now, where we originally set ALUfunc in FETCH2, I set ALUfunc_buffer instead, and in ALUI3/ALUR3, I set ALUfunc to ALUfunc_buffer. This fixed the ALU problem we were having; it also made the compiler warn us not to use such a register as a latch, but I'm not sure why.

My individual contribution to this project consisted of designing our processor's state machine and working out how the memory-mapped I/O worked. My original code to handle memory-mapped I/O didn't work to begin with, but my partner and I both debugged it together, along with the rest of the code as well. I believe my contribution was about 40-50% of the project.


IncPC, LdPC
+1
PC
LdA
A
B
LdB
LdMAR
MAR
MMIO
IMem
Addr
2^14 x 32 bits
Dout
op/opc
ALUFunc_bufFn
ALU
WrReg
Regs
16 x 32 bits
Din
regno
Dout
WrMem
DMem
Addr
Din
2^14 x 32 bits
Dout
MMIO
MemWE
IR
LdIR
op1
op2
rs
rd
rt
imm
sign extend
<<2
DrOff
ShOff
DrPC
DrALU
DrReg
DrMem

State #	Macro State	LdPC	DrPC	IncPC	LdMAR	WrMem	DrMem	LdIR	DrOff	ShOff	LdA	LdB	ALUfunc	DrALU	RegSel	DrReg	WrReg	next_state
0	FETCH1	0	0	1	0	0	0	1	0	0	0	0	0000	0	00	0	0	FETCH2
1	FETCH2	0	0	0	0	0	0	0	0	0	0	0	0000	0	00	0	0	Check op1 and op2
2	ALUR1	0	0	0	0	0	0	0	0	0	1	0	0000	0	00	1	0	ALUR2
3	ALUR2	0	0	0	0	0	0	0	0	0	0	1	0000	0	01	1	0	ALUR3
4	ALUR3	0	0	0	0	0	0	0	0	0	0	0	0011	1	10	0	1	FETCH1
5	ALUI1	0	0	0	0	0	0	0	0	0	1	0	0000	0	00	1	0	ALUI2
6	ALUI2	0	0	0	0	0	0	0	1	0	0	1	0000	0	00	0	0	ALUI3
7	ALUI3	0	0	0	0	0	0	0	0	0	0	0	0000	1	01	0	1	FETCH1
8	BEQ1	0	0	0	0	0	0	0	0	0	1	0	0000	0	00	1	0	BEQ2
9	BEQ2	0	0	0	0	0	0	0	0	0	0	1	0000	0	01	1	0	BEQ3
10	BEQ3	0	0	0	0	0	0	0	0	0	0	0	1010	1	00	0	0	(bus==1)?BR1:FETCH1
11	BNE1	0	0	0	0	0	0	0	0	0	1	0	0000	0	00	1	0	BNE2
12	BNE2	0	0	0	0	0	0	0	0	0	0	1	0000	0	01	1	0	BNE3
13	BNE3	0	0	0	0	0	0	0	0	0	0	0	1101	1	00	0	0	(bus==1)?BR1:FETCH1
14	BLT1	0	0	0	0	0	0	0	0	0	1	0	0000	0	00	1	0	BLT2
15	BLT2	0	0	0	0	0	0	0	0	0	0	1	0000	0	01	1	0	BLT3
16	BLT3	0	0	0	0	0	0	0	0	0	0	0	1011	1	00	0	0	(bus==1)?BR1:FETCH1
17	BLE1	0	0	0	0	0	0	0	0	0	1	0	0000	0	00	1	0	BLE2
18	BLE2	0	0	0	0	0	0	0	0	0	0	1	0000	0	01	1	0	BLE3
19	BLE3	0	0	0	0	0	0	0	0	0	0	0	1100	1	00	0	0	(bus==1)?BR1:FETCH1
20	BR1	0	1	1	0	0	0	0	0	0	1	0	0000	0	00	0	0	BR1
21	BR2	0	0	0	0	0	0	0	0	1	0	1	0000	0	00	0	0	BR2
22	BR3	1	0	0	0	0	0	0	0	0	0	0	0000	1	00	0	0	FETCH1
23	JAL1	0	1	0	0	0	0	0	0	0	0	0	0000	0	01	0	1	JAL2
24	JAL2	0	0	0	0	0	0	0	0	0	1	0	0000	0	00	1	0	JAL3
25	JAL3	0	0	0	0	0	0	0	0	1	0	1	0000	0	00	0	0	JAL4
26	JAL4	1	0	0	0	0	0	0	0	0	0	0	0000	1	00	0	0	FETCH1

27	SW1	0	0	0	0	0	0	0	0	0	1	0	0000	0	00	1	0	SW2
28	SW2	0	0	0	0	0	0	0	1	0	0	1	0000	0	00	0	0	SW3
29	SW3	0	0	0	1	0	0	0	0	0	0	0	0000	1	00	0	0	SW4
30	SW4	0	0	0	0	1	0	0	0	0	0	0	0000	0	01	1	0	FETCH1
31	LW1	0	0	0	0	0	0	0	0	0	1	0	0000	0	00	1	0	LW2
32	LW2	0	0	0	0	0	0	0	1	0	0	1	0000	0	00	0	0	LW3
33	LW3	0	0	0	1	0	0	0	0	0	0	0	0000	1	00	0	0	LW4
34	LW4	0	0	0	0	0	1	0	0	0	0	0	0000	0	01	0	1	FETCH1

	ALUfunc				RegSel	
0	0000	ADD		0	00	Rs
1	0001	AND		1	01	Rt
2	0010	OR		2	10	Rd
3	0011	XOR		3	11	N/A
4	0100	SUB				
5	0101	NAND				
6	0110	NOR				
7	0111	NXOR				
8	1000	RSHF				
9	1001	LSHF				
10	1010	EQ				
11	1011	LT				
12	1100	LE				
13	1101	NE				