Organization of Digital Computer Lab
EECS112L/CSE 132L

Assignment 4
Multi-cycle MIPS Processor

prepared by: Team CART

Student name	**Student ID**
Raquel Fallman	26316814
Arash Nabili	37684183
Christine Srun	15386050
Tyler Stevens	40051117

EECS Department
Henry Samueli School of Engineering
University of California, Irvine

February 13, 2016

Contents

List of Figures

1 Introduction

For this lab, we extended the single-cycle MIPS processor to have a multi-cycle datapath. With a multi-cycle datapath, each instruction is broken up into separate steps. The steps are instruction fetch, instruction decode, execute, read/write data memory, and write back. Each step takes a single clock cycle and a functional unit can be used more than once in an instruction if it is used in a different clock cycle. In comparison to the single-cycle processor, this only has one memory unit, as the instruction memory and data memory are combined to create a single memory unit. The benefits of a multi-cycle datapath is that it reduces the average instruction time and the amount of hardware needed.

2 Processor

The multi-cycle processor merges several blocks with similar functionality used in the single-cycle design, and this merging is made possible by the fact that the blocks can serve different purposes in each clock cycle. Some of the merged components include:

- The ALU and the adders used for incrementing PC and adding branch offset
- The instruction and data memories

One side effect of merging components is the need for additional multiplexers to choose between different values for the merged blocks, as well as registers to store some values that are to be used later. The following are some of the multiplexers and registers needed to accommodate the changes:

- A 4-to-1 multiplexer for choosing the operand B input of the ALU
- A 4-to-1 multiplexer for choosing the next value for the program counter
- Registers to store the current instruction, and data loaded from memory
- Registers to store read data 1 and 2 from the register file
- A register to store the result of the ALU

The processor ports are shown below.

Processor Port Description			
Port name	Port size	Port Type	Description
ref_clk	1	IN	clock signal
reset	1	IN	reset to normal state

2.1 Design Schematic

Below is the design schematic for our implementation of the processor, showing the merged arithmetic units and memories, and newly added multiplexers and registers:

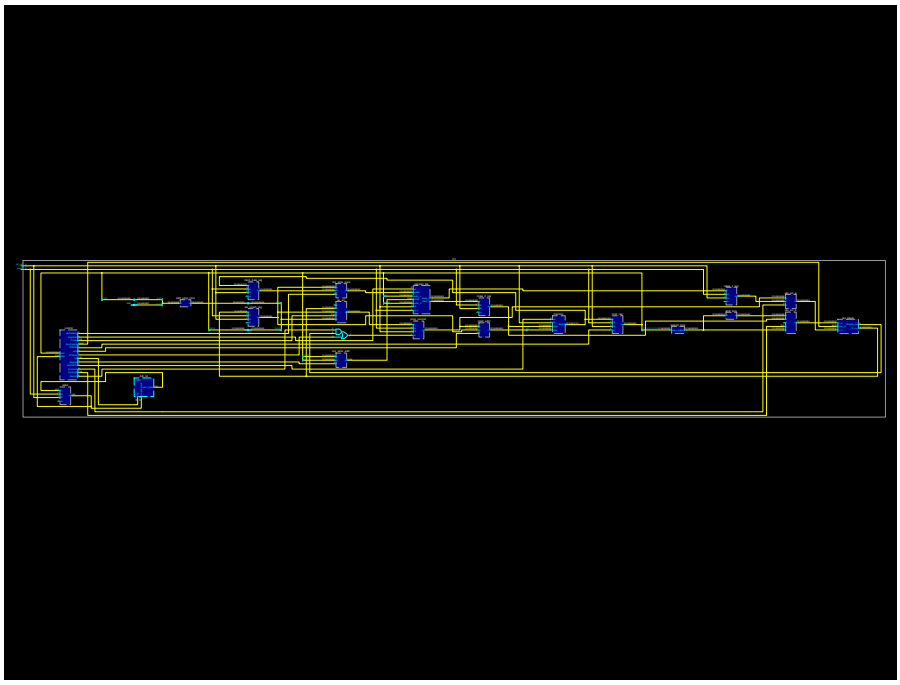


Figure 1: Schematic of Processor.

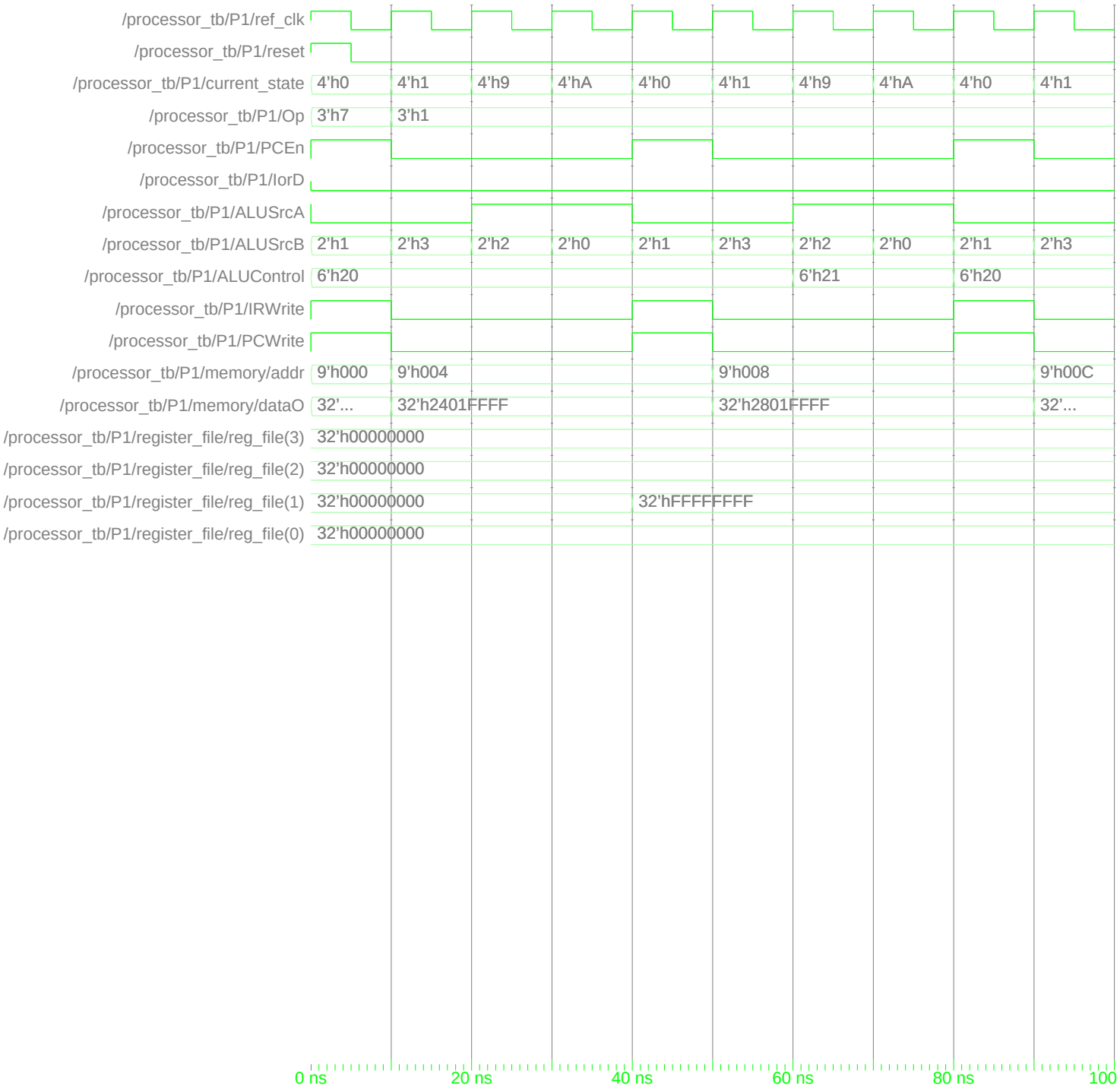
2.2 Simulation

To simulate the processor, we created a testbench and used the provided MIPS code to verify the design. The code is preloaded into the instruction memory. It tests R-type, I-type, J-type, load, and store instructions.

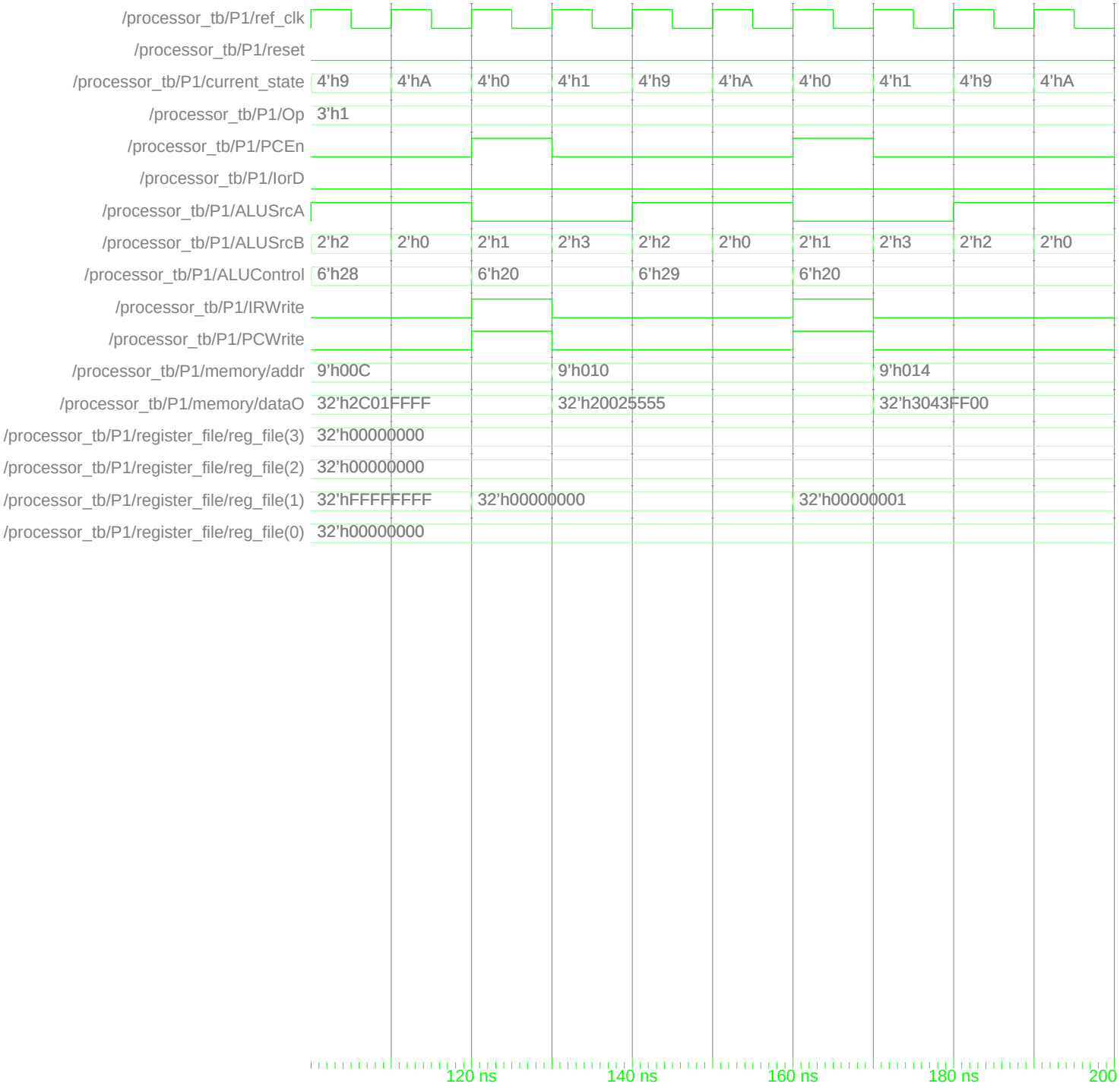
The waveforms were generated from our own test code to make sure that all the operations were working properly.

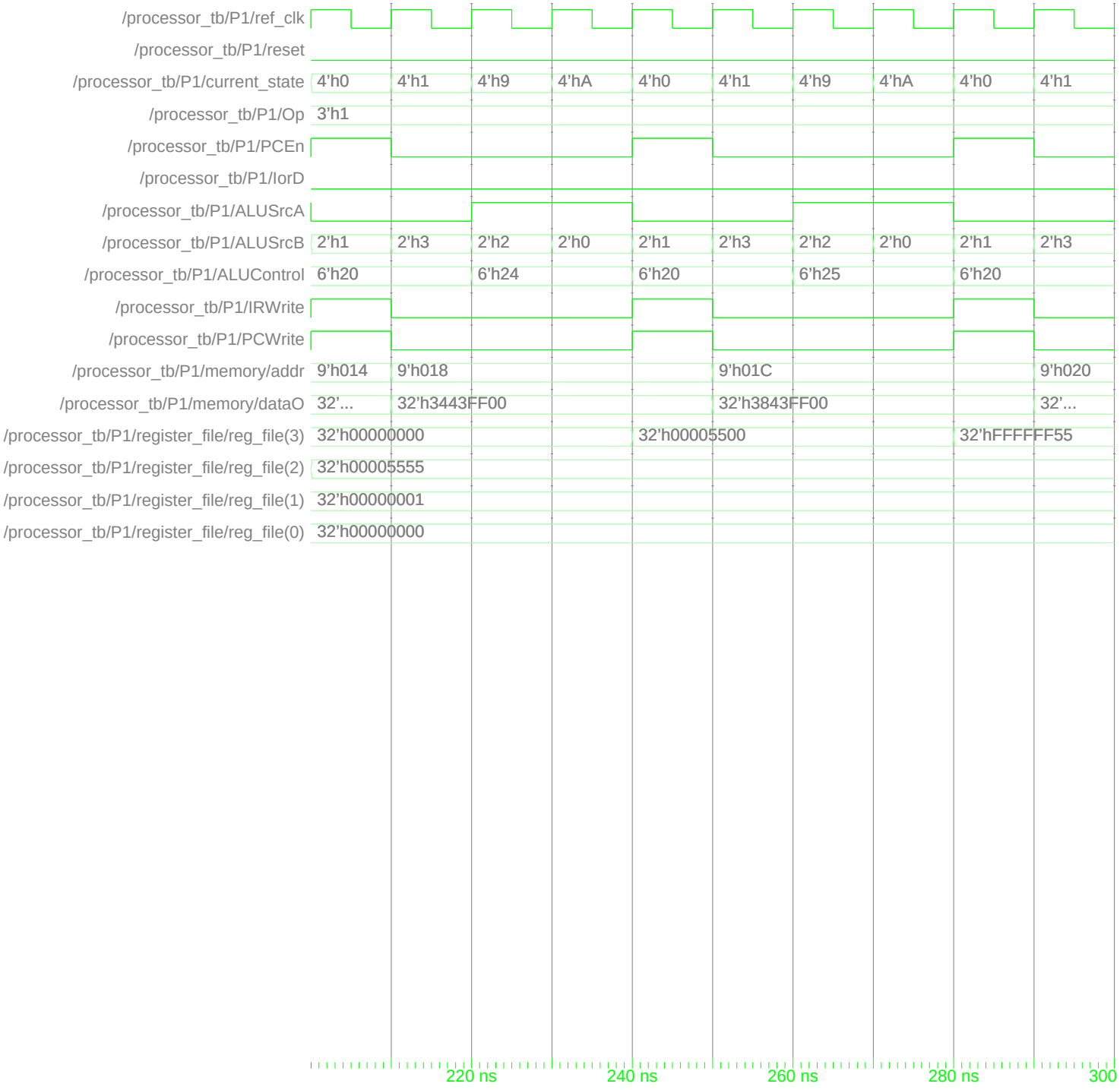
2.2.1 Waveform for test program.

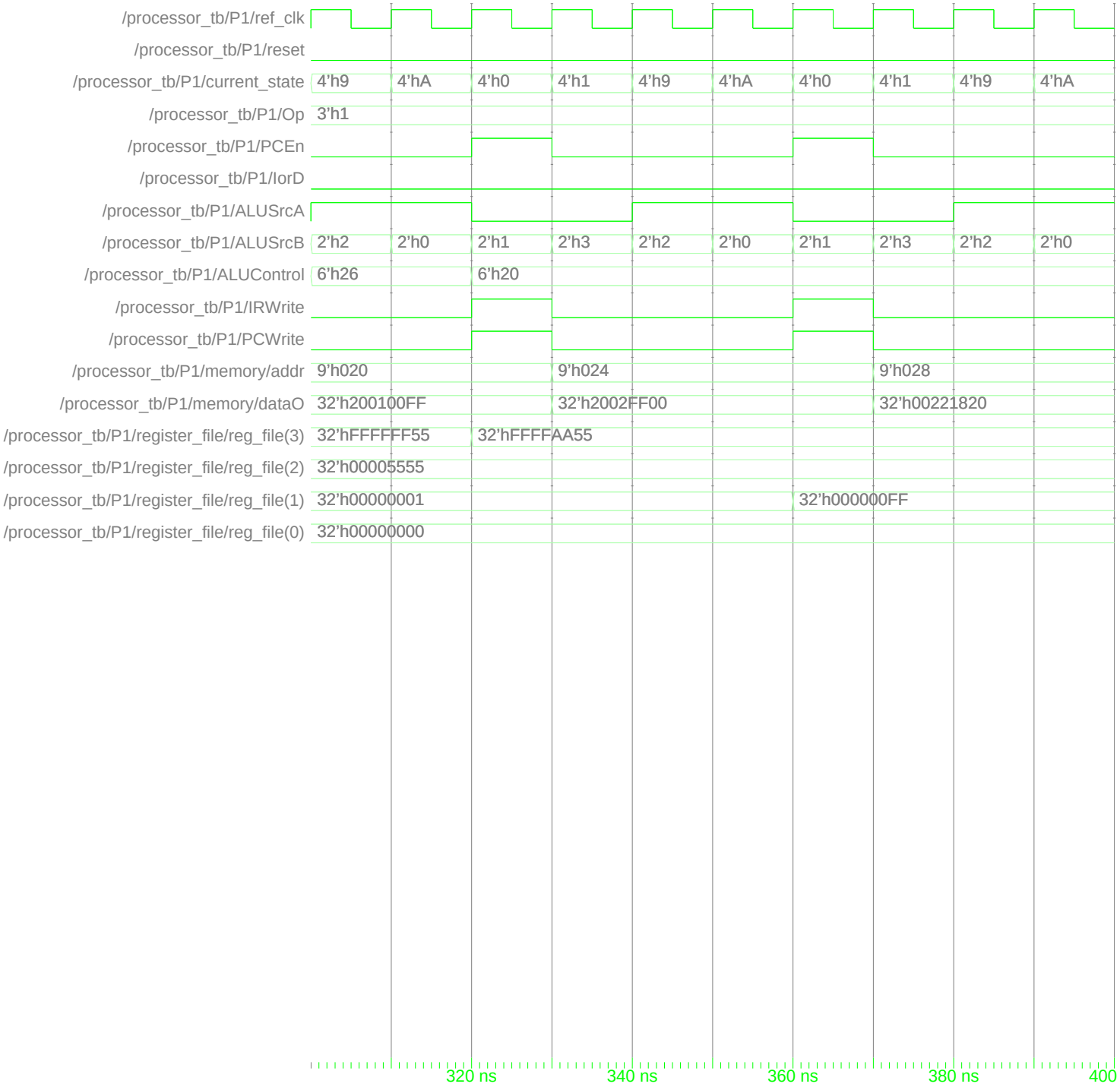
A separate file called rom.txt provides the MIPS equivalent of the binary code.

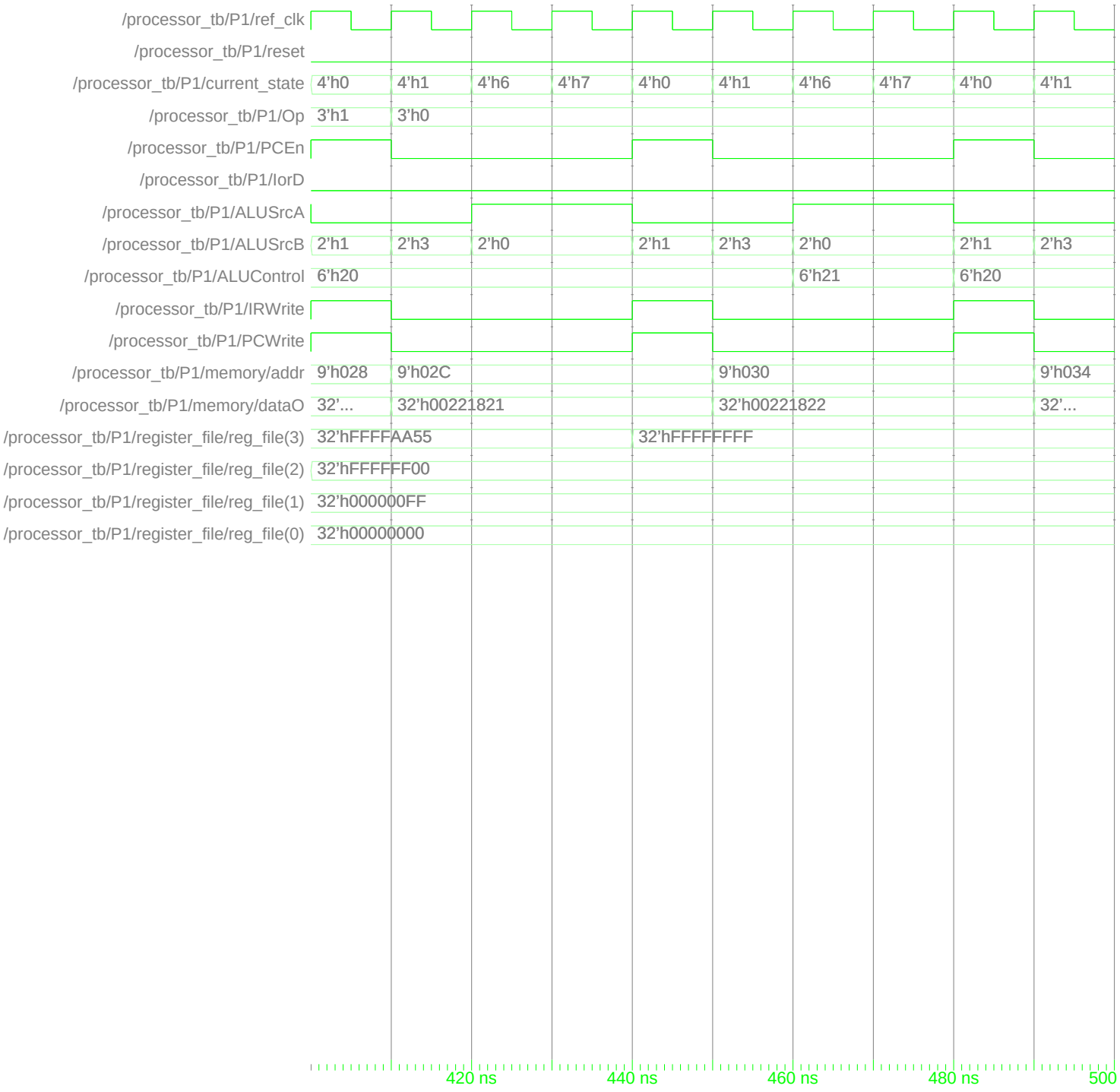


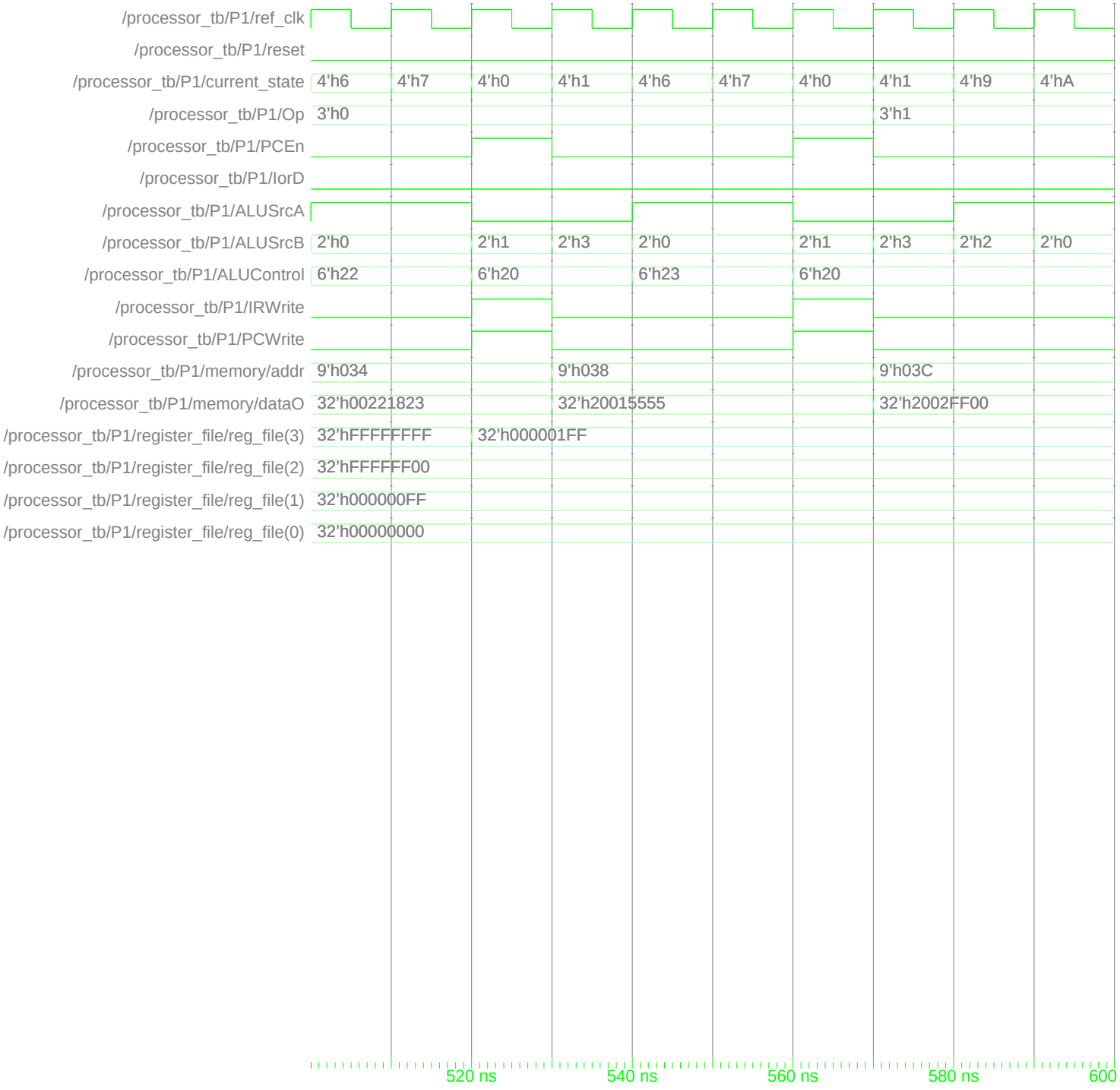
Entity: Architecture: Date: Mon Feb 22 10:06:00 PST 2016 Row: 1 Page: 1

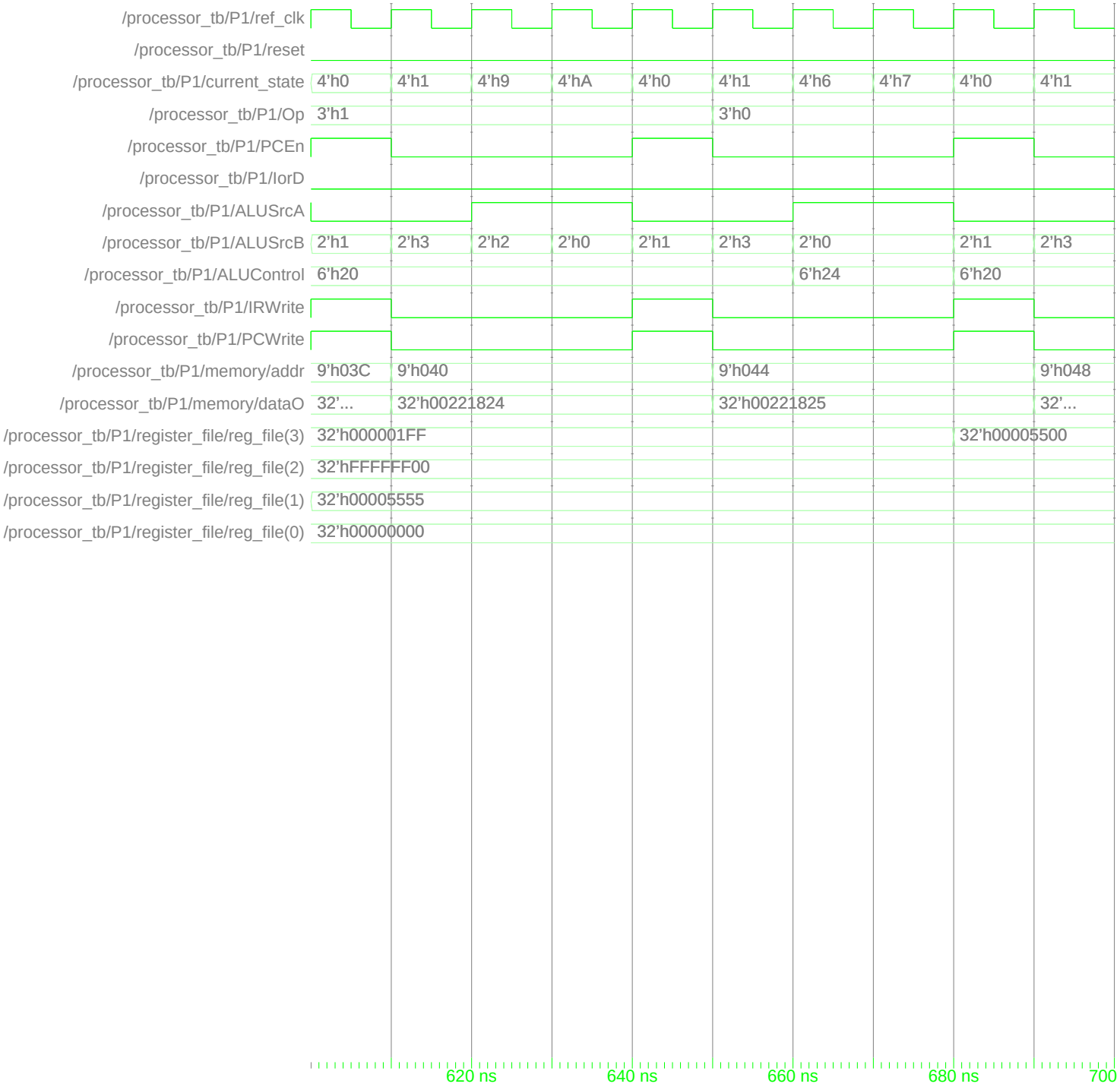

/processor_tb/P1/ref_clk
/processor_tb/P1/reset
/processor_tb/P1/current_state
4'h9
4'hA
4'h0
4'h1
4'h9
4'hA
4'h0
4'h1
4'h9
4'hA
/processor_tb/P1/Op
3'h1
/processor_tb/P1/PCEn
/processor_tb/P1/IorD
/processor_tb/P1/ALUSrcA
/processor_tb/P1/ALUSrcB
2'h2
2'h0
2'h1
2'h3
2'h2
2'h0
2'h1
2'h3
2'h2
2'h0
/processor_tb/P1/ALUControl
6'h28
6'h20
6'h29
6'h20
/processor_tb/P1/IRWrite
/processor_tb/P1/PCWrite
/processor_tb/P1/memory/addr
9'h00C
9'h010
9'h014
/processor_tb/P1/memory/dataO
32'h2C01FFFF
32'h20025555
32'h3043FF00
/processor_tb/P1/register_file/reg_file(3)
32'h00000000
/processor_tb/P1/register_file/reg_file(2)
32'h00000000
/processor_tb/P1/register_file/reg_file(1)
32'hFFFFFFFF
32'h00000000
32'h00000001
/processor_tb/P1/register_file/reg_file(0)
32'h00000000
120 ns
140 ns
160 ns
180 ns
200


/processor_tb/P1/ref_clk
/processor_tb/P1/reset
/processor_tb/P1/current_state
4'h0
4'h1
4'h9
4'hA
4'h0
4'h1
4'h9
4'hA
4'h0
4'h1
/processor_tb/P1/Op
3'h1
/processor_tb/P1/PCEn
/processor_tb/P1/IorD
/processor_tb/P1/ALUSrcA
/processor_tb/P1/ALUSrcB
2'h1
2'h3
2'h2
2'h0
2'h1
2'h3
2'h2
2'h0
2'h1
2'h3
/processor_tb/P1/ALUControl
6'h20
6'h24
6'h20
6'h25
6'h20
/processor_tb/P1/IRWrite
/processor_tb/P1/PCWrite
/processor_tb/P1/memory/addr
9'h014
9'h018
9'h01C
9'h020
/processor_tb/P1/memory/dataO
32'...
32'h3443FF00
32'h3843FF00
32'...
/processor_tb/P1/register_file/reg_file(3)
32'h00000000
32'h00005500
32'hFFFFFF55
/processor_tb/P1/register_file/reg_file(2)
32'h00005555
/processor_tb/P1/register_file/reg_file(1)
32'h00000001
/processor_tb/P1/register_file/reg_file(0)
32'h00000000
220 ns
240 ns
260 ns
280 ns
300

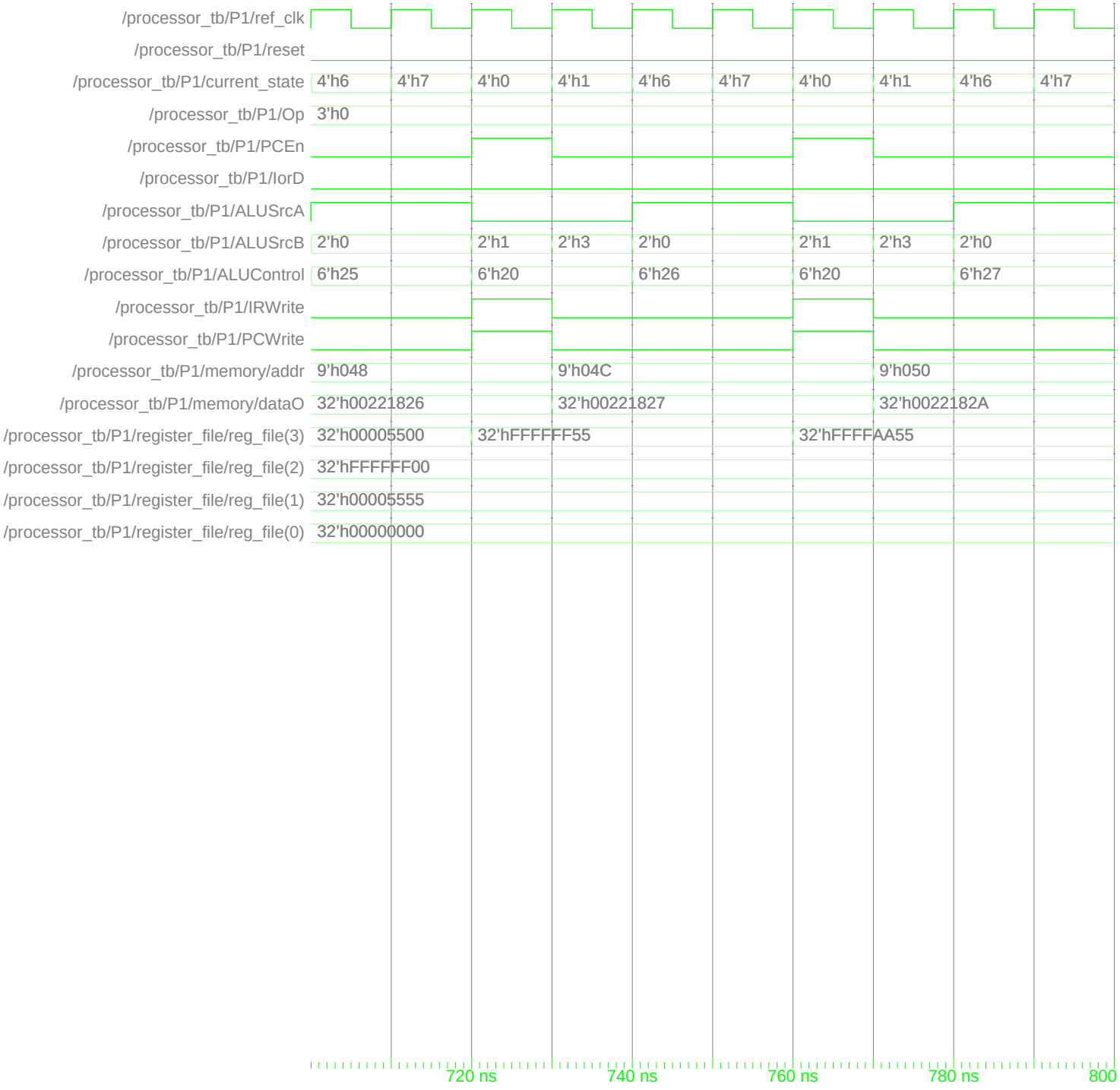

/processor_tb/P1/ref_clk
/processor_tb/P1/reset
/processor_tb/P1/current_state
4'h9
4'hA
4'h0
4'h1
4'h9
4'hA
4'h0
4'h1
4'h9
4'hA
/processor_tb/P1/Op
3'h1
/processor_tb/P1/PCEn
/processor_tb/P1/IorD
/processor_tb/P1/ALUSrcA
/processor_tb/P1/ALUSrcB
2'h2
2'h0
2'h1
2'h3
2'h2
2'h0
2'h1
2'h3
2'h2
2'h0
/processor_tb/P1/ALUControl
6'h26
6'h20
/processor_tb/P1/IRWrite
/processor_tb/P1/PCWrite
/processor_tb/P1/memory/addr
9'h020
9'h024
9'h028
/processor_tb/P1/memory/dataO
32'h200100FF
32'h2002FF00
32'h00221820
/processor_tb/P1/register_file/reg_file(3)
32'hFFFFFF55
32'hFFFFAA55
/processor_tb/P1/register_file/reg_file(2)
32'h00005555
/processor_tb/P1/register_file/reg_file(1)
32'h00000001
32'h000000FF
/processor_tb/P1/register_file/reg_file(0)
32'h00000000
320 ns
340 ns
360 ns
380 ns
400

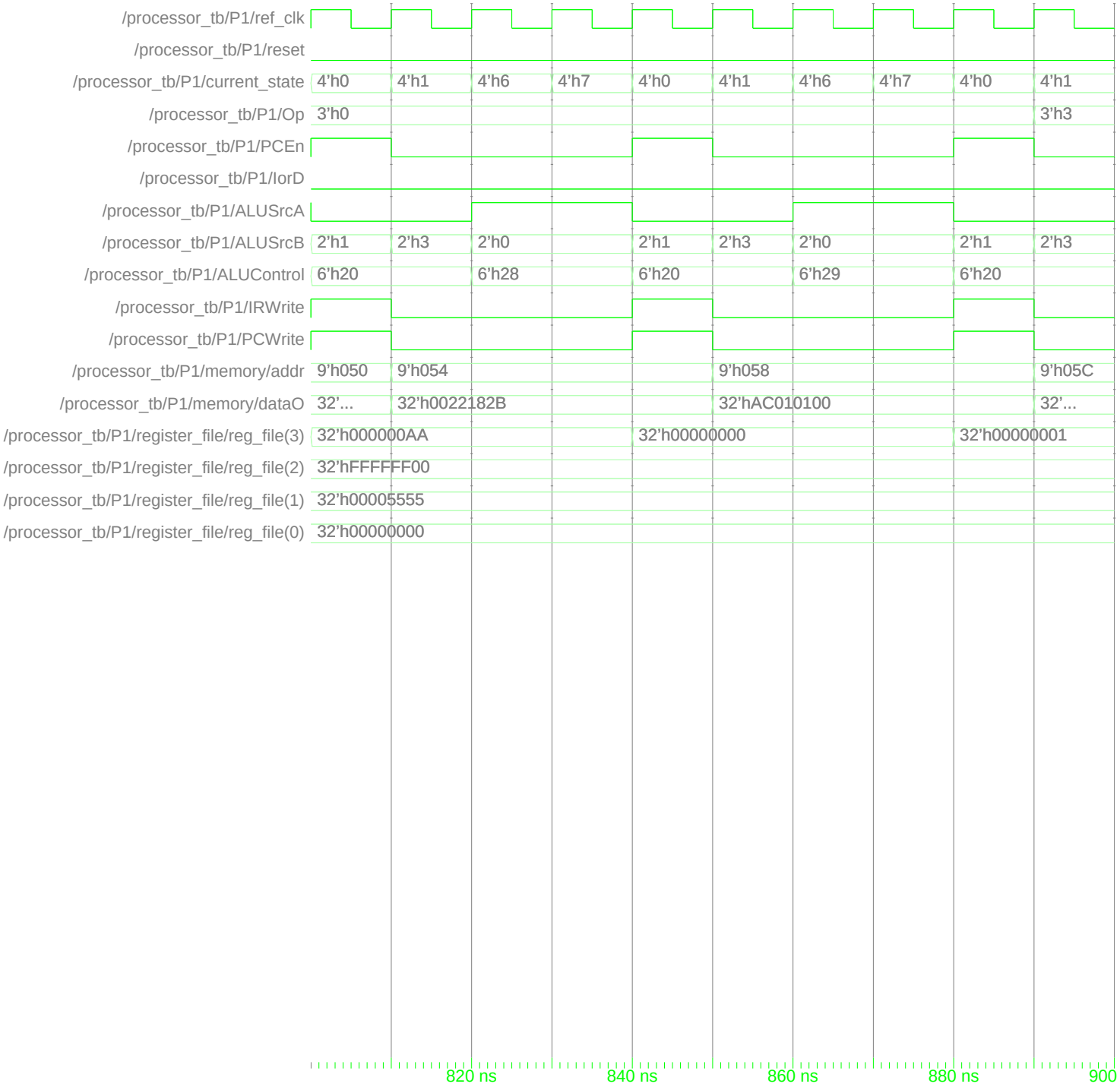

/processor_tb/P1/ref_clk
/processor_tb/P1/reset
/processor_tb/P1/current_state
4'h0
4'h1
4'h6
4'h7
4'h0
4'h1
4'h6
4'h7
4'h0
4'h1
/processor_tb/P1/Op
3'h1
3'h0
/processor_tb/P1/PCEn
/processor_tb/P1/IorD
/processor_tb/P1/ALUSrcA
/processor_tb/P1/ALUSrcB
2'h1
2'h3
2'h0
2'h1
2'h3
2'h0
2'h1
2'h3
/processor_tb/P1/ALUControl
6'h20
6'h21
6'h20
/processor_tb/P1/IRWrite
/processor_tb/P1/PCWrite
/processor_tb/P1/memory/addr
9'h028
9'h02C
9'h030
9'h034
/processor_tb/P1/memory/dataO
32'...
32'h00221821
32'h00221822
32'...
/processor_tb/P1/register_file/reg_file(3)
32'hFFFFAA55
32'hFFFFFFFF
/processor_tb/P1/register_file/reg_file(2)
32'hFFFFFF00
/processor_tb/P1/register_file/reg_file(1)
32'h000000FF
/processor_tb/P1/register_file/reg_file(0)
32'h00000000
420 ns
440 ns
460 ns
480 ns
500


/processor_tb/P1/ref_clk
/processor_tb/P1/reset
/processor_tb/P1/current_state
4'h6
4'h7
4'h0
4'h1
4'h6
4'h7
4'h0
4'h1
4'h9
4'hA
/processor_tb/P1/Op
3'h0
3'h1
/processor_tb/P1/PCEn
/processor_tb/P1/IorD
/processor_tb/P1/ALUSrcA
/processor_tb/P1/ALUSrcB
2'h0
2'h1
2'h3
2'h0
2'h1
2'h3
2'h2
2'h0
/processor_tb/P1/ALUControl
6'h22
6'h20
6'h23
6'h20
/processor_tb/P1/IRWrite
/processor_tb/P1/PCWrite
/processor_tb/P1/memory/addr
9'h034
9'h038
9'h03C
/processor_tb/P1/memory/dataO
32'h00221823
32'h20015555
32'h2002FF00
/processor_tb/P1/register_file/reg_file(3)
32'hFFFFFFFF
32'h000001FF
/processor_tb/P1/register_file/reg_file(2)
32'hFFFFFF00
/processor_tb/P1/register_file/reg_file(1)
32'h000000FF
/processor_tb/P1/register_file/reg_file(0)
32'h00000000
520 ns
540 ns
560 ns
580 ns
600

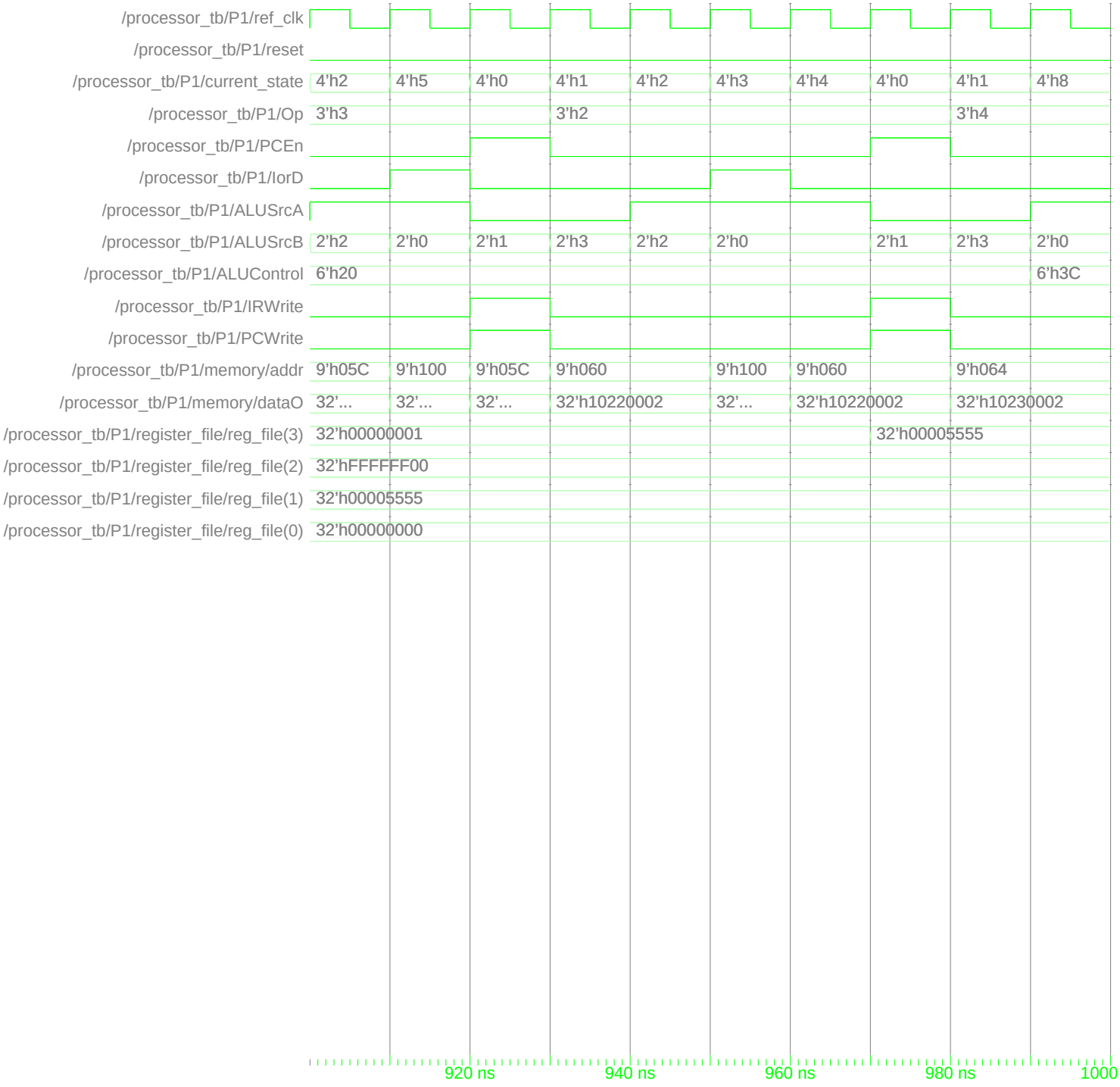


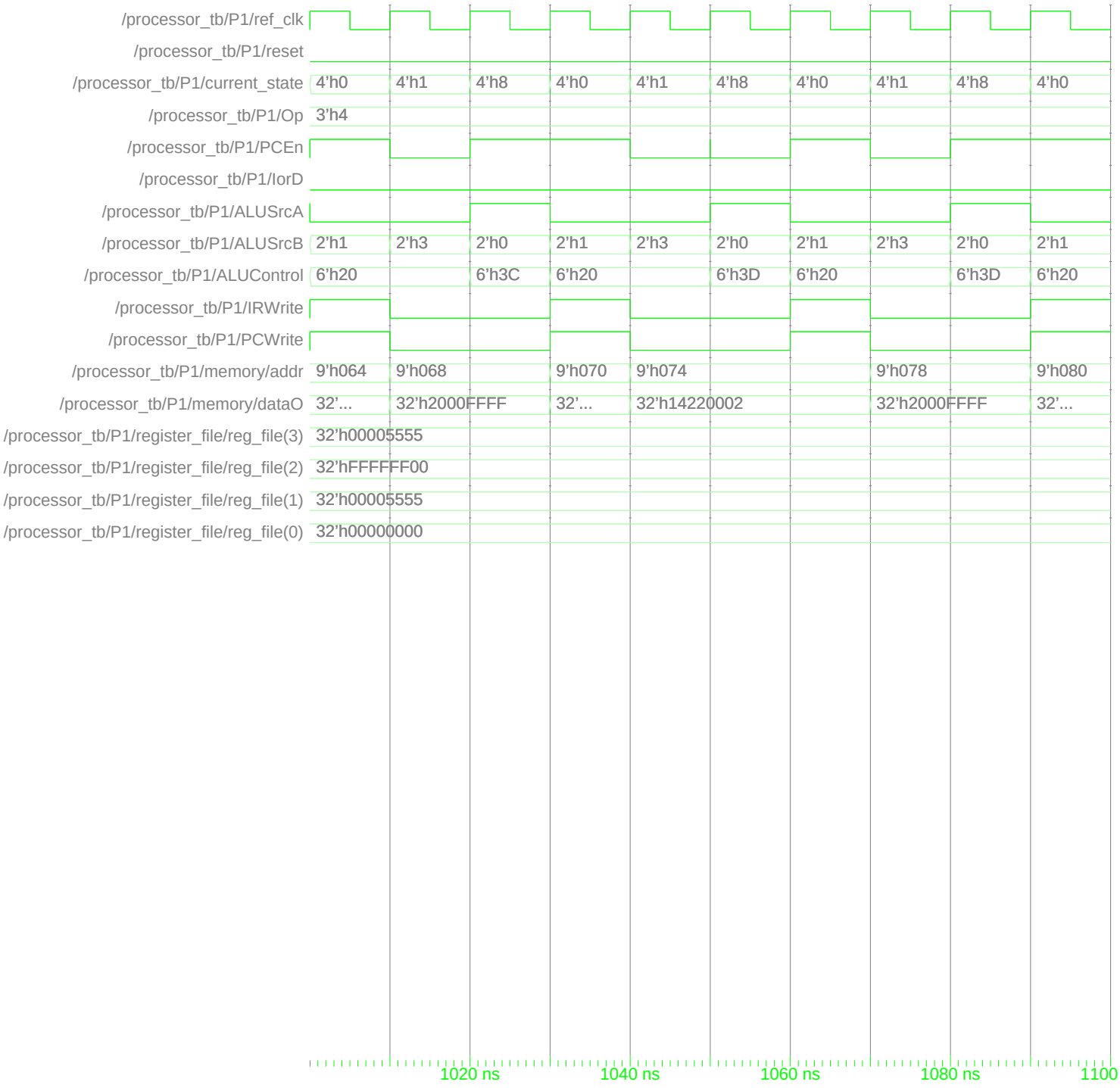
Entity: Architecture: Date: Mon Feb 22 10:06:00 PST 2016 Row: 7 Page: 7

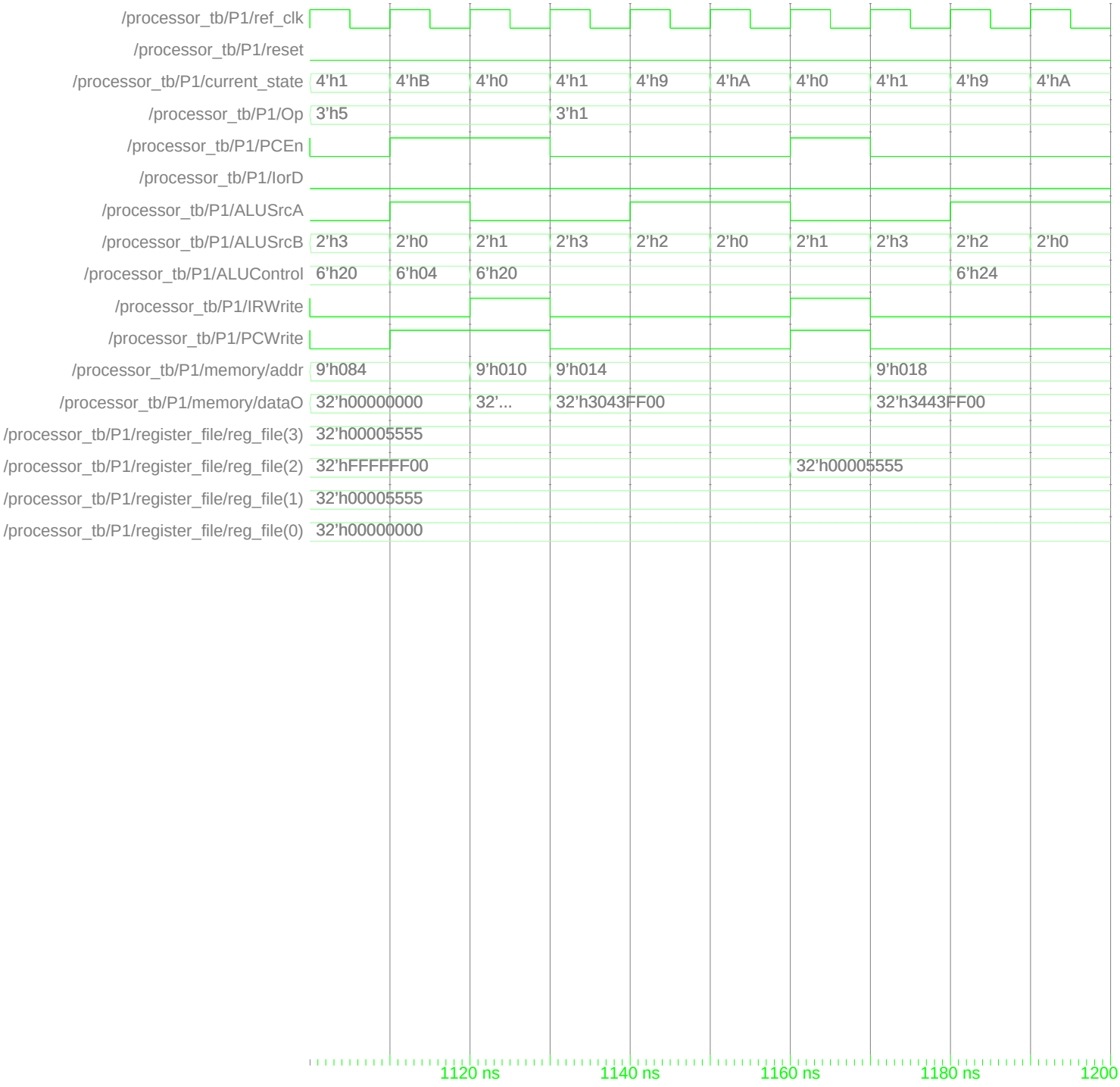

/processor_tb/P1/ref_clk
/processor_tb/P1/reset
/processor_tb/P1/current_state
4'h6
4'h7
4'h0
4'h1
4'h6
4'h7
4'h0
4'h1
4'h6
4'h7
/processor_tb/P1/Op
3'h0
/processor_tb/P1/PCEn
/processor_tb/P1/IorD
/processor_tb/P1/ALUSrcA
/processor_tb/P1/ALUSrcB
2'h0
2'h1
2'h3
2'h0
2'h1
2'h3
2'h0
/processor_tb/P1/ALUControl
6'h25
6'h20
6'h26
6'h20
6'h27
/processor_tb/P1/IRWrite
/processor_tb/P1/PCWrite
/processor_tb/P1/memory/addr
9'h048
9'h04C
9'h050
/processor_tb/P1/memory/dataO
32'h00221826
32'h00221827
32'h0022182A
/processor_tb/P1/register_file/reg_file(3)
32'h00005500
32'hFFFFFF55
32'hFFFFAA55
/processor_tb/P1/register_file/reg_file(2)
32'hFFFFFF00
/processor_tb/P1/register_file/reg_file(1)
32'h00005555
/processor_tb/P1/register_file/reg_file(0)
32'h00000000
720 ns
740 ns
760 ns
780 ns
800



Entity: Architecture: Date: Mon Feb 22 10:06:00 PST 2016 Row: 9 Page: 9


/processor_tb/P1/ref_clk
/processor_tb/P1/reset
/processor_tb/P1/current_state
4'h2
4'h5
4'h0
4'h1
4'h2
4'h3
4'h4
4'h0
4'h1
4'h8
/processor_tb/P1/Op
3'h3
3'h2
3'h4
/processor_tb/P1/PCEn
/processor_tb/P1/IorD
/processor_tb/P1/ALUSrcA
/processor_tb/P1/ALUSrcB
2'h2
2'h0
2'h1
2'h3
2'h2
2'h0
2'h1
2'h3
2'h0
/processor_tb/P1/ALUControl
6'h20
6'h3C
/processor_tb/P1/IRWrite
/processor_tb/P1/PCWrite
/processor_tb/P1/memory/addr
9'h05C
9'h100
9'h05C
9'h060
9'h100
9'h060
9'h064
/processor_tb/P1/memory/dataO
32'...
32'...
32'...
32'h10220002
32'...
32'h10220002
32'h10230002
/processor_tb/P1/register_file/reg_file(3)
32'h00000001
32'h00005555
/processor_tb/P1/register_file/reg_file(2)
32'hFFFFFF00
/processor_tb/P1/register_file/reg_file(1)
32'h00005555
/processor_tb/P1/register_file/reg_file(0)
32'h00000000
920 ns
940 ns
960 ns
980 ns
1000


/processor_tb/P1/ref_clk
/processor_tb/P1/reset
/processor_tb/P1/current_state
4'h0
4'h1
4'h8
4'h0
4'h1
4'h8
4'h0
4'h1
4'h8
4'h0
/processor_tb/P1/Op
3'h4
/processor_tb/P1/PCEn
/processor_tb/P1/IorD
/processor_tb/P1/ALUSrcA
/processor_tb/P1/ALUSrcB
2'h1
2'h3
2'h0
2'h1
2'h3
2'h0
2'h1
2'h3
2'h0
2'h1
/processor_tb/P1/ALUControl
6'h20
6'h3C
6'h20
6'h3D
6'h20
6'h3D
6'h20
/processor_tb/P1/IRWrite
/processor_tb/P1/PCWrite
/processor_tb/P1/memory/addr
9'h064
9'h068
9'h070
9'h074
9'h078
9'h080
/processor_tb/P1/memory/dataO
32'...
32'h2000FFFF
32'...
32'h14220002
32'h2000FFFF
32'...
/processor_tb/P1/register_file/reg_file(3)
32'h00005555
/processor_tb/P1/register_file/reg_file(2)
32'hFFFFFF00
/processor_tb/P1/register_file/reg_file(1)
32'h00005555
/processor_tb/P1/register_file/reg_file(0)
32'h00000000
1020 ns
1040 ns
1060 ns
1080 ns
1100


/processor_tb/P1/ref_clk
/processor_tb/P1/reset
/processor_tb/P1/current_state
4'h1
4'hB
4'h0
4'h1
4'h9
4'hA
4'h0
4'h1
4'h9
4'hA
/processor_tb/P1/Op
3'h5
3'h1
/processor_tb/P1/PCEn
/processor_tb/P1/IorD
/processor_tb/P1/ALUSrcA
/processor_tb/P1/ALUSrcB
2'h3
2'h0
2'h1
2'h3
2'h2
2'h0
2'h1
2'h3
2'h2
2'h0
/processor_tb/P1/ALUControl
6'h20
6'h04
6'h20
6'h24
/processor_tb/P1/IRWrite
/processor_tb/P1/PCWrite
/processor_tb/P1/memory/addr
9'h084
9'h010
9'h014
9'h018
/processor_tb/P1/memory/dataO
32'h00000000
32'...
32'h3043FF00
32'h3443FF00
/processor_tb/P1/register_file/reg_file(3)
32'h00005555
/processor_tb/P1/register_file/reg_file(2)
32'hFFFFFF00
32'h00005555
/processor_tb/P1/register_file/reg_file(1)
32'h00005555
/processor_tb/P1/register_file/reg_file(0)
32'h00000000
1120 ns
1140 ns
1160 ns
1180 ns
1200

CPU operation types		
op	Operation Type	Comments
000000	R-type instruction	The funct field will be sent to the ALU
001XXX	I-type instruction	The controller will determine the ALU function code
00001X	J-type instruction	The controller will determine the ALU function code
100011	Load Word	ALU operation will be addi
101011	Store Word	ALU operation will be addi
Others	nop	ALU function code will be set to nop

3 Changes from Single-cycle Processor

3.1 Memory Unit

There is a single memory unit in this design which is the location for both the instructions and data. The RAM used as data memory replaced the ROM used as instruction memory, and the RAM can now store both data and instructions. Since the instructions and data are sharing the memory unit, a MUX is needed to choose between the two types of memory. The size of the memory unit in our design is 512 lines with one word or 32-bits per line.

RAM Port Description			
Port name	Port size	Port Type	Description
clk	1	IN	clock signal
we	1	IN	write enable
addr	32	IN	address
dataI	32	IN	input data
dataO	32	OUT	output data

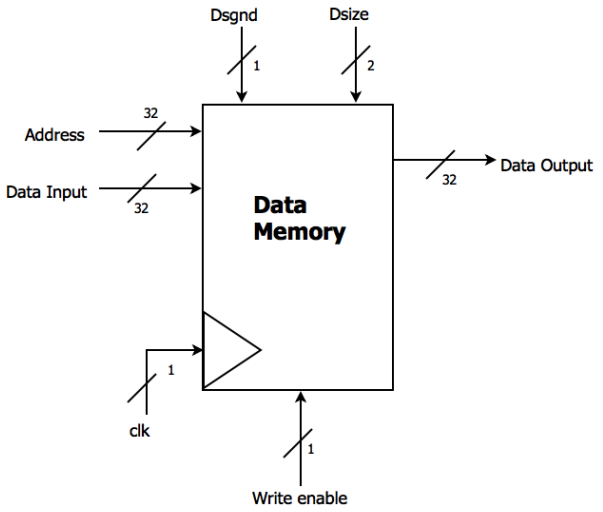


Figure 2: Random Access Memory

3.2 ALU

In addition to performing the regular arithmetic and logic operations for the instruction, the ALU now also handles the incrementing of the program counter, and the addition of the branch offset to the program counter. The ALU's A and B inputs are now connected to multiplexers. The multiplexer connected to the ALU's A input allows for choosing between register read data 1 and the program counter value. The multiplexer connected to the ALU's B input allows for choosing between register read data 2, 4 (for incrementing the program counter), and the shifted immediate value (used as branch offset for branch instructions).

3.3 Controller

The controller has been modified to handle select values for newly added selectors, as well as enabling or disabling writing to the registers in the design. In addition to taking the current instruction, the controller now also takes the current state, which it uses to set the correct values for select and write enable signals for each state of the FSM. The control signals and their purpose are as follows:

Controller Port Description			
Port name	Port size	Port Type	Description
Instr	32	IN	Current instruction
state	4	IN	Current state
ALUControl	6	OUT	ALU function code
ALUSrcA	1	OUT	Selecting ALU Operand A
ALUSrcB	2	OUT	Selecting ALU Operand B
MemtoReg	1	OUT	Selecting data to write to regfile
MemWrite	1	OUT	Enable memory write
RegWrite	1	OUT	Enable regfile write
Branch	1	OUT	controls branch instruction
Op	3	OUT	Indicates the current instruction's type
IorD	1	OUT	Chooses between instruction and data for the address to the memory
IRWrite	1	OUT	Update the instruction register
PCWrite	1	OUT	Update the program counter
RegDst	1	OUT	Select between rt and rd for register write address
PCSrc	2	OUT	Select the source address for the program counter

4 Supported Instructions

The following table shows the instructions supported by this multi-cycle processor:

Supported Instructions			
Arithmetic	Shift	Branch	Memory Operation
ADD	SLLV	BEQ	LW
ADDU	SRLV	BNE	SW
SUB	SRAV	BLTZ	
SUBU		BGEZ	
AND		BLEZ	
OR		BGTZ	
XOR		JUMP	
NOR			
SLT			
SLTU			
ADDIU			
ANDI			
ORI			
XORI			
SLTI			
SLTIU			

4.1 Newly Added Components

To be able to use the ALU and RAM for multiple purposes, we need additional multiplexers for selecting between different inputs to these blocks, as well as registers to hold values for later use. The added multiplexers and registers are:

- A mux for choosing between an instruction address or a data address to input to the memory
- A register to store the current instruction
- A register to store data loaded from the memory
- Registers to store the register file's read data 1 and 2
- A mux to select between the register read data 1 and the program counter's value for the ALU's A input
- A mux to select between the register read data 2, the value 4, and the shifted immediate value for the ALU's B input
- A register to store the result of the ALU
- A mux to select between PC+4, jump target, or branch target, for the next value of the program counter

4.2 Removed components

As the ALU is modified to handle incrementing of the program counter, as well as the addition of the branch offset to the program counter, the two adders used for these two operations have been removed. Also, the instruction memory has been replaced by the data memory, which is now capable of storing instructions and reading both data and instructions.

5 Unmodified Components

These components have not been modified from the previous design and thus the waveforms associated with the testbenches are not included.

5.1 Program Counter

The program counter is a register that stores the address of the instruction being executed. The address in the program counter is passed to the instruction memory. After the instruction is fetched, the counter gets incremented to go to the next address. In our case, the program counter is incremented by 4 using an adder as our design is byte-addressable. Our implementation of the program counter, takes in a clock signal, reset, and input. It outputs a 32-bit address. The program counter was changed from the previous design so that it no longer increments its output by itself. Instead, an adder takes the output of the program counter.

Program Counter Port Description			
Port name	Port size	Port Type	Description
clk	1	IN	clock signal
rst	1	IN	reset bit
input	32	IN	incremented value
output	32	OUT	outputs an address

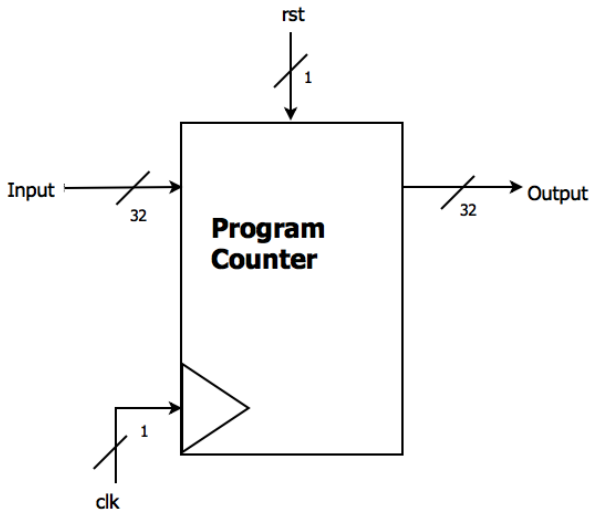


Figure 3: Program Counter Diagram

5.2 Register File

The register file has 32 registers, each 32 bits wide. There are 2 read ports and one write port. The read ports are asynchronous and the write port is synchronous. The register file has a synchronous reset signal and a write enable signal.

Register Port Description			
Port name	Port size	Port Type	Description
clk	1	IN	clock signal
rst_s	1	IN	synchronous reset
we	1	IN	write enable
raddr_1	5	IN	read address 1
raddr_2	5	IN	read address 2
waddr	5	IN	write address
rdata_1	32	OUT	read data 1
rdata_2	32	OUT	read data 2
wdata	32	IN	write data

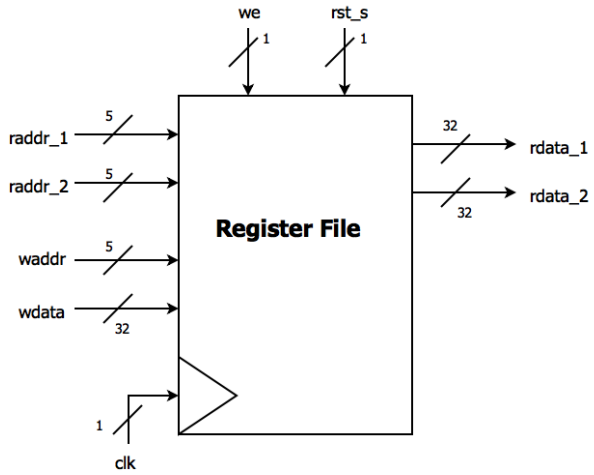


Figure 4: Register file

5.3 ALU

The ALU performs arithmetic and logic operations on two 32-bit operands and produces a 32-bit output. It also provides a one-bit value for the branch instruction.

ALU Port Description			
Port name	Port size	Port Type	Description
Func_in	6	IN	opcode from controller
A_in	32	IN	operand A
B_in	32	IN	operand B
O_out	32	OUT	output from ALU
Branch_out	1	OUT	for branch instruction

ALU Function Description			
Instruction	Description	Function Code	Comments
nop	nothing	"000000"	
add/addi	rd ← rs + rt / rd ← rs + immediate	"100000"	
addu/addiu	rd ← rs + rt / rd ← rs + immediate	"100001"	
sub	rd ← rs - rt / rd ← rs - immediate	"100010"	
subu	rd ← rs - rt / rd ← rs - immediate	"100011"	
and/andi	rd ← rs AND rt / rd ← rs AND immediate	"100100"	
or/ori	rd ← rs OR rt / rd ← rs OR immediate	"100101"	
xor/xori	rd ← rs XOR rt / rd ← rs XOR immediate	"100110"	
nor	rd ← rs XOR rt / rd ← rs NOR immediate	"100111"	
slt/slti	Set rd if rs < rt / set rd if rs < immediate	"101000"	If the condition is satisfied set else reset destination
sltu/sltiu	Set rd if rs < rt / set rd if rs < immediate	"101001"	If the condition is satisfied set else reset destination
sll	B << A	000X00	
srl	B >> A	000X10	
sra	B >>> A	000X11	
bltz	A < 0	111000	
bgez	A >= 0	111001	
beq	A == B	111100	
bne	A ! = B	111101	
blez	A <= 0	111110	
bgtz	A > 0	111111	
nop	nothing	others	Any other function code does nothing

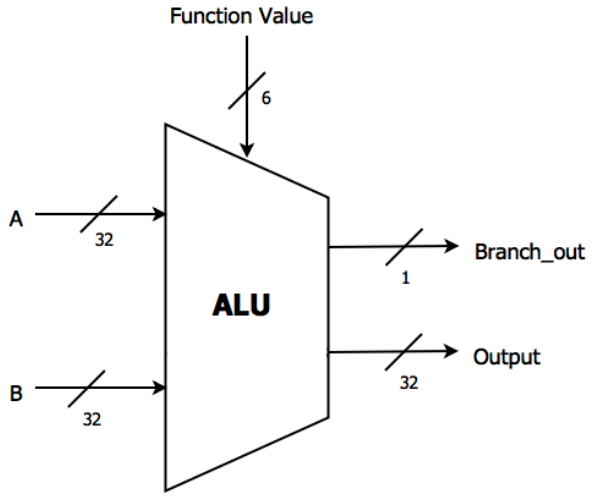


Figure 5: The ALU has 3 inputs and two outputs. There are two 32-bit inputs that act as the operands and a 6-bit input that lets the ALU know which operation to perform. The result is then sent to the output and a value for the branch.

5.4 Multiplexer

In general, the multiplexer chooses an output from several possible inputs based on the value of the select signal. Our design uses four 2:1 multiplexers, and two 4:1 multiplexers. The following are signals from the controller for the select values of the multiplexers:

1. ALUSrcA chooses the value for the ALU's A input.
2. ALUSrcB chooses the value for the ALU's B input.
3. MemtoReg chooses the value for the register file's write data
4. PCSrc chooses the next value for the program counter.
5. IorD chooses the address for the memory.
6. RegDst chooses the write address for the register file.

Multiplexer Port Description			
Port name	Port size	Port Type	Description
s	1	IN	select line
a	32	IN	data 1
b	32	IN	data 2
o	32	OUT	selected data

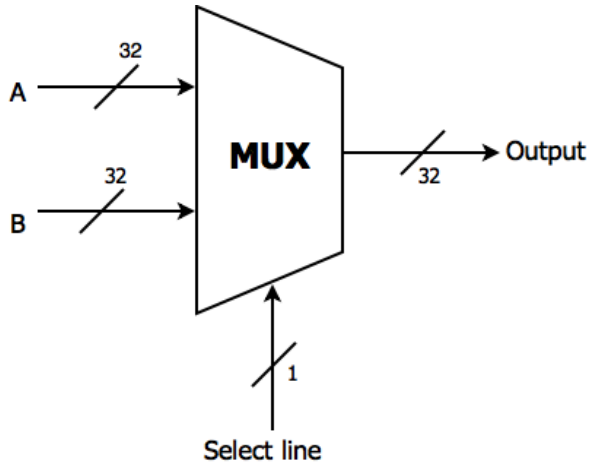


Figure 6: 2-to-1 MUX

5.5 Sign Extension Unit

The sign extension unit performs sign extension on the 16-bit immediate value provided by the controller to make it 32-bits wide, so that it can be used in as a second operand in the ALU.

Sign Extender Port Description			
Port name	Port size	Port Type	Description
input	16	IN	input value
output	32	OUT	extended value to 32-bits

Figure 7: Takes in a 16-bit value and extends it to 32-bits.

6 Conclusion

In a single-cycle processor, each component has a specific function, and all components perform their functions for the same instruction in one cycle. In contrast, in a multi-cycle processor, each instruction spans multiple cycles, and different instructions may take different numbers of cycles to complete. Also, since each cycle is devoted to a specific phase for the instruction, blocks with the same functionality can be merged, as one block can serve different purposes in each cycle.

The use of an FSM to represent the phases of a multi-cycle processor aids in making the processor design systematic and readable. Each state represents a phase in the execution of an instruction (fetch, decode, execute, memory, write-back), and transitions to each state depend on the type of the current instruction.