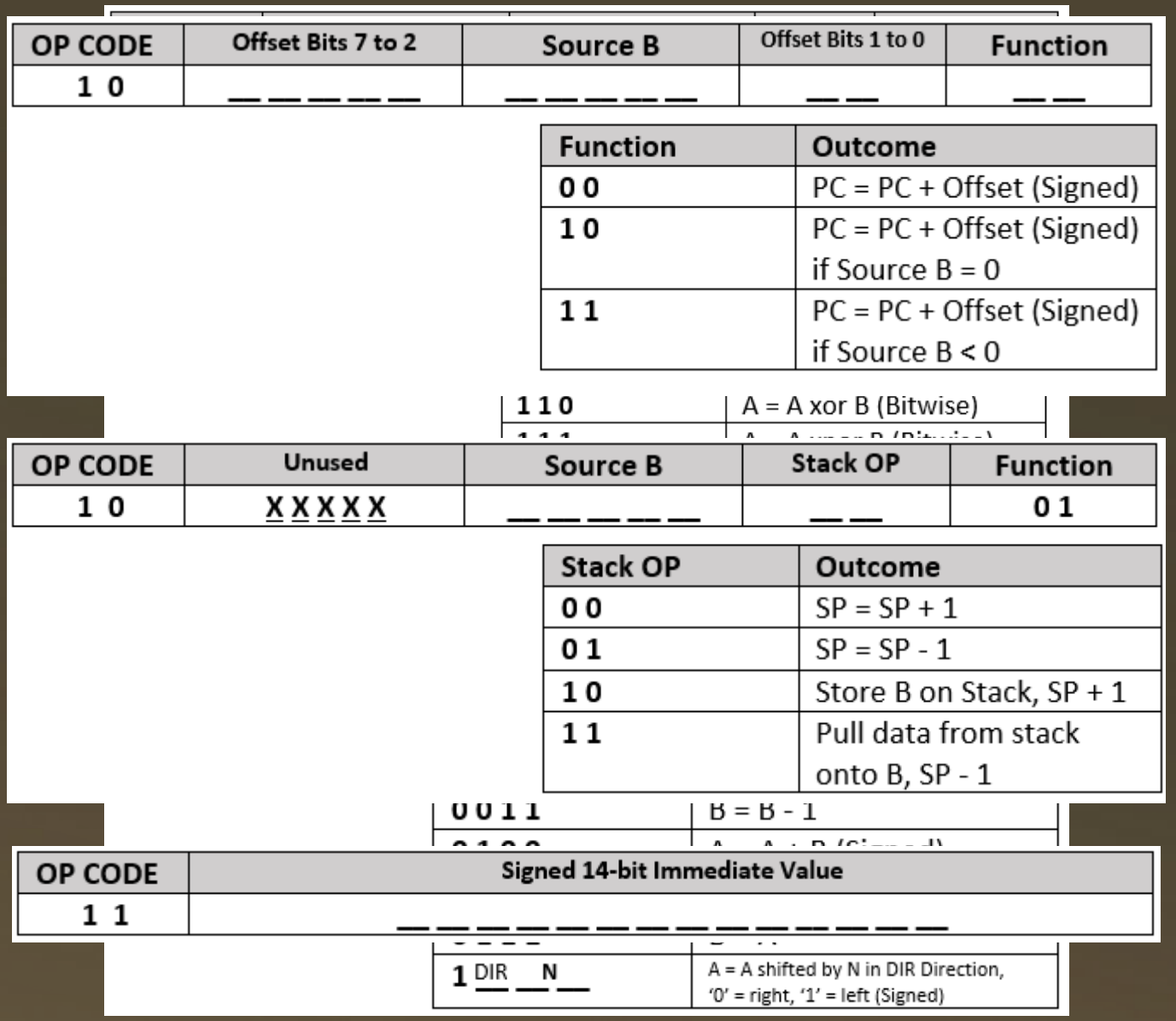
ECE-4710 Final Project

Professor Daniel Llamocca

Submitted by:
Connor Goetz
Ethan Postma
Matthew Hait

Introduction

- 16-bit machine code, multicycle microprocessor circuit via FPGA implementation
- 31 accessible registers
  - 2 utilized for specific purposes
  - 1 indexed (Register 1 address 0x01)
- Included program and data memories (using blockRAM)
- Four basic machine code categories



OP CODE	Offset Bits 7 to 2	Source B	Offset Bits 1 to 0	Function
1 0	_ _ _ _ _ _	_ _ _ _ _ _	_ _	_ _

Function	Outcome
0 0	PC = PC + Offset (Signed)
1 0	PC = PC + Offset (Signed) if Source B = 0
1 1	PC = PC + Offset (Signed) if Source B < 0

1 1 0	A = A xor B (Bitwise)

OP CODE	Unused	Source B	Stack OP	Function
1 0	X X X X X	_ _ _ _ _ _	_ _	0 1

Stack OP	Outcome
0 0	SP = SP + 1
0 1	SP = SP - 1
1 0	Store B on Stack, SP + 1
1 1	Pull data from stack onto B, SP - 1

0 0 1 1	B = B - 1

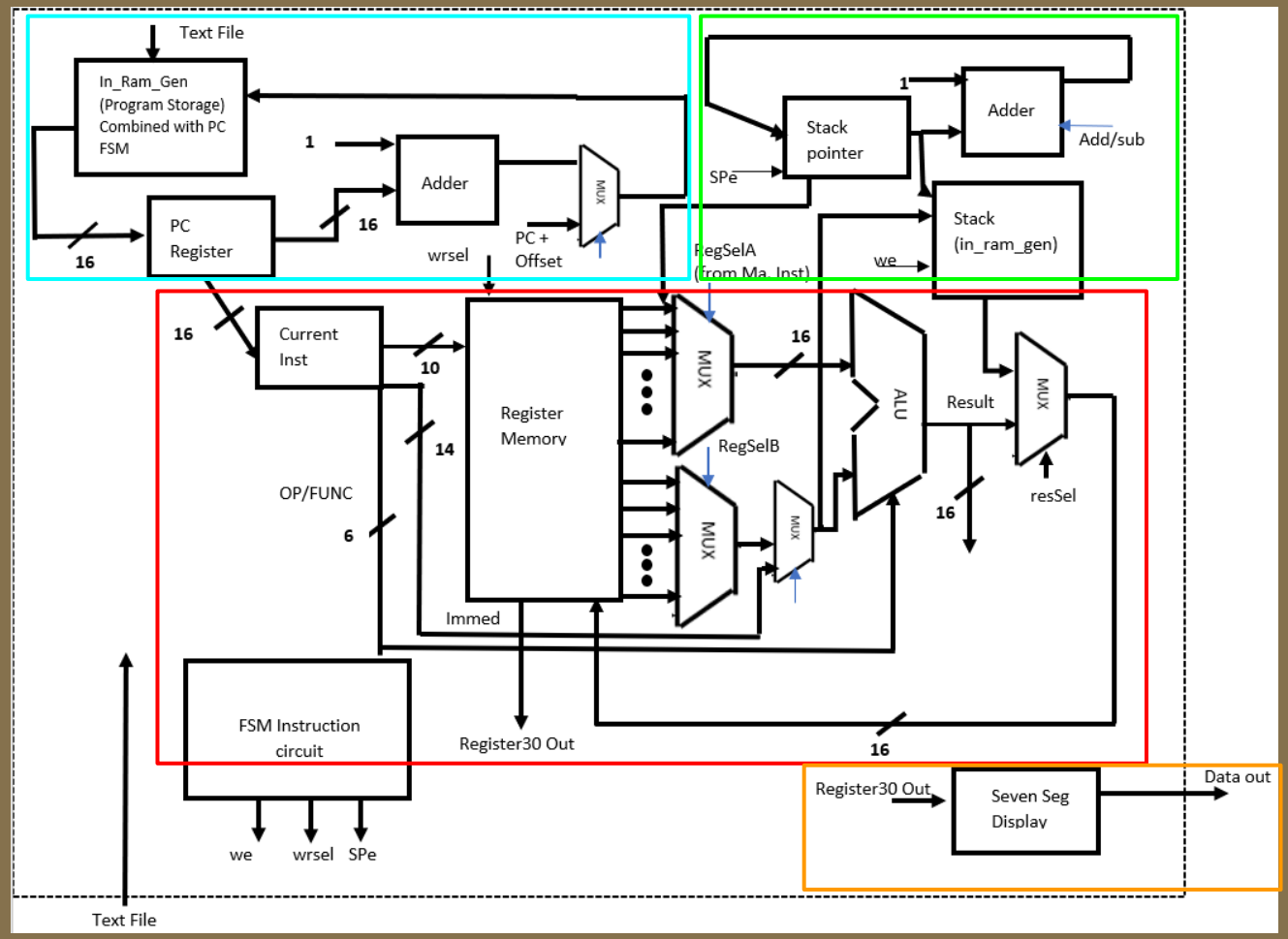
OP CODE	Signed 14-bit Immediate Value
1 1	_ _ _ _ _ _ _ _ _ _ _ _ _ _

1 DIR N _ _ _	A = A shifted by N in DIR Direction, '0' = right, '1' = left (Signed)

OAKLAND UNIVERSITY

Block Overview

- 4 Basic Blocks:
  - Data Path
  - Program Counter
  - Data Memory
  - Serializer
- 2 Finite State Machines
  - Program Counter
  - Data Path
- Inherent FSM and blockRAM reads cause delays
  - Multi-cycle Processor as a result
- User input is machine code .txt file
- Output is displayed on seven segment displays
  - Register 30 (Address 0x1E) reserved for output data



OAKLAND UNIVERSITY

Data Path

Register Memory:

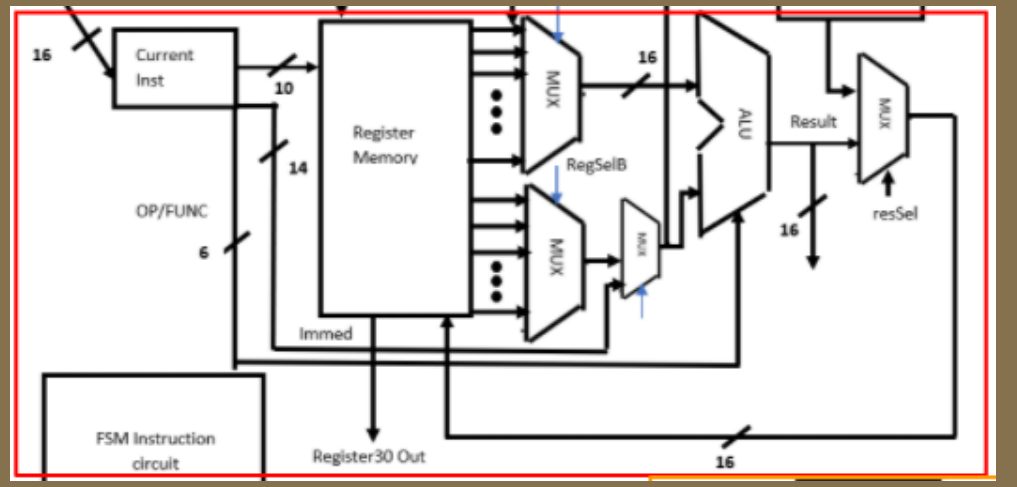
- 31 Registers, my_rege

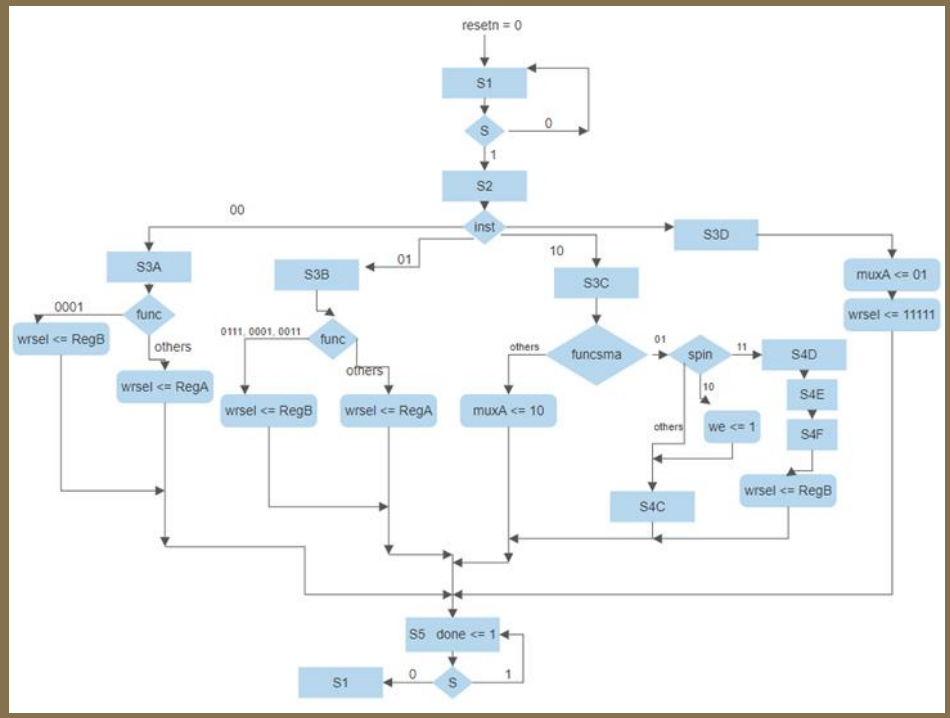
ALU:

- Takes instructions directly from machine code
- Ability to perform arithmetic, logic, and branch checking operations
  - Emits asynchronous “branch_check” signal based upon current instruction (check for negative or zero)
  - Made up of adders, logic statements, and a bit-shifter
  - Purely combinatorial circuit, i.e. all possible outcomes calculated each cycle
  - Calculations asynchronous at cost of extra hardware

Finite State Machine:

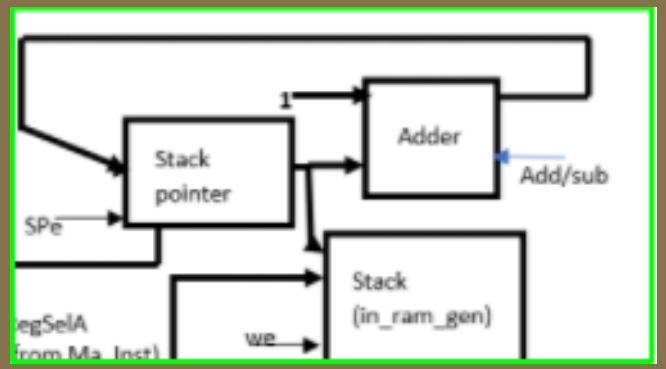
- Controls register enables and write signals
- Controls stack data memory signals
- Multicycle, but simplifies instruction decoding





OAKLAND UNIVERSITY

Data Path - Data Memory



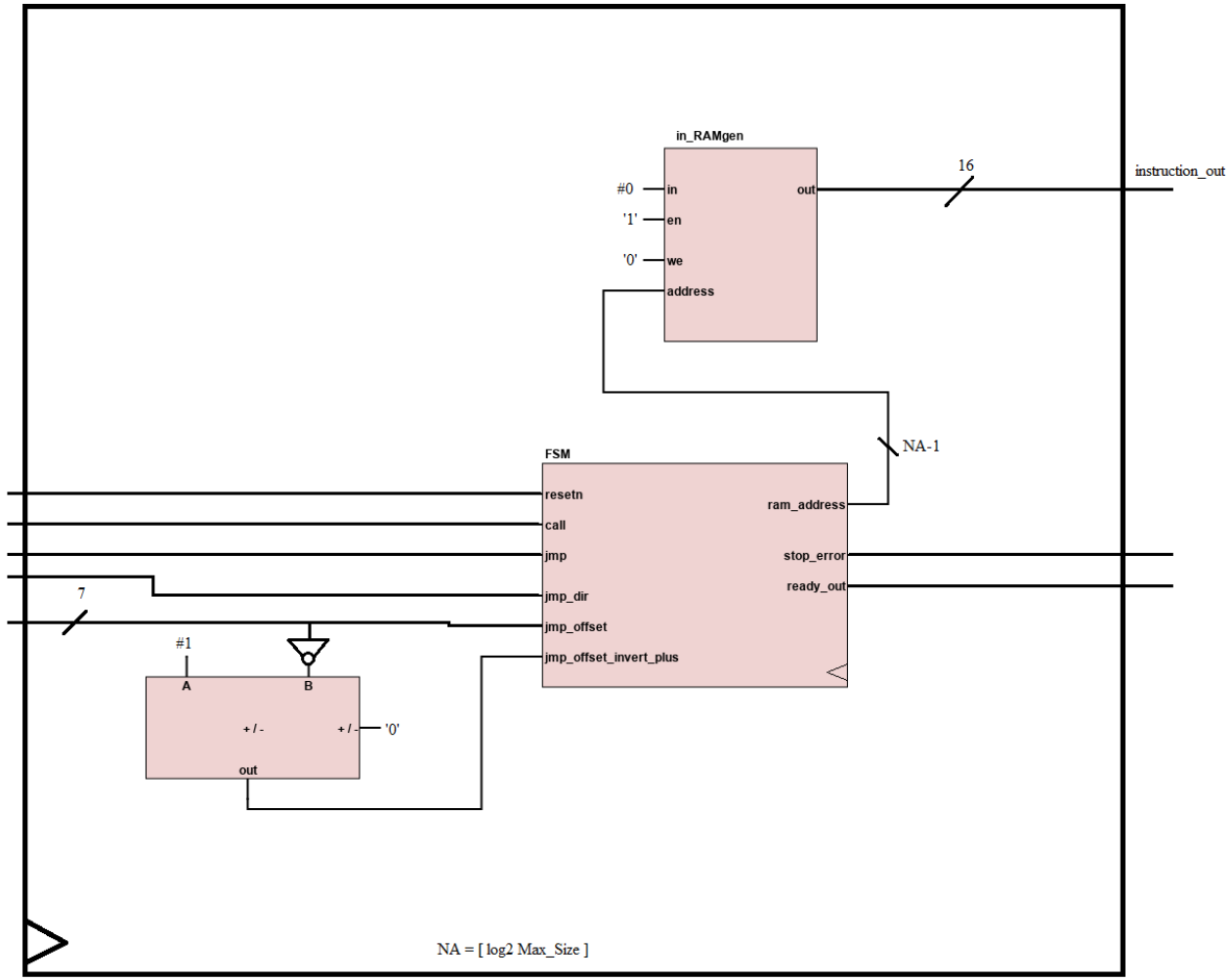
- “Stack” like system
  - User can pull data, push data, increment or decrement pointer
  - Unlike traditional stack, fills top up
  - 16-bit width, same as onboard registers

Stack OP	Outcome
0 0	SP = SP + 1
0 1	SP = SP - 1
1 0	Store B on Stack, SP + 1
1 1	Pull data from stack onto B, SP - 1

- Utilizing the onboard blockRAM
  - Creates a time delay as reading takes extra cycles
  - Very large space relative to what can be accomplished with a ram emulator
  - Shares blockRAM with the program file
  - To prevent data overwrites, begins at address 0x2000

```
--RAM
stack: in_RAMgen generic map(nrows => 65536, ncols => 1, FILE_IMG => "myinival.txt", INIT_VALU
           port map(clock => clock, inRam_idata => ALU_B,
                    inRam_add => SP, inRam_we => we, inRam_en => '1',
                    inRAM_odata => outram);
```


OAKLAND UNIVERSITY



Program Counter

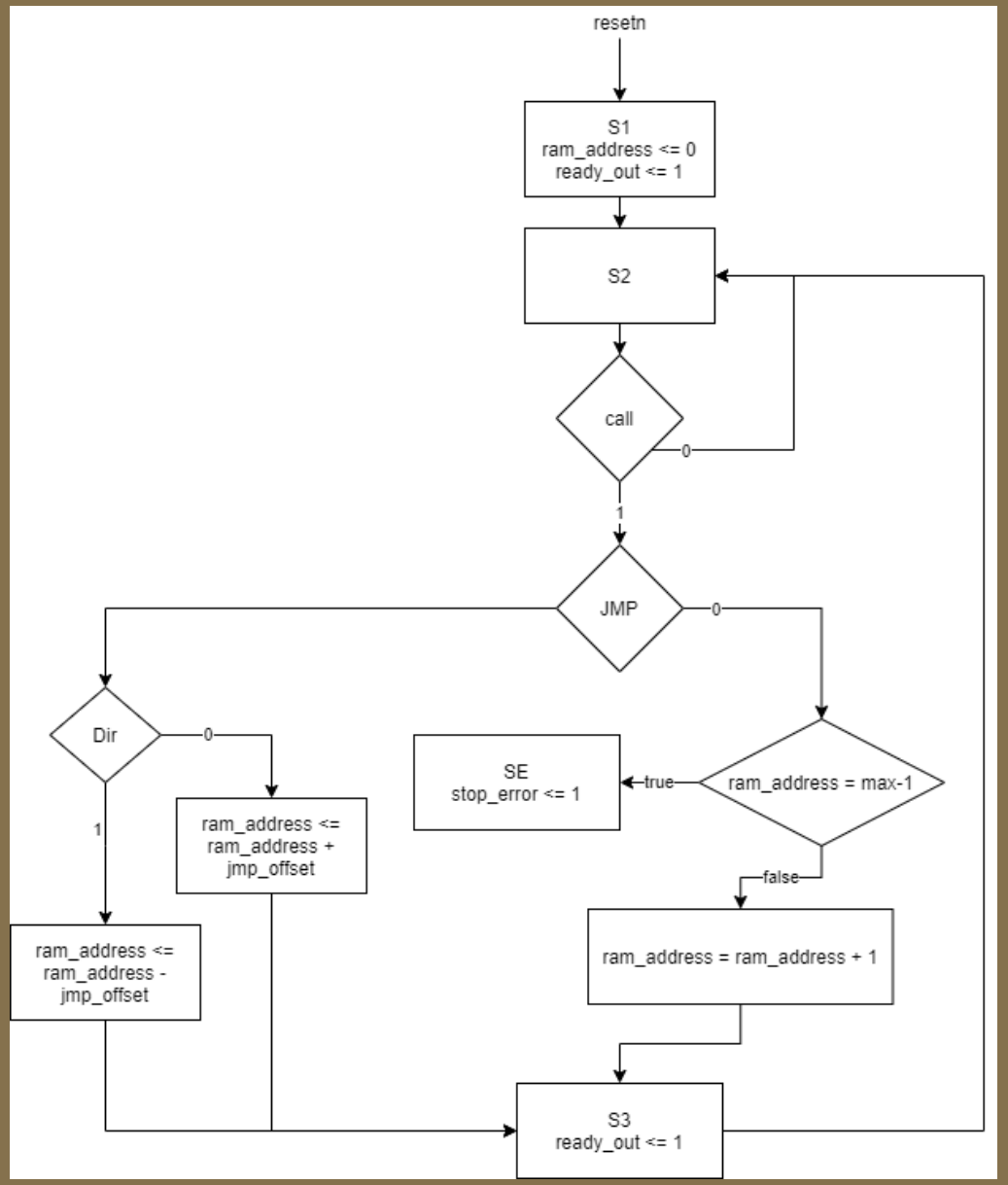
Block Diagram

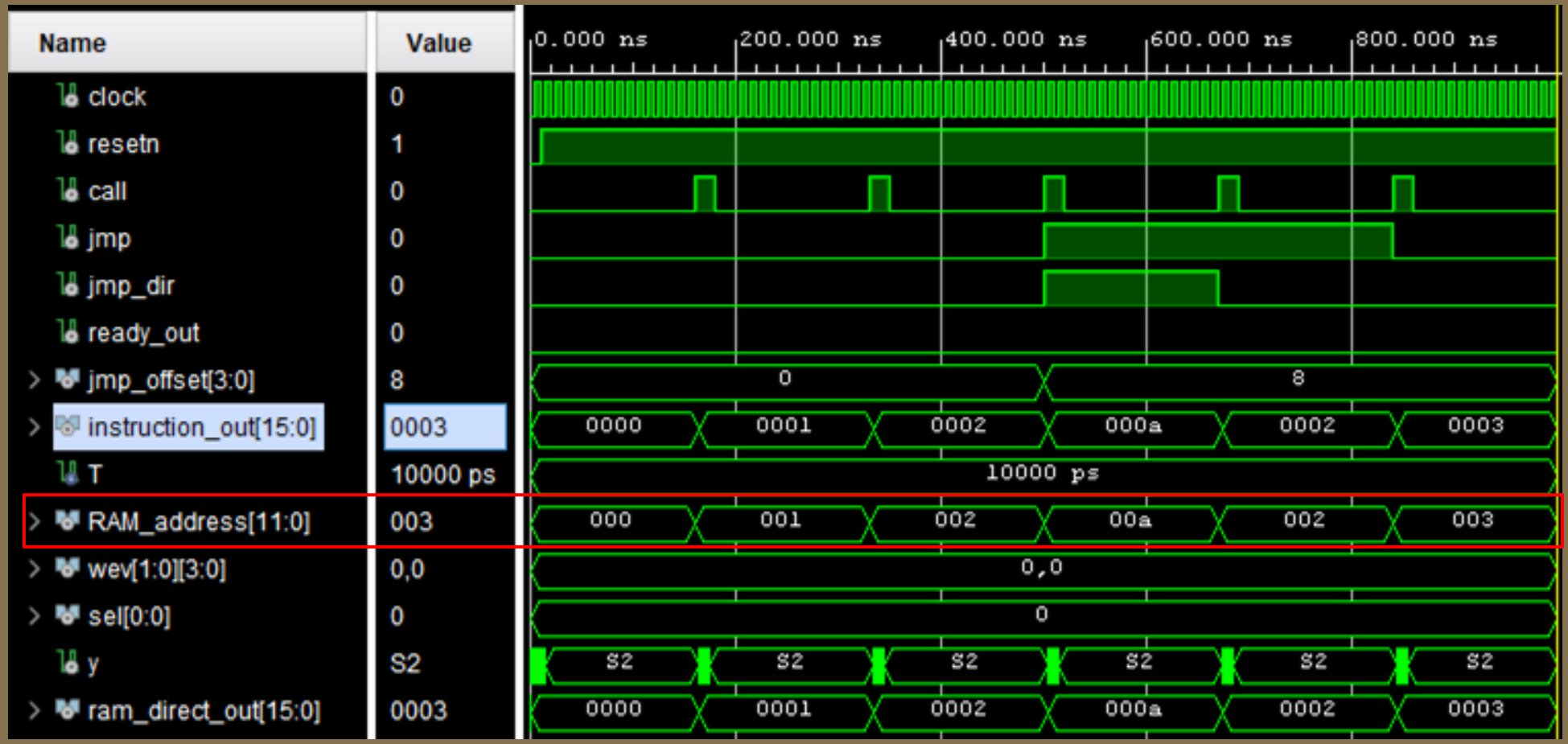
- "Max_Size" is number of stored instruction
- Utilizes blockRAM
- Block *grows* to fit instructions
  - Max_Size changes NA width
  - NA affects RAM allocation
  - NA affects ram_address width

Program Counter

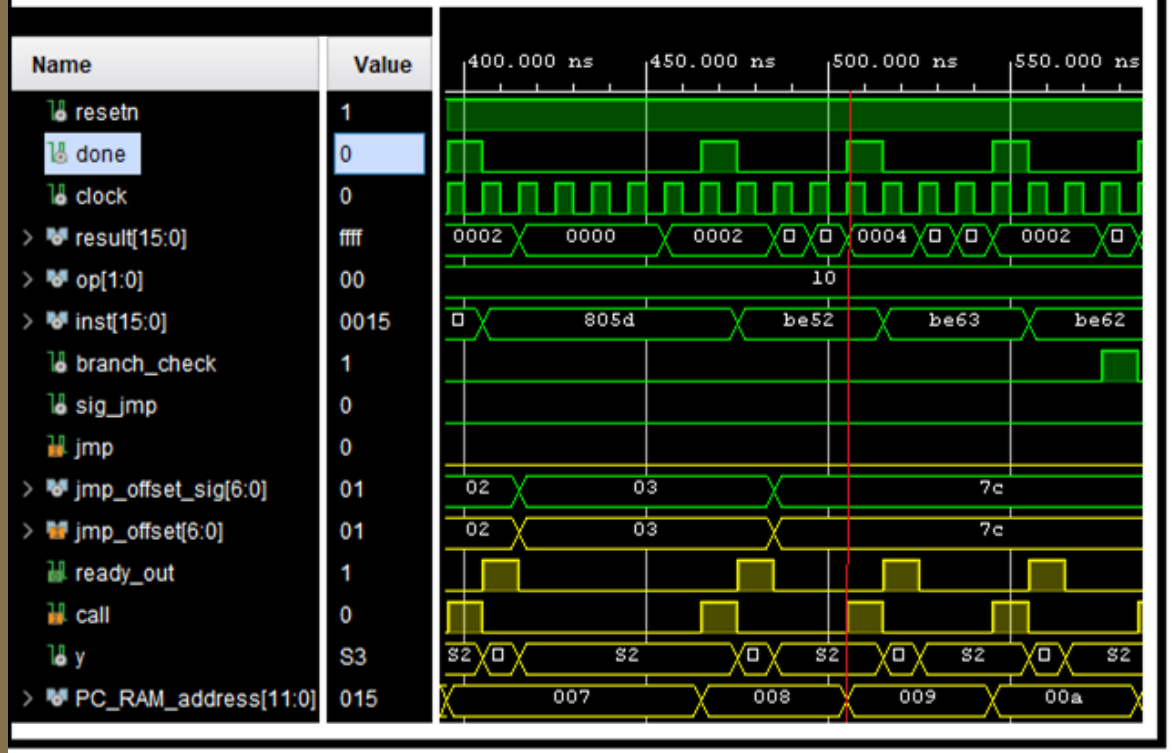
Finite State Machine

- “stop_error” end of instruction indicator
- S3 initializes processor

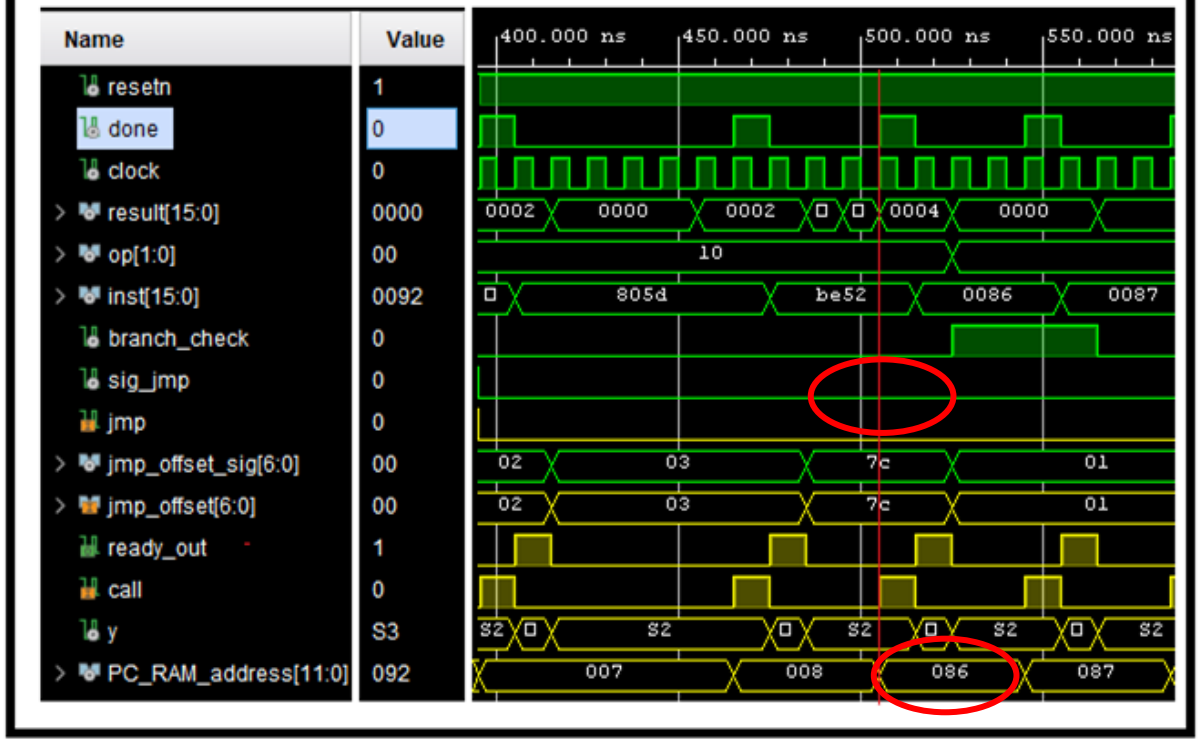



Name
Value
0.000 ns
200.000 ns
400.000 ns
600.000 ns
800.000 ns
clock
0
resetn
1
call
0
jmp
0
jmp_dir
0
ready_out
0
jmp_offset[3:0]
8
0
8
instruction_out[15:0]
0003
0000
0001
0002
000a
0002
0003
T
10000 ps
10000 ps
RAM_address[11:0]
003
000
001
002
00a
002
003
wev[1:0][3:0]
0,0
0,0
sel[0:0]
0
0
y
S2
S2
S2
S2
S2
S2
S2
ram_direct_out[15:0]
0003
0000
0001
0002
000a
0002
0003

```
jmp_check: process (op, funcsma, branch_check)
begin
    sig_jmp <= '0';
    if op = "10" then
        if funcsma = "00" then
            sig_jmp <= '1';
        end if;
        if branch_check = '1' then
            sig_jmp <= '0'; --problem here
        end if;
    end if;
end process;
```




```
jmp_check: process (op, funcsma, branch_check)
begin
    sig_jmp <= '0';
    if op = "10" then
        if funcsma = "00" then
            sig_jmp <= '1';
        end if;
        if branch_check = '1' then
            sig_jmp <= '1'; --problem here
        end if;
    end if;
end process;
```

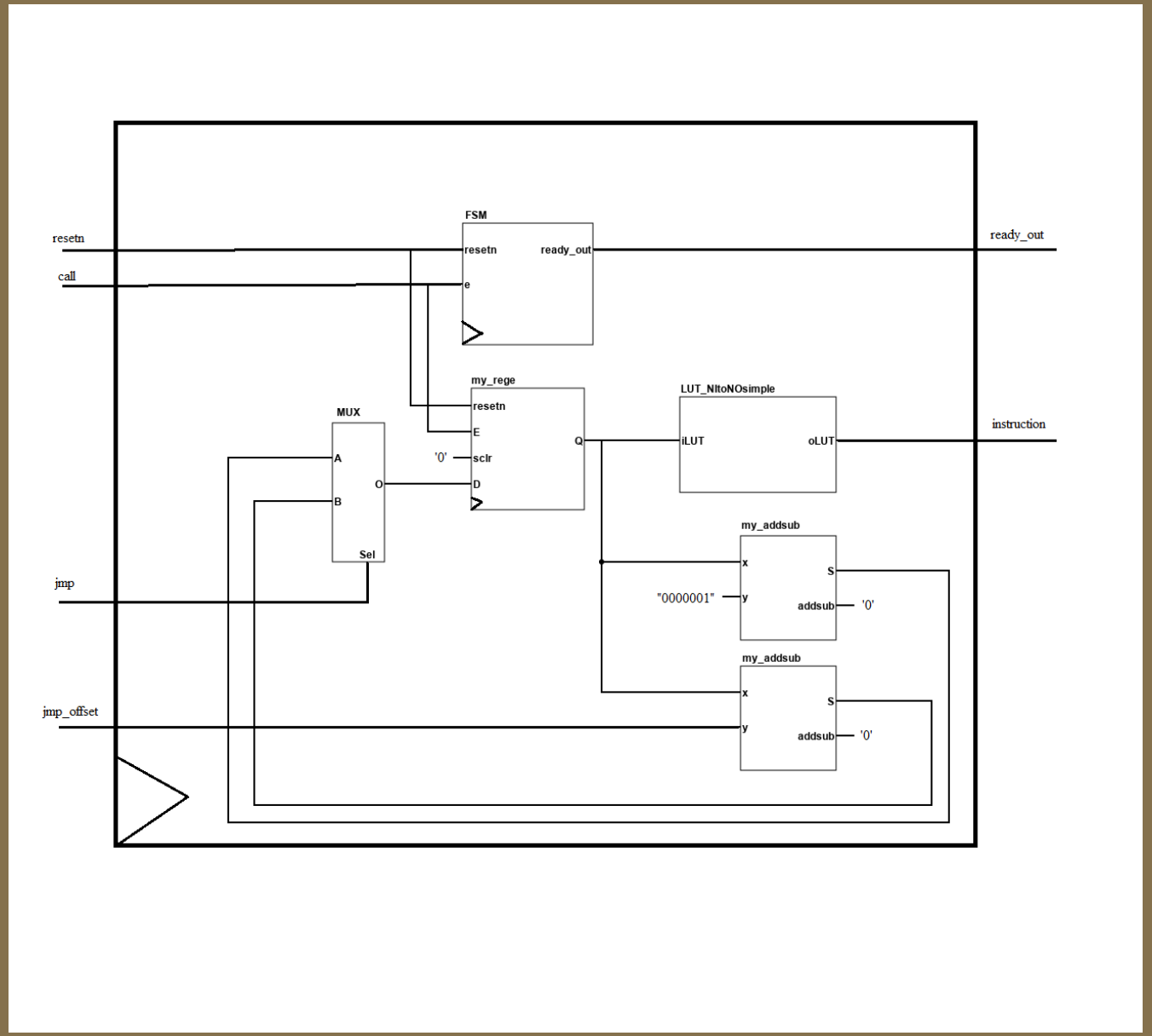


Aside: An extremely small timing error

- “branch_check” flips once after 520 ns
- Flip last for infinitely short period of time
- Resolved by adding clock event check

```
when S2 =>
    -- Added clock check to stop simulation error
    if (clock'event and clock = '1') then
        if call =  '1' then
            if jmp = '1' then
                -- Jump Ram Address
                if jmp_dir = '1' then
                    -- Negative
                    PC_RAM_address <= PC_RAM_address - conv_std_logic_vector (unsigned(jmp_offset_invert_plus),NA-1);
                else
                    -- Positive
                    PC_RAM_address <= PC_RAM_address + conv_std_logic_vector (unsigned(jmp_offset),NA-1);
                end if;
            else
                -- No Jump, index by 1
                PC_RAM_address <= PC_RAM_address + conv_std_logic_vector (1,NA-1);
            end if;
        end if;
    end if;
```

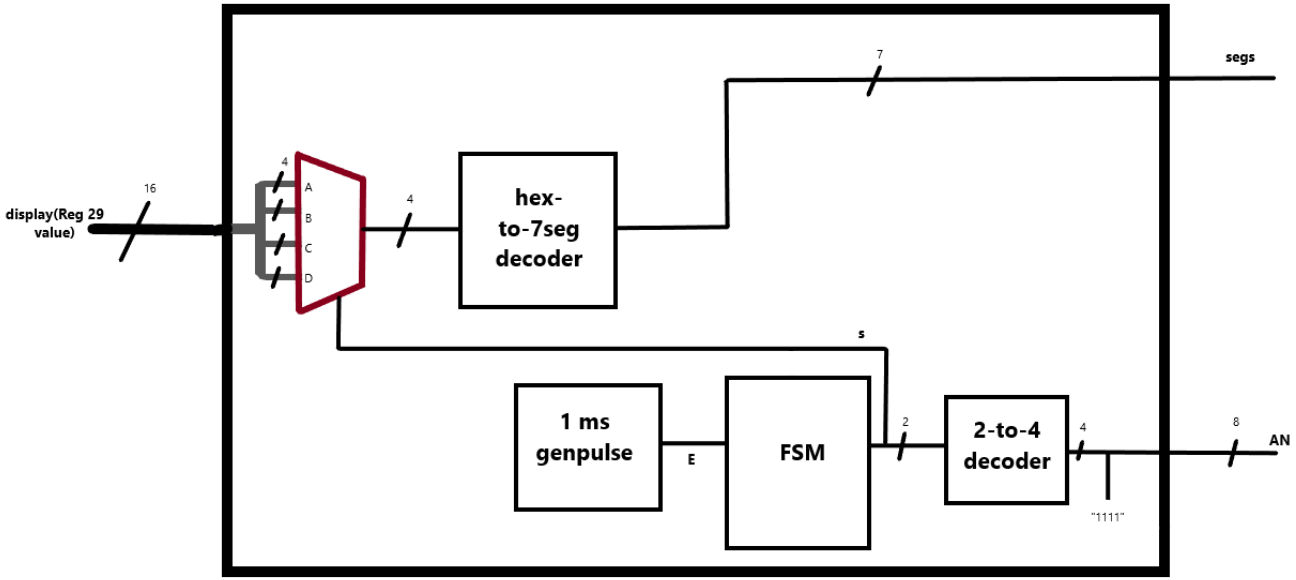
- Original program counter failed in post-synthesis timing
  - Unmapped synthesized ram signals
  - Possibly result of using two in_RAMgen
- Revised PC uses LUT
  - Loads from text file



OAKLAND UNIVERSITY

Serializer

Takes constant output from register 29, the “output” register



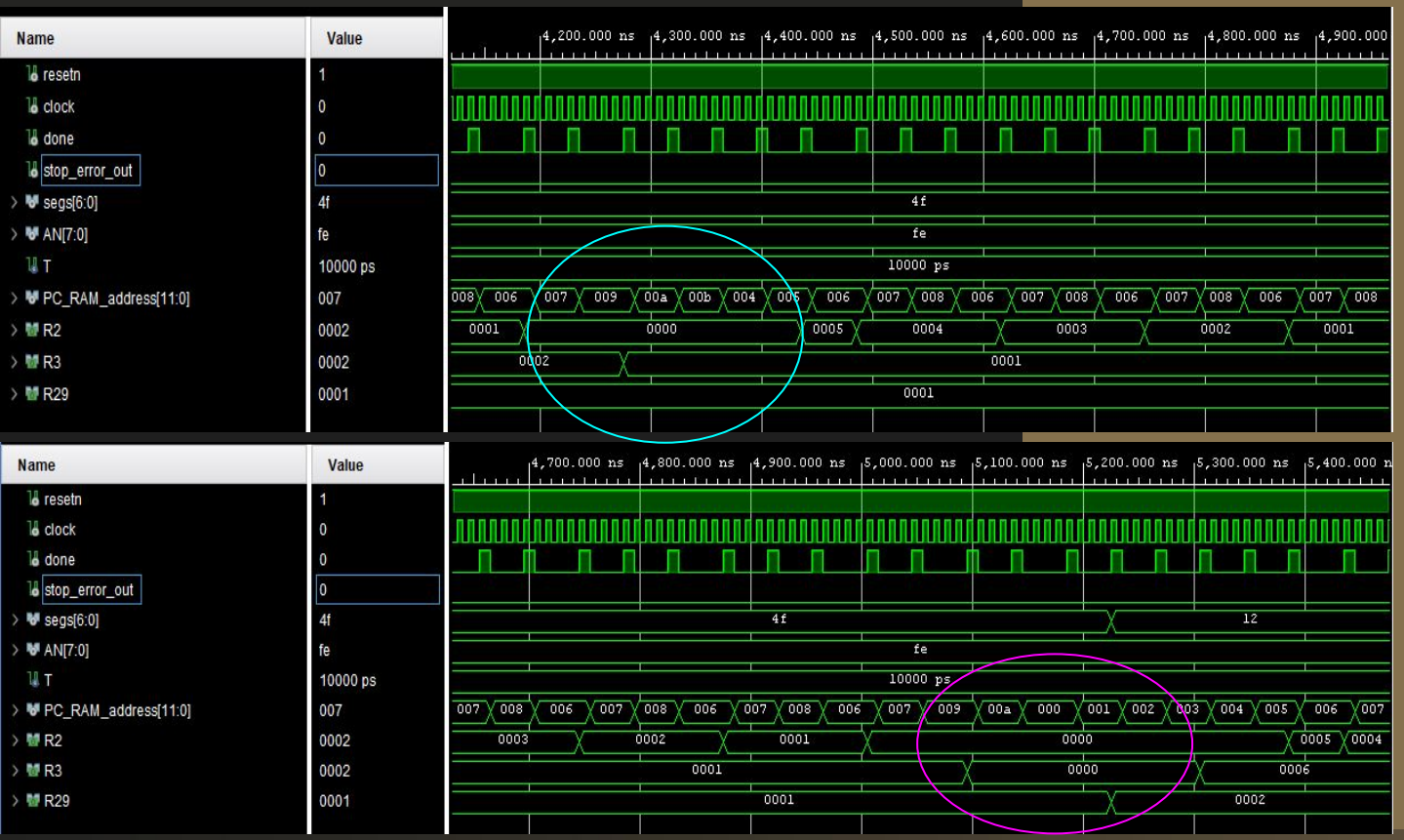
Basic parametric code, important to the real life applications of our microprocessor

Machine Code Instructions

Table shows the different machine code inputs for the functions in our microprocessor

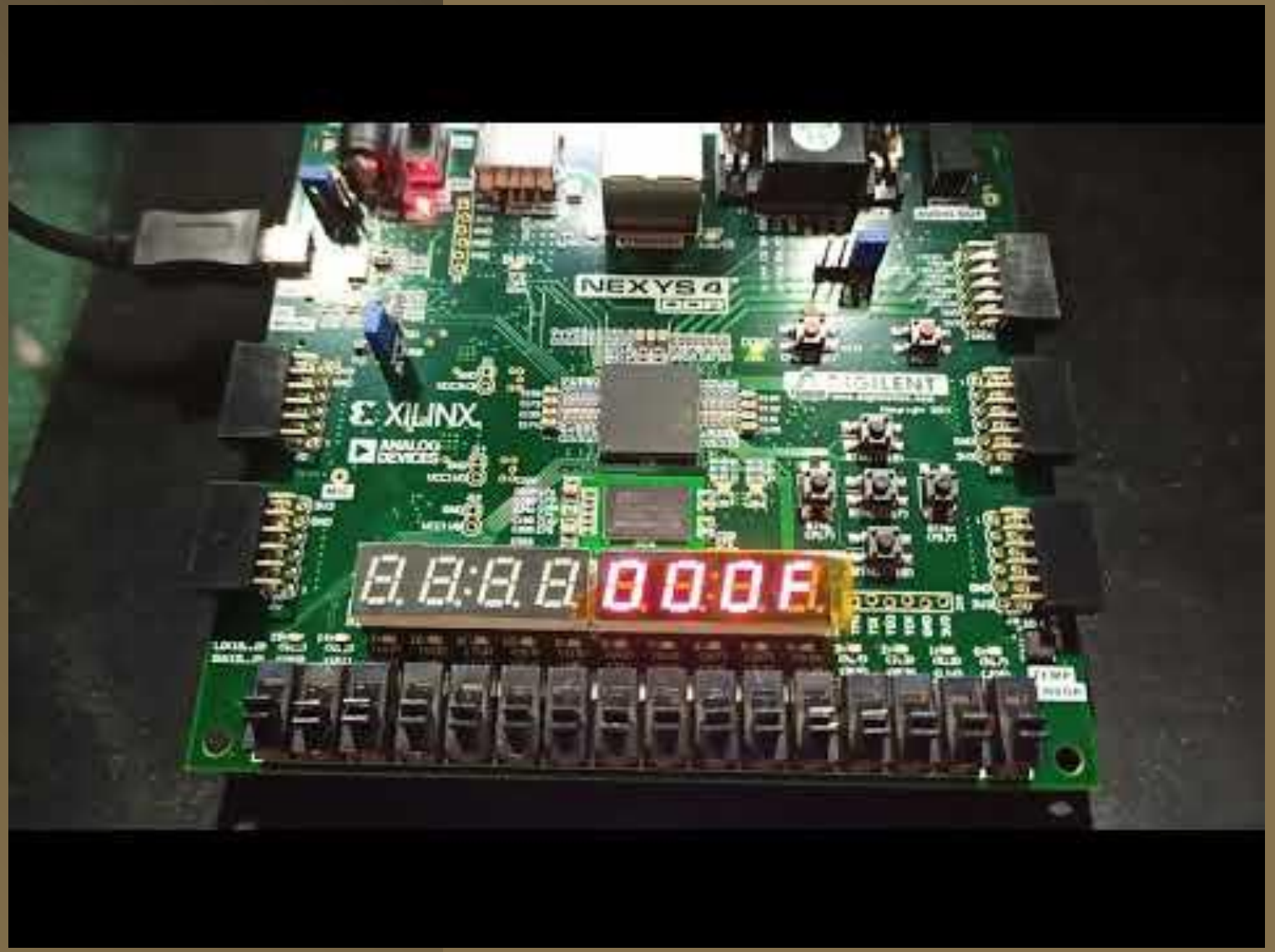
Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Instruction
Logic:	0	0	Destination Register/ Register A					Register B					Function Op Code				
	0	0											x	0	0	0	A = Not A
	0	0											x	0	0	1	B = Not B
	0	0											x	0	1	0	A = A and B
	0	0											x	0	1	1	A = A or B
	0	0											x	1	0	0	A = A nand B
	0	0											x	1	0	1	A = A nor B
	0	0											x	1	1	0	A = A xor B
	0	0											x	1	1	1	A = A xnor B
	0	0															
Arithmatic:	0	1	Destination Register/ Register A					Register B					Function Op Code				
	0	1											0	0	0	0	A = A + 1
	0	1											0	0	0	1	B = B + 1
	0	1											0	0	1	0	A = A - 1
	0	1											0	0	1	1	B = B - 1
	0	1											0	1	0	0	A = A + B
	0	1											0	1	0	1	A = A - B
	0	1											0	1	1	1	B = A
	0	1															
	0	1															
Move/Stack:	1	0	Offset High Bits										Offset Low		OP		
	1	0	dir	off(5)	off(4)	off(3)	off(2)						off(1)	off(0)	0	0	JMP
	1	0	dir	off(5)	off(4)	off(3)	off(2)						off(1)	off(0)	1	0	BRN if B = 0
	1	0	dir	off(5)	off(4)	off(3)	off(2)						off(1)	off(0)	1	1	BRN if B is -
	1	0	x	x	x	x	x						0	0	0	1	SP = SP + 1
	1	0	x	x	x	x	x						0	1	0	1	SP = SP - 1
	1	0	x	x	x	x	x						1	0	0	1	Push to stack
	1	0	x	x	x	x	x						1	1	0	1	Pull from stack
	1	0															
Load IM(R30)	1	1	Load Value														

Test Program




```
4200;    R1 = R1 + 1                          PC: 000
43E7;    R29(output) = R1                PC: 001
C006;    IM = 6                               PC: 002
7E37;    R3 = IM                              PC: 003
C005;    IM = 5                               PC: 004
7E27;    R2 = IM                              PC: 005
4402;    R2 = R2 - 1                          PC: 006
802A;    Branch to 009 if R2 = 0         PC: 007
BE38;    JMP to 006                           PC: 008
4602;    R3 = R3 - 1                          PC: 009
BA3A;    Branch to 000 if R3 = 0         PC: 00A
BC34;    JMP to 004                           PC: 00B
```

In order to demonstrate this counter working on the board, the values input to Reg 2 and Reg 3 have to be changed to create a 1 second delay. Since each instruction takes about 40 ns, FFFF is loaded into R2 and 007F is loaded into R3 to create approximately a 1 second delay, incrementing the value of the output register every 1 second. This calculation takes the timing for the jump instructions into consideration.



https://www.youtube.com/watch?v=DOc6HEfDoSk