Grant Lance
ECE 474 Spring 2024
Final Project - Microprocessor

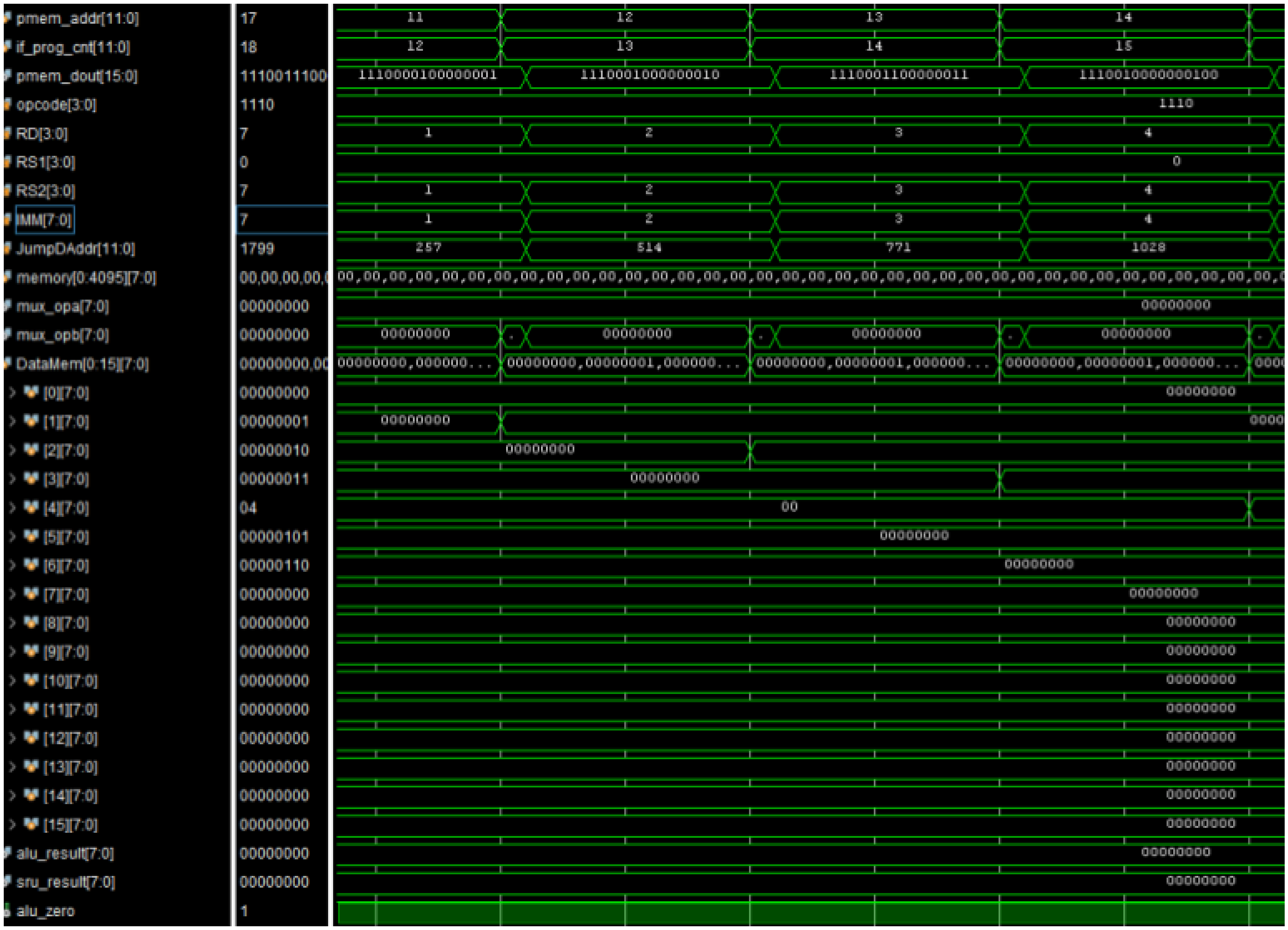
This document is broken into two sections- Validation and Unresolved Issues. The first will focus on validating the functionalities that my microprocessor has- ALU/SRU commands, MEMW/MEMR, JMPC/JMPD, and LDI. The second will focus on the issues that I could not resolve before the deadline. Mainly this was the implementation of the stack but also no-op.

Validation

A single test bench that "turns the processor on" is used to demo the code and test its functionality. This is a bit brute force, but allowed me to see how my changes would affect the overall performance of the processor. The main functionalities are split amongst the if_module and ex_module. if_module handles reading the ROM and storing the program count to a stack for JMPS/RETS. ex_module handles all ALU/SRU operations as well as managing the data memory registers and storage (addressable BRAM). All memory was initialized from the block memory generator in the IP Catalog. This section will have screenshots of the test bench with narration as well as a truth table with expected/actual results. First LDI and ALU/SRU functions will be demonstrated, then MEMW/MEMR, and finally JMPC/JMPD.

ALU/SRU

The ALU/SRU blocks perform the following functions NAND, NOR, XOR, ADD, ROTL, SLA, SRA, and SRL. In order to be able to demonstrate their functionality LDI is needed alongside: a top module being initiated connecting IF/EX, EX to have ROM reading capabilities, and IF being able to manage a register bank correctly. By testing some commands to load the register banks and perform some operations on them by using the ALU/SRU, all of these functionalities can be validated at once. Figure 1 shows LDI and NOR.



```
pmem_addr[11:0]      17
if_prog_cnt[11:0]    18
pmem_dout[15:0]      1110011100
opcode[3:0]          1110
RD[3:0]              7
RS1[3:0]             0
RS2[3:0]             7
IMM[7:0]             7
JumpDAddr[11:0]      1799
memory[0:4095][7:0]  00,00,00,00,0
mux_opa[7:0]         00000000
mux_opb[7:0]         00000000
DataMem[0:15][7:0]   00000000,00
  [0][7:0]           00000000
  [1][7:0]           00000001
  [2][7:0]           00000010
  [3][7:0]           00000011
  [4][7:0]           04
  [5][7:0]           00000101
  [6][7:0]           00000110
  [7][7:0]           00000000
  [8][7:0]           00000000
  [9][7:0]           00000000
  [10][7:0]          00000000
  [11][7:0]          00000000
  [12][7:0]          00000000
  [13][7:0]          00000000
  [14][7:0]          00000000
  [15][7:0]          00000000
alu_result[7:0]      00000000
sru_result[7:0]      00000000
alu_zero             1
```

Figure 1: Screenshot of LDI and NOR functionalities

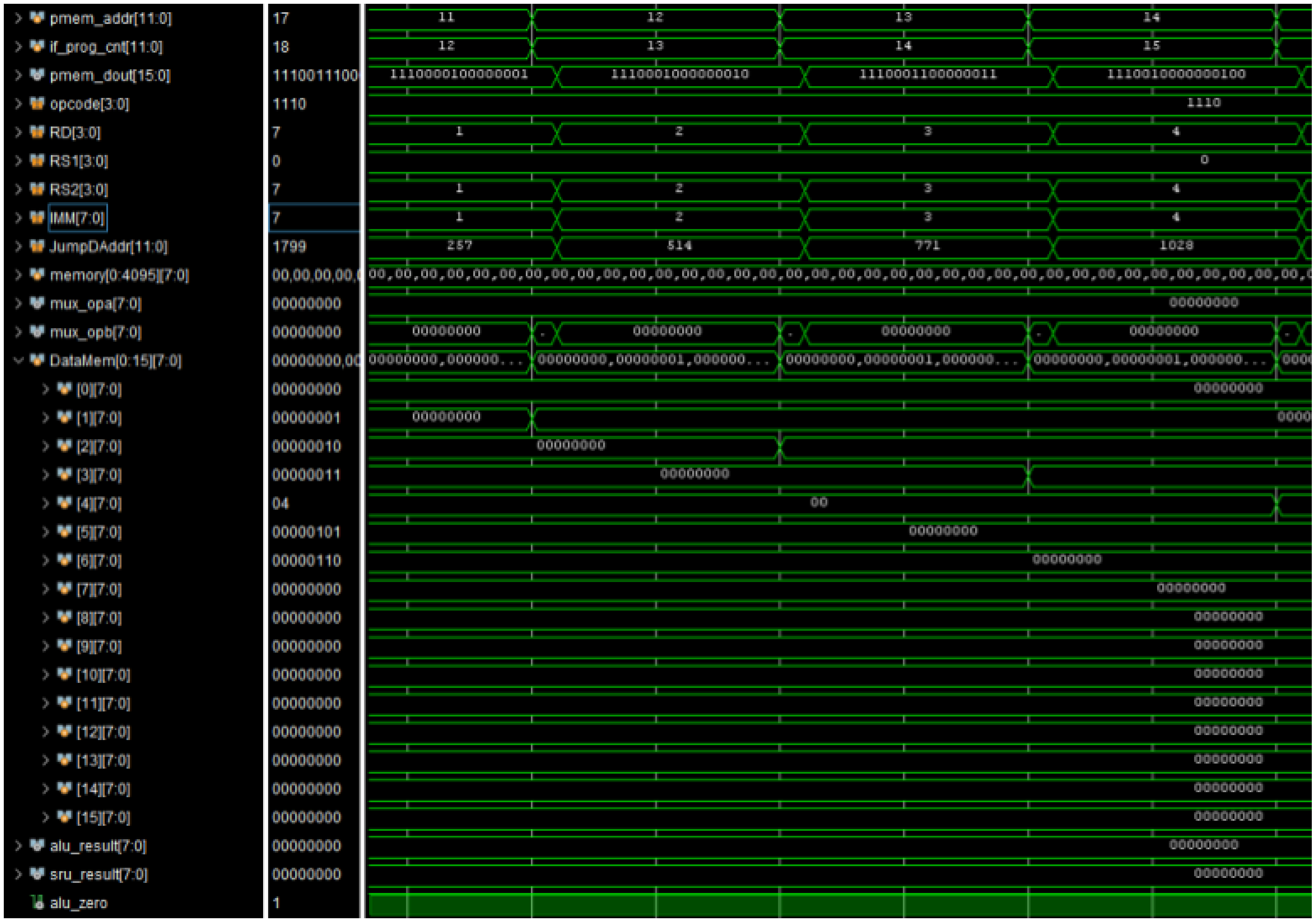


Figure 2: Screenshot of LDI and NOR functionalities

Command	OPCODE (4’b)	RD (Decimal)	RS1 (Decimal)	RS2 (Decimal)	IMM/ADDR (Decimal)	Expected Output (Hex/Binary)	Expected Destination (R0-15)	Actual Output (Hex/Binary)	Actual Destination (R0-15)
NOR	0001	0	0	0	n/a	FF	R0	FF	R0
			8’b0	8’b0					
LDI	1110	2	n/a	n/a	2	2	R2	2	R2

NAND	0000	1	1	2	n/a				
			8’b00000001	8’b00000010		FF	R1	FF	R1
XOR	0010	3	3	4	n/a	7	R3	7	R3
			8’b00000011	8’b00000100					
ADD	0011	4	4	5	n/a	9	R4	9	R4
ROTL	0100	5	5	n/a	n/a	8’b00001010	R5	8’b00001010	R5
			8’b00001010						
SLA	0101	6	6	n/a	n/a	8’b00001100	R6	8’b00001100	R6
			8’b00000110						
SRA	0110	6	6	n/a	n/a	8’b00000110	R6	8’b00000110	R6
			8’b00001100						
SRL	0111	6	6	n/a	n/a	8’b00000011	R6	8’b00000011	R6
			8’b00000110						

Table 1: SRU/ALU validation

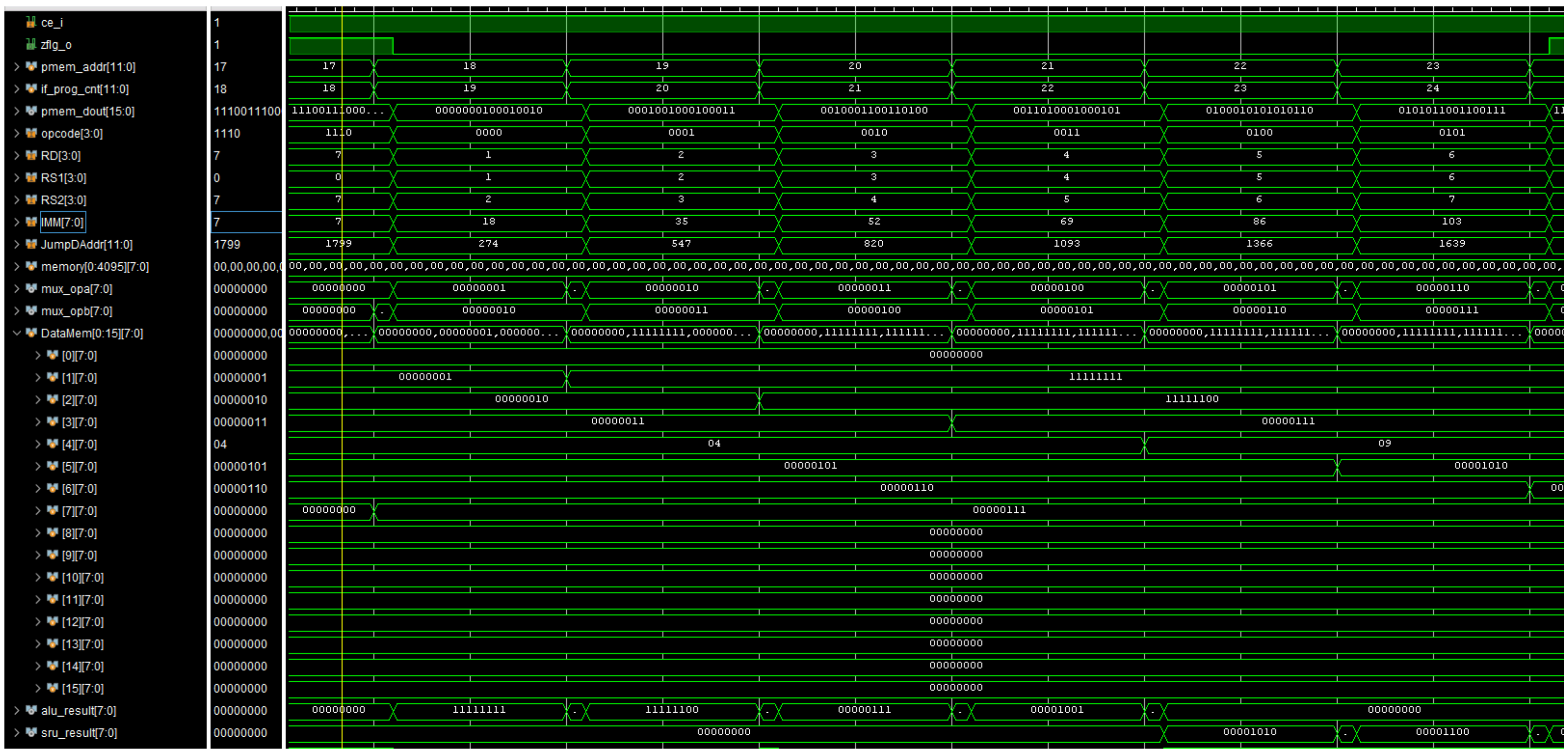


Figure 3: Screenshot of ALU, SLA, and ROTL working

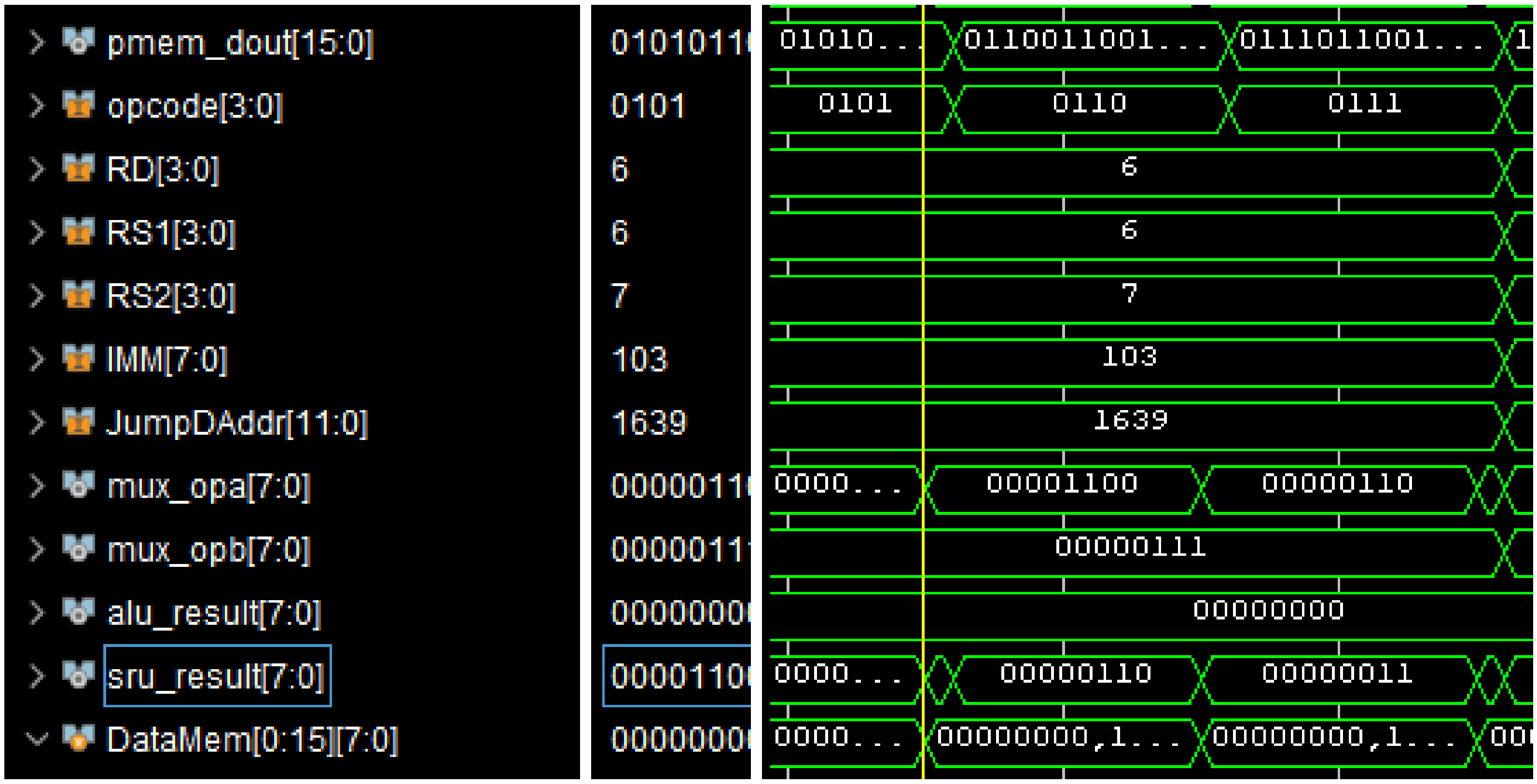


Figure 4: Screenshot of SRA/SRL working

MEMW/MEMR

MEMW/MEMR is implemented through a BRAM module in EX. EX grabs the relevant data from the requested register and stores it to a memory location specified by the value of another register (MEMW). Also, EX can grab data from storage given an address and a destination register. Below is a screenshot showing two times when MEMW/MEMR was called. In the first instance FC (hex) is stored to memory address 0 (from register 10). It is then read from memory and stored into register 0. This same cycle occurs for F4 further into program memory.

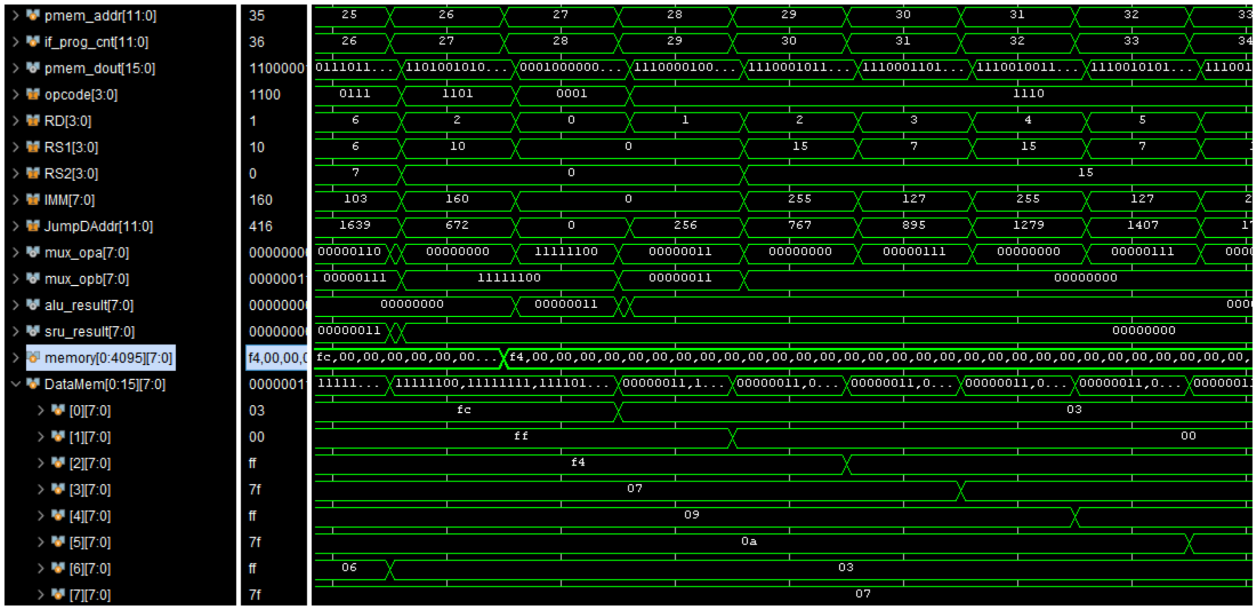


Figure 5: Screenshot of MEMW/MEMR

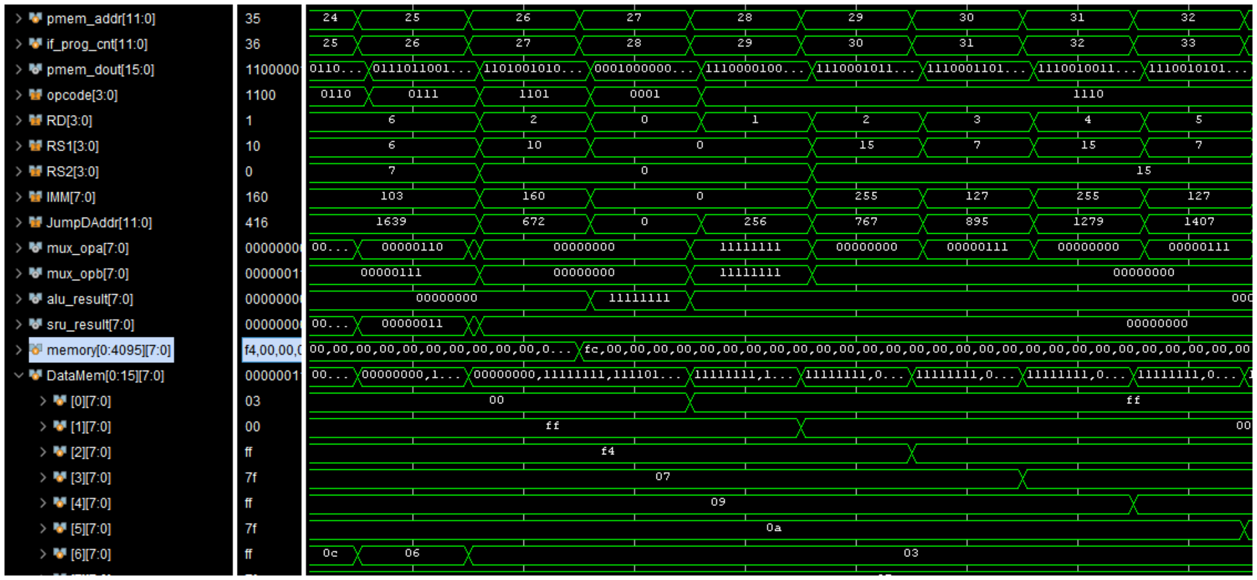


Figure 6: Screenshot of MEMW/MEMR

JMPD/JMPC

Conditional jump and jump direct both work as expected. The figure below shows an immediate jump to program memory 128 when given the JMPD command with an immediate value of 128.

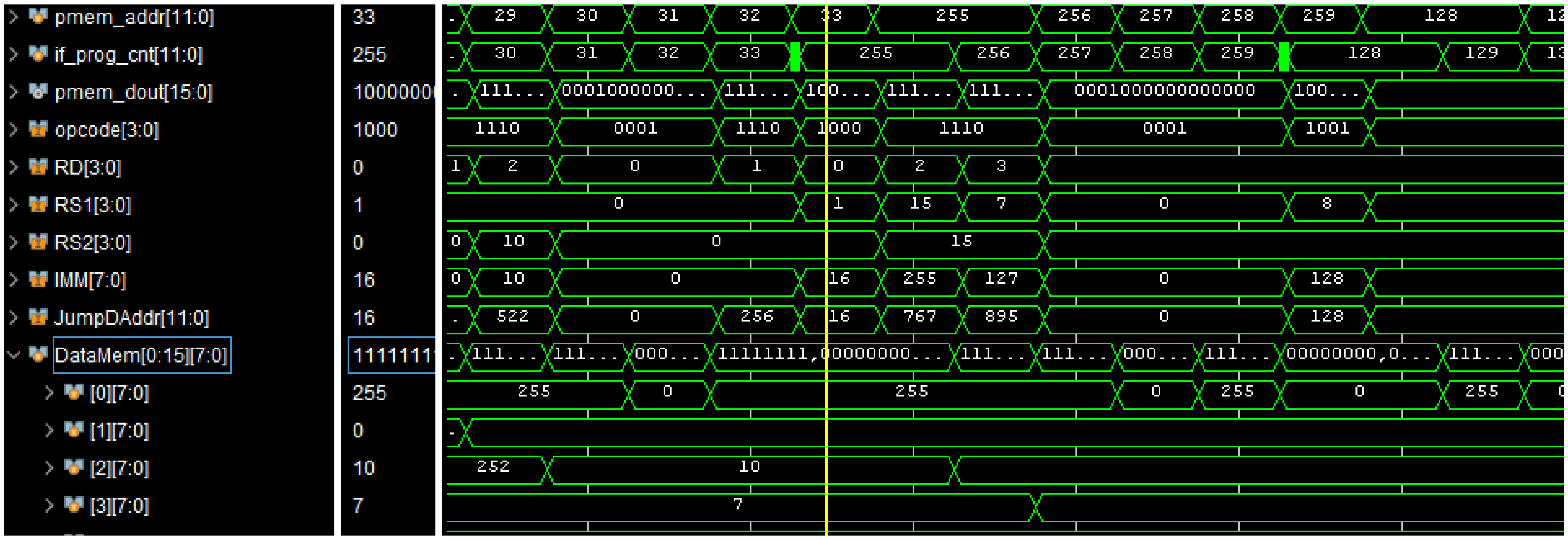


Figure 7: Screenshot of JMPC functioning

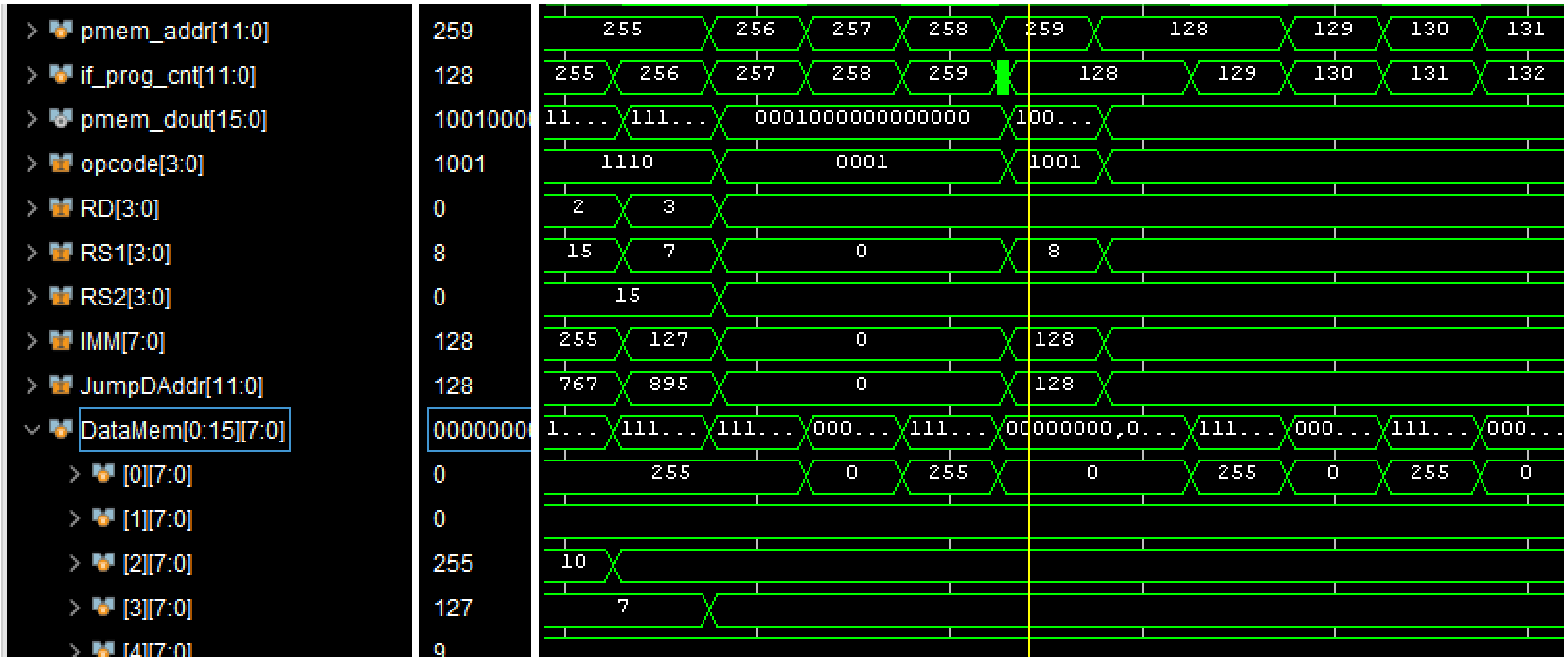


Figure 8: Screenshot of JMPD functioning

Unresolved Issues

Aside from the issues discussed below, the microprocessor should meet all requirements set by the assignment handout.

The Stack

The largest piece that is missing from my implementation is the stack for pushing and popping the program count when jumping to and from subroutines. I was able to implement the memory module and start to save some things, but I was having issues trying to get the stack completely functioning. Below is a screenshot of the stack successfully looping indefinitely. The processor grabs the PC and successfully pushes it to the stack, but never successfully pops the stack, returning from the subroutine. Also, this first push is not saved- exact reason for either of these scenarios is unknown.

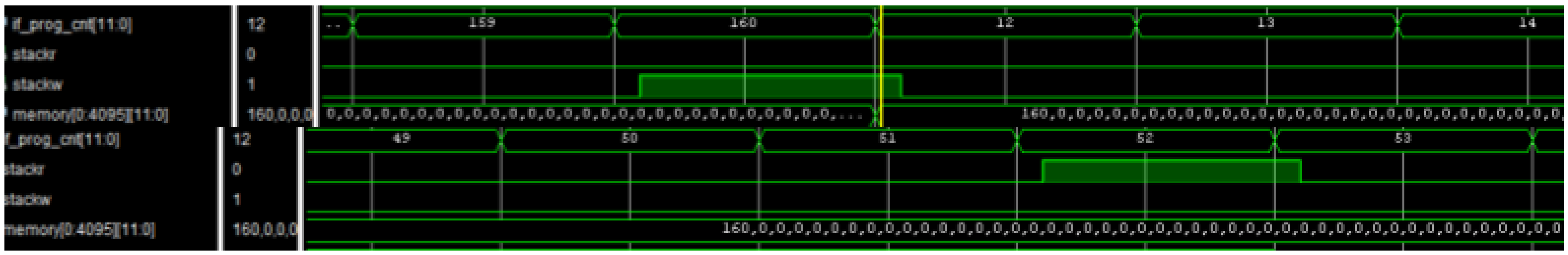


Figure 9: Screenshot of Stack looping. stackr/stackw notes when the stack should pop/push.

Data Storage

Similar to the stack there was only one issue with the BRAM implementation for data storage. The first write to memory does not actually save anything to the memory. Here again, I was not able to find a solution for this or ultimately find the issue.

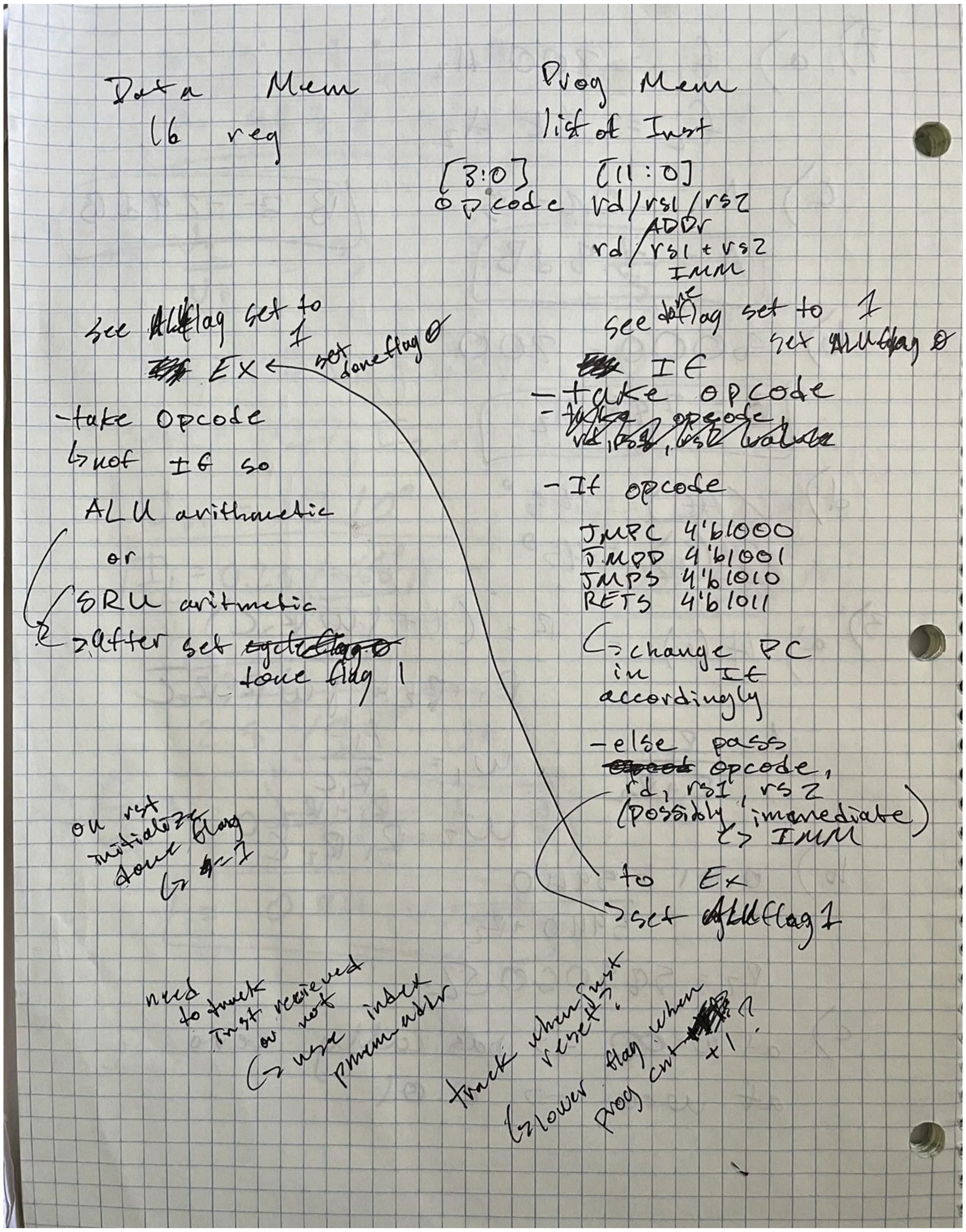
No-Operation

While creating the modules, I used materials from the week 7 slides. These materials did not include the no-operation op-code so it was not initially included in my design. In the end, I could not get the no op functionality implemented.

Synthesis/Clock Speed

Another neglect on my part was that the implementation had to pass synthesis. I did not check this until the end when it was too late. Because of this I also do not know the clock speed of the processor.

Note Appendix



Data Mem
16 reg

see ~~ALU~~flag set to 1
~~[illegible]~~ EX ← set doneflag 0

-take Opcode
↳not IF so
ALU arithmetic
or
SRU aritmetic
↳after set ~~cycleflag 0~~
done flag 1

on rst
initialize
done flag
↳ ~~0~~=1

need to track
Inst received or not
↳ use index
pmem addr

Prog Mem
list of Inst

[3:0] [11:0]
opcode rd/rs1/rs2
ADDr
rd/rs1 + rs2
IMM

see ~~do~~doneflag set to 1
set ALUflag 0

~~[illegible]~~ IF
-take opcode
-~~take opcode~~
~~rd, rs1, rs2 value~~

-If opcode

JMPC 4'b1000
JMPD 4'b1001
JMPS 4'b1010
RETS 4'b1011

↳change PC
in IF
accordingly

-else pass
~~opcod~~ opcode,
rd, rs1, rs2
(possibly immediate)
↳ IMM
to Ex
↳set ALUflag 1

track when inst
reset?

↳lower flag when
prog cnt ~~[illegible]~~ ×1?