Tyler Thompson
An Nguyen

ECE 3570 Lab 3b
10-bit Instruction Set Architecture CPU - Single-Cycle CPU
March 21st, 2018

Critical Path Delay

Register File: 6.482ns
ALU: 8.823ns
Fetch Unit: 4.520ns
Control Unit: 6.141ns
Data Memory: 9.630ns
Total Max Path Delay: 35.396ns

Answers to Design Questions

A. Changes were made in the decode side of the register file so that it used gate level logic to decode instead of a bunch of multiplexers. The stack pointer register was modified so that it's reset value will point it to the top of the stack.

B. The total longest path delay is 35.396ns, which includes delay times for all modules. This results in a minimum clock period time of **36ns**.

C. A two's complement module was added to the ALU so that taking the two's complement of a number can happen in one instruction. The decode side of the register file was optimized so that writing to registers happens faster.

D. The ALU takes the longest time to execute instructions, so all registers values and instructions are read at the beginning of the clock cycle in order to feed that ALU as quickly as possible. Data memory also takes a long time to write to memory, so values are stored at the beginning of the clock cycle. That way values to be store for and instruction are stored at the beginning of the next instruction.

E. Program 1 - (19n^2) -9 (Assuming array starts in the worst case of being sorted backwards. I.e. n = 10, IC = 1891)

   Program 2 - 14 + 4*(X - 1) (i.e. X=4, IC = 26)

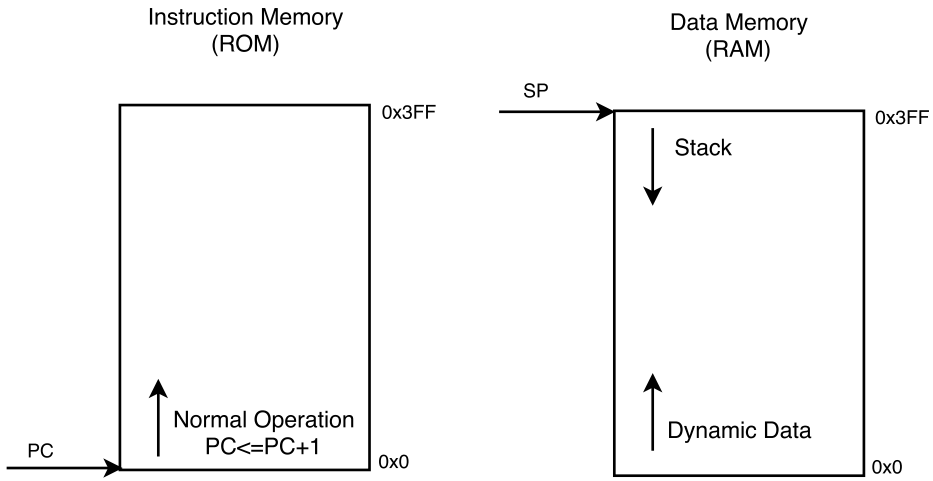
   Program 3 - 10 + 6 * Size Of Array (i.e. Array Size = 10, IC = 70)

10-Bit ISA Instruction Reference Sheet

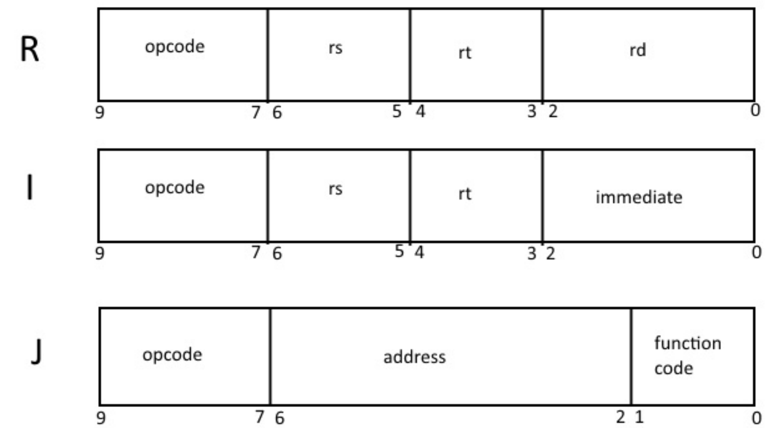
Instructions

Name	**Description**	**Format**	**RTL**	**Syntax**	**OP Code (hex)**	**Function Code (hex)**
add	Add two values in registers together	R	if MEM[PC] == ADD rs rt rd [rs] <= [rt] + [rd] PC <= PC + 1	add $rs, $rt, $rd	0x0	
addi	Add a 3-bit signed contant to a value in a register	I	if MEM[PC] == ADDI rs rt imm [rs] <= [rt] + sign-ext(imm) PC <= PC + 1	addi $rs, $rt, imm	0x4	
tcp	Take the two's complement of a value in a register Imm value should be set to 100 (-0) to differentiate from addi	I	if MEM[PC] == TCP rs rt [rs] <= ~[rt] + 1 PC <= PC + 1	tcp $rs, $rt	0x4	
sw	Store a word in memory	I	if MEM[PC] == SW rt offset(base) address = sign-extend(offset) + [base] MEM[address] <= [rt] PC <= PC + 1	sw $rt, M($rs)	0x5	
lw	Load a word from memory	I	if MEM[PC] == LW rt offset(base) address = sign-extend(offset) + [base] [rt] <= MEM[address] PC <= PC + 1	lw $rs, M($rt)	0x6	
sll	Shift left logical	R	if MEM[PC] == SLL rd rs rt [rd] <= [rs] << [rt] PC = PC + 1	sll $rd, $rs, $rt	0x1	
cmp	Compare	R	if MEM[PC] == CMP rd rs rt [rd] <= [rs] < [rt] - 1 [rd] <= [rs] == [rt] - 0 [rd] <= [rs] > [rt] - 2 PC = PC + 1	cmp $rd, $rs, $rt	0x2	
beq	Branch equal	R	if MEM[PC] == BEQ rd rs rt if [rs] == [rt] PC <= PC + 4 + [rd]	beq $rd, $rs, $rt	0x3	
jal	Jump and link	J	if MEM[PC] == JAL address [ra] <= PC + 1 PC <= PC + address	jal Label	0x7	0x2
j	Jump	J	if MEM[PC] == J address PC <= PC + address	j Label	0x7	0x1
jr	Jump Register	J	if MEM[PC] == JR $reg PC <= $reg	jr $reg	0x7	0x0
halt	Halt the machine	J	if MEM[PC] == HALT PC <= 0x0	halt	0x7	0x3

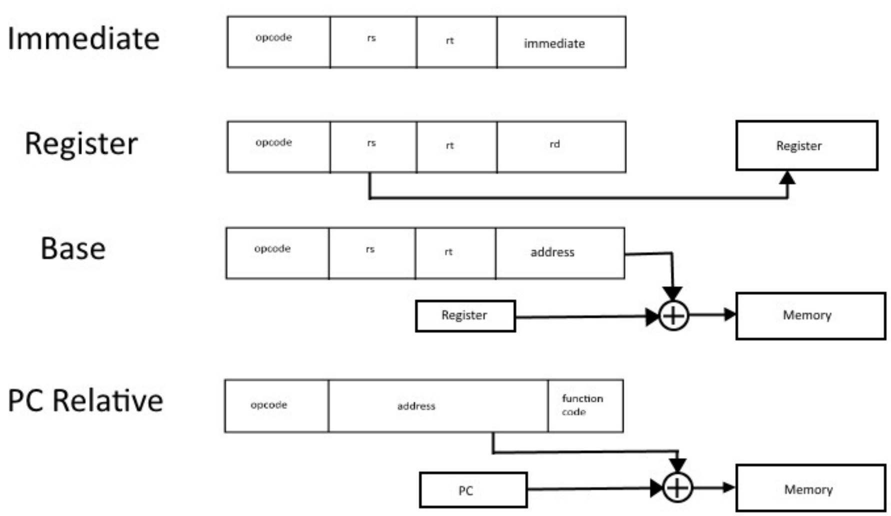
Memory Allocation



Instruction Formats



Addressing Modes

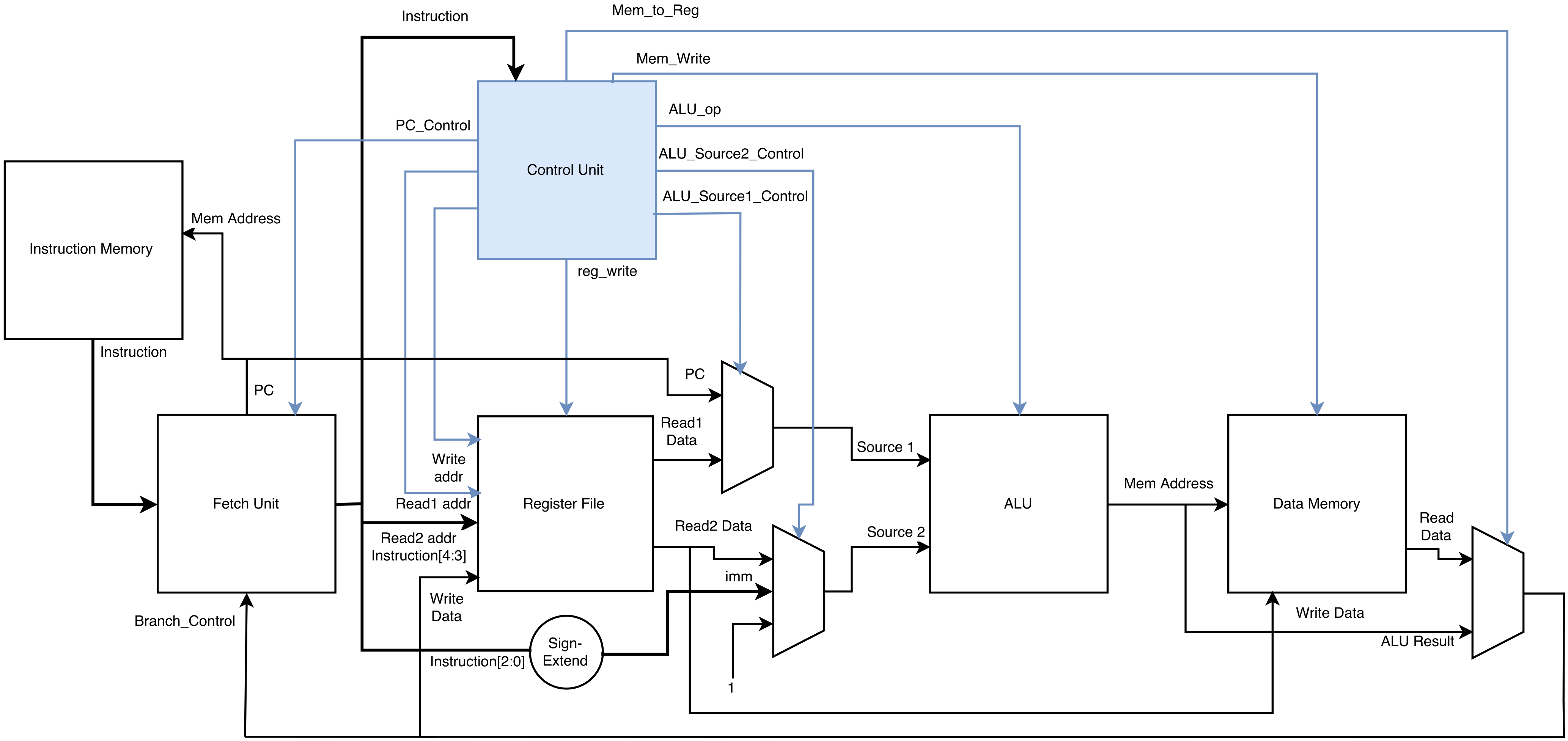


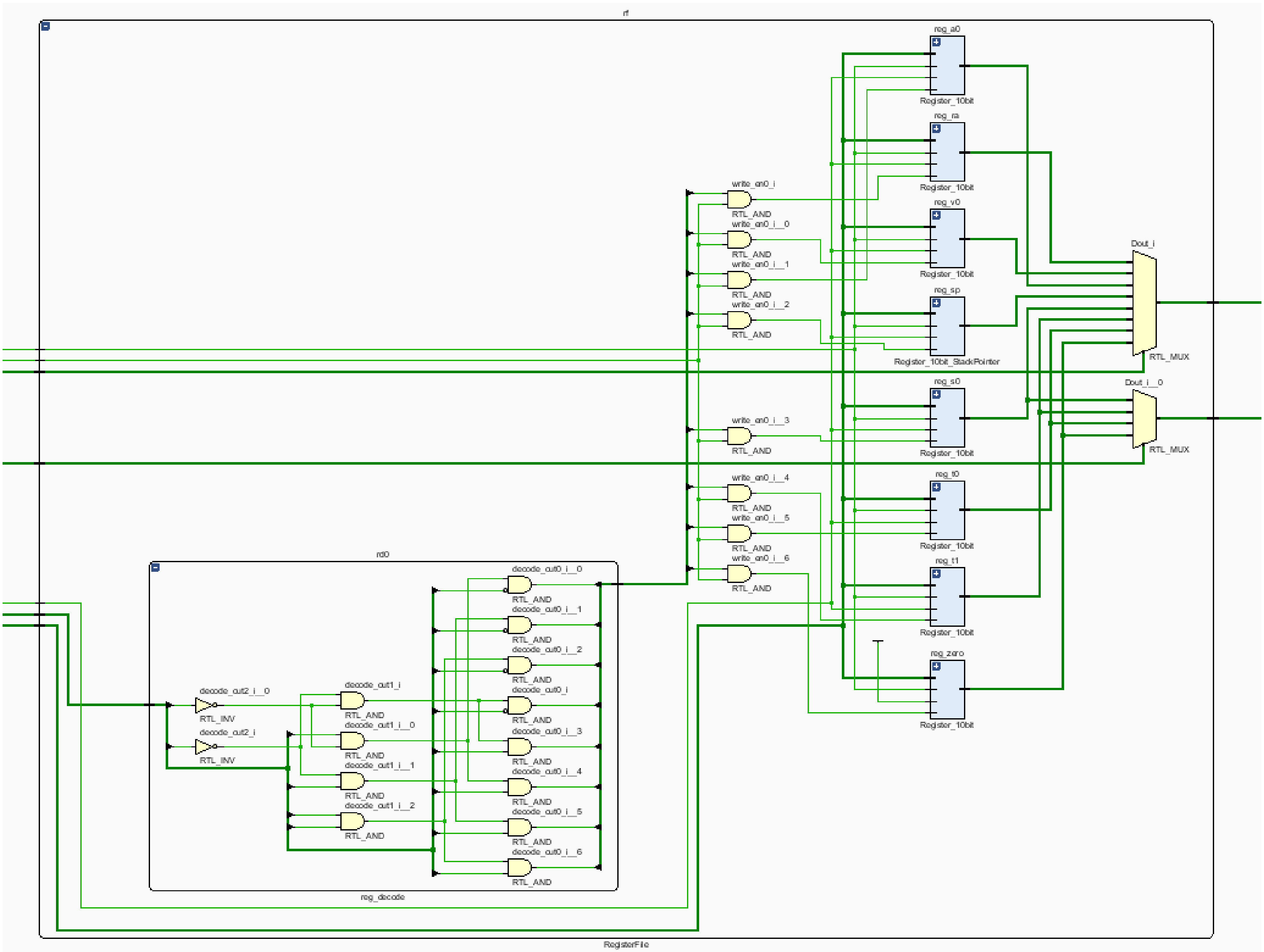
Registers		
Name	**Number**	**Use**
$Zero	0	The constant 0
$t0	1	Temporary
$t1	2	Temporary
$s0	3	Saved Temporary
$sp	4	Stack Pointer
$a0	5	Argument
$v0	6	Function Return
$ra	7	Return Address

Notes

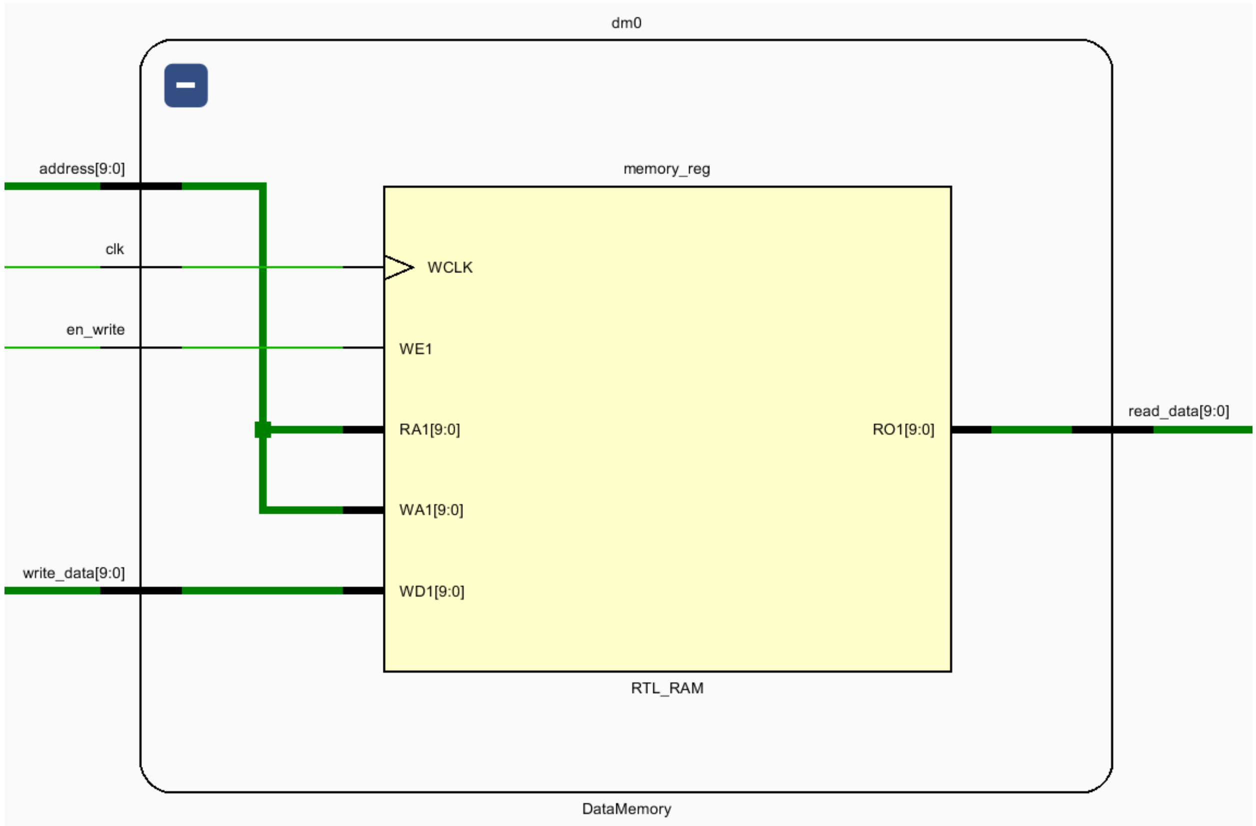
- Jump instructions can only jump 32 instructions at a time
- beq instruction only goes forward, min 1, max 7
- I type instruction immediate has max of 3 and min of -3
- $rs and $rt can only access registers 0 to 3. $rd can access all 8
- Load value from registers 4 to 7: add $rs, $rt, $rd -> $rs = $rt + $rd
- Store value into registers 4 to 7: sll $rd, $rs, $zero -> $rd = $rs << 0

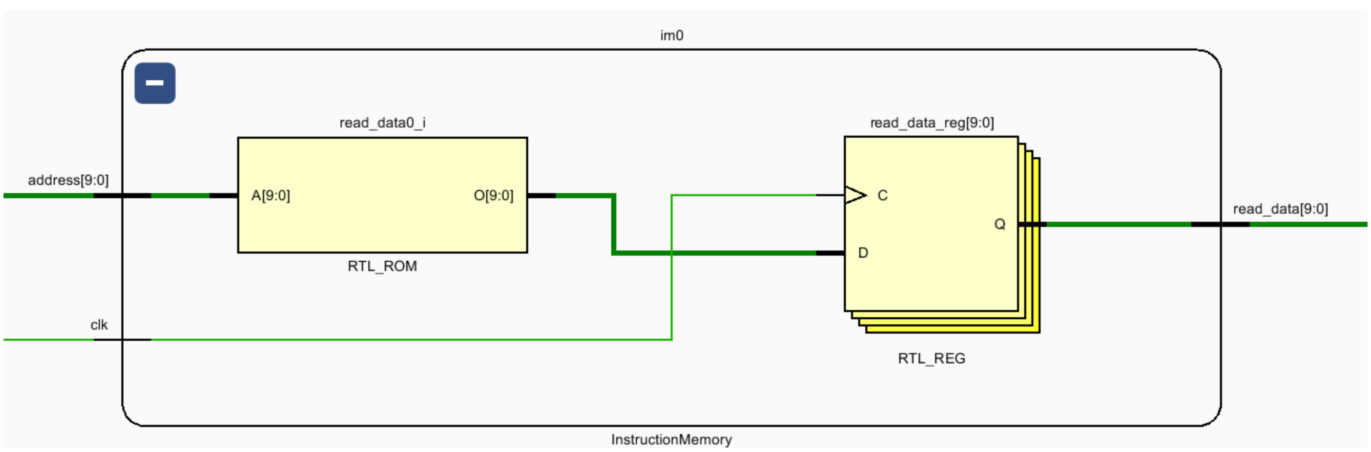
10-Bit Instruction Set Architecture
Datapath Schematic





rf
reg_a0
Register_10bit
reg_ra
Register_10bit
reg_v0
Register_10bit
reg_sp
Register_10bit_StackPointer
reg_s0
Register_10bit
reg_t0
Register_10bit
reg_t1
Register_10bit
reg_zero
Register_10bit
Dout_i
RTL_MUX
Dout_i__0
RTL_MUX
write_en0_i
RTL_AND
write_en0_i__0
RTL_AND
write_en0_i__1
RTL_AND
write_en0_i__2
RTL_AND
write_en0_i__3
RTL_AND
write_en0_i__4
RTL_AND
write_en0_i__5
RTL_AND
write_en0_i__6
RTL_AND
rd0
decode_out0_i__0
RTL_AND
decode_out0_i__1
RTL_AND
decode_out0_i__2
RTL_AND
decode_out0_i
RTL_AND
decode_out0_i__3
RTL_AND
decode_out0_i__4
RTL_AND
decode_out0_i__5
RTL_AND
decode_out0_i__6
RTL_AND
decode_out1_i
RTL_AND
decode_out1_i__0
RTL_AND
decode_out1_i__1
RTL_AND
decode_out1_i__2
RTL_AND
decode_out2_i__0
RTL_INV
decode_out2_i
RTL_INV
reg_decode
RegisterFile


dm0
memory_reg
address[9:0]
clk
en_write
write_data[9:0]
WCLK
WE1
RA1[9:0]
WA1[9:0]
WD1[9:0]
RO1[9:0]
read_data[9:0]
RTL_RAM
DataMemory

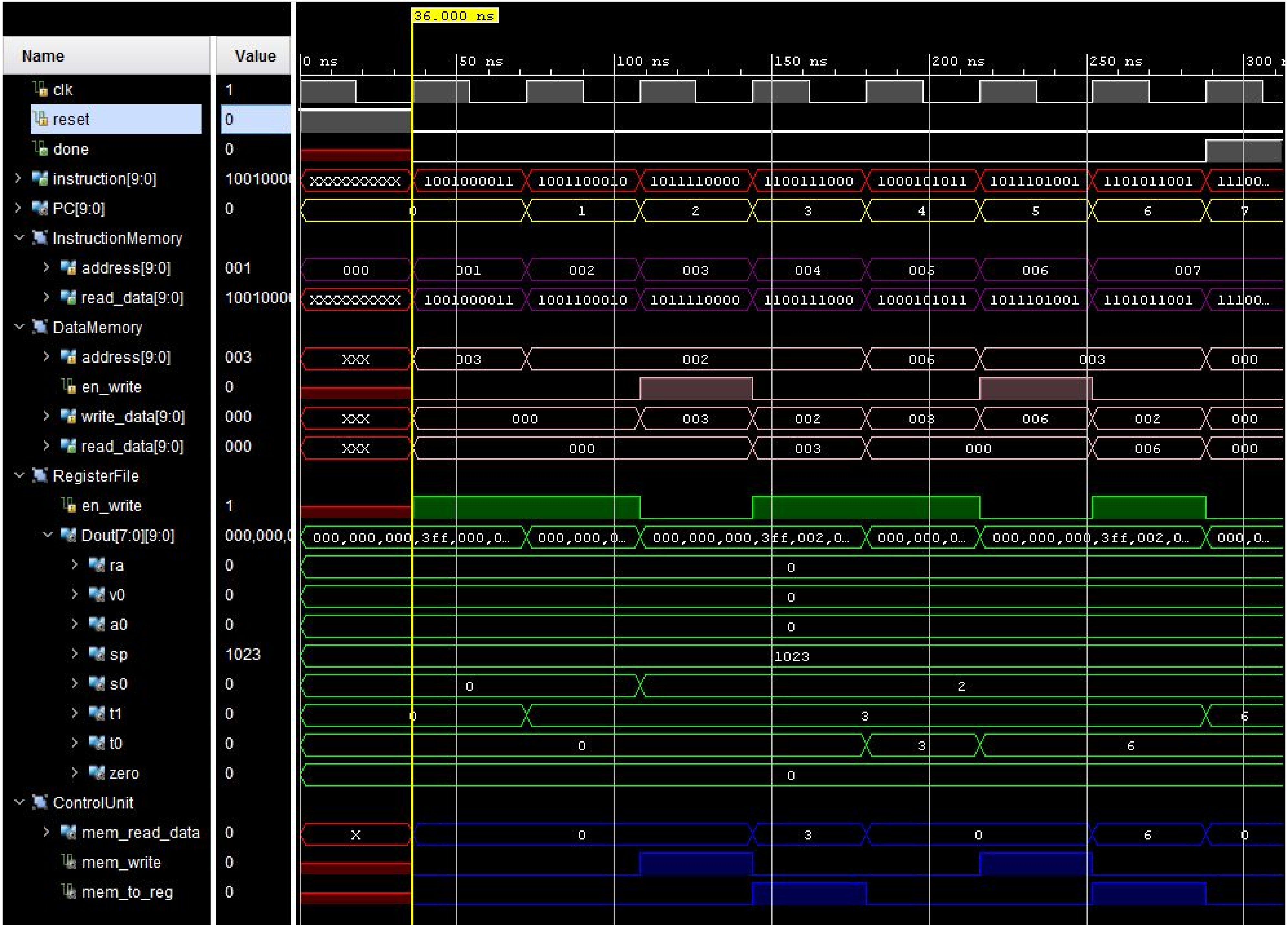

im0
read_data0_i
read_data_reg[9:0]
address[9:0]
A[9:0]
O[9:0]
C
D
Q
read_data[9:0]
clk
RTL_ROM
RTL_REG
InstructionMemory

```
; Tyler Thompson
; ECE3570 Lab3b
; mem[0] = arraysize -3
; mem[1] = array address
; mem[2] = i
; mem[3] = j
; mem[4] = 11

sw $zero, 3($zero)
lw $t0, 2($zero) // t0 = i
lw $t1, 0($zero) // t1 = n-3
addi $t1, $t1, 1 // t1 = n-2
cmp $s0, $t0, $t1  // i > n - 2 => 2
addi $t0, $t0, 1 // i++
sw $t0, 2($zero) // store i
addi $t0, $zero, 2
beq $s0, $t0, -2
j 2
halt

lw $t0, 3($zero) // t0 = j
lw $s0, 1($zero) // s0 = v
add $s0, $s0, $t0  // s0 = addr v[j]
lw $t1, 0($s0)  // t1 = v[j]
lw $s0, 1($s0) // s0 = v[j + 1]
cmp $t0, $t1, $s0  // v[j] > v[j+1] => 2
addi $s0, $zero, 2 // s0 = 2
beq $s0, $t0, -2
j 8

lw $t0, 3($zero) // t0 = j
lw $s0, 1($zero) // s0 = v
add $s0, $s0, $t0  // s0 = addr v[j]
lw $t1, 0($s0)  // t1 = v[j]
lw $t0, 1($s0) // s0 = v[j + 1]
sw $t1, 1($s0) // v[j+1] = v[j]
sw $t0, 0($s0) // v[j] = v[j+1]

lw $t0, 3($zero) // t0 = j
lw $t1, 0($zero) // t1 = n-3
cmp $s0, $t0, $t1  // i > n - 2 => 2
addi $t0, $t0, 1 // j++
sw $t0, 3($zero) // store j
addi $t0, $zero, 2
beq $s0, $t0, 0
addi $t0, $zero, 3
lw $t1, 1($t0) // t1 = 10
jr $t1 // j l2
jr $zero // j l1
```

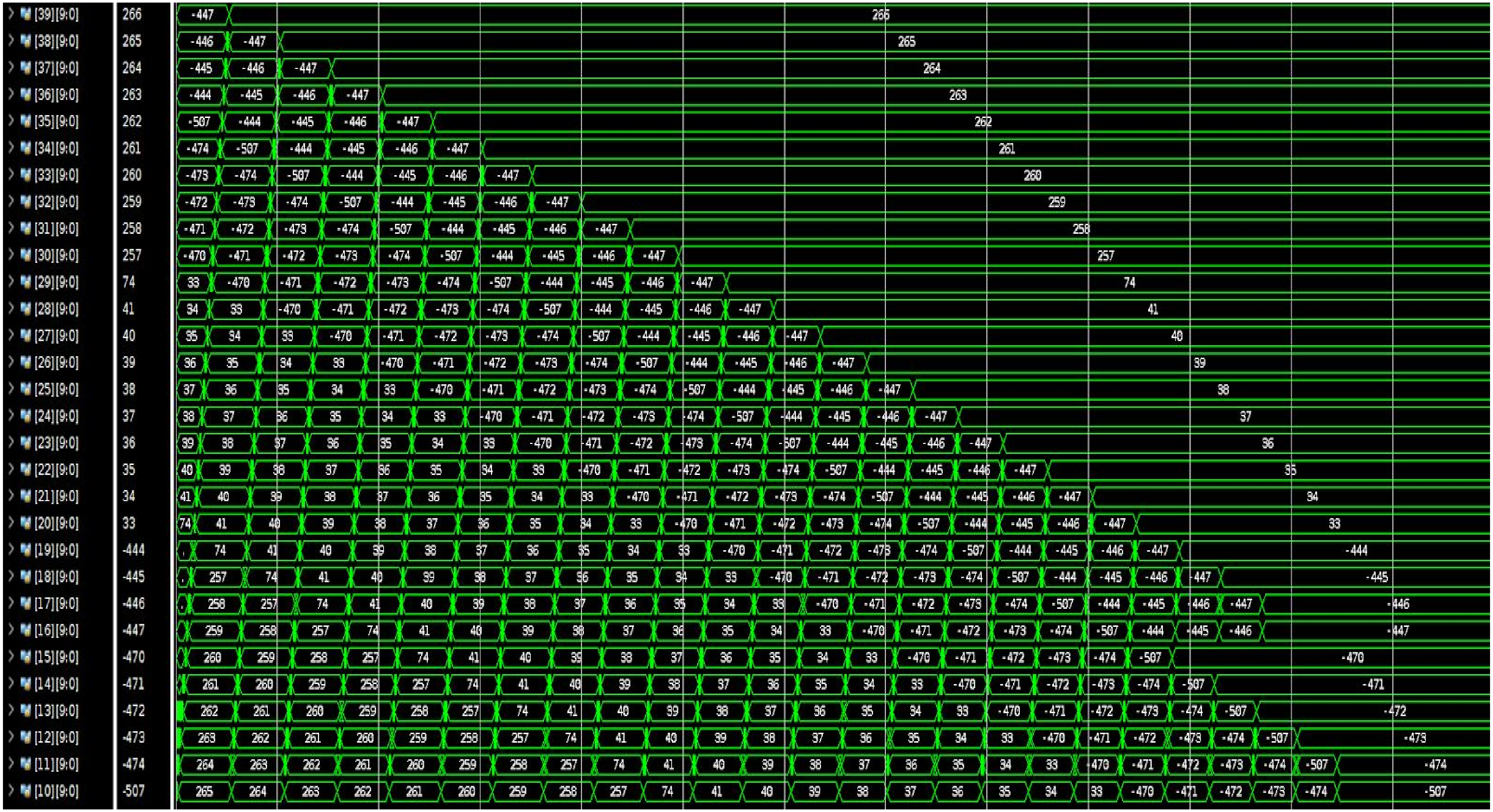
```
; Tyler Thompson
; ECE3570 Lab3b
; 3/13/2018
; Program 2 re-written for Lab3b
; F = ( X*Y ) - 4
; Assume X is in address 0 of data memory
; Assume Y is in address 1 of data memory
; F will be placed in $v0 at the end of the program


1101000000  lw $t1, 0($zero) ;$t1=x
0011000101      sll $a0, $t1, $zero     ;store original x in $a0
1001010111  addi $t1, $t1, -1    ;x--
1101100001  lw $s0, 1($zero) ;$s0=y
0000111000      add $t0, $s0, $zer0     ;$t0=4

0111000000      beq $t1, $zer0, 0       ;if x==0, pc=pc+4
0001111001      add $s0, $s0, $t0       ;y=y+4
1001010111      addi $t1, $t1, -1       ;x--
1111110101      j -3                    ;pc=pc-3

0001000101      add $t1, $zero, $a0     ;load original x
0101000010      cmp $t1, $t1, $zer0
1000100010      addi $t0, $zero, 1
0110110110      beq $t0, $t1, -2        ;if x is pos, pc=pc+2
1001111100      tcp $s0, $s0            ;two comp of y
1001111110      addi $s0, $s0, -2
1001111110      addi $s0, $s0, -2       ;y=y-4
0011100110      sll $v0, $s0, $zero     ;f=(x*y)-4

1110000011      halt
```

```
; Tyler Thompson
; ECE3570 Lab3b
; 3/13/2018
; Program 3 re-written for Lab3b
; assume data memory address 0 point to base address of A
; assume data memory address 1 point to base address of new_A
; The number 10 is returned in $v0 at the end of the program

1100100000      lw $t0, 0($zero) ;t0 = A address
1101000001      lw $t1, 1($zero) ;t1 = new_A address

1101101000      lw $s0, 0($t0) ;load A[i]
0111100001      beq $s0 $zero, 1 ;branch to finish if araay reaches end
1011011000      sw $s0, 0($t1) ;store new_A[i]
1000101001      addi $t0, $t0, 1 ;increment pointers
1001010001      addi $t1, $t1, 1
1111101101      j -5

1000100011      addi $t0, $zero, 3
1000101011      addi $t0, $t0, 3
1000101011      addi $t0, $t0, 3
1000101001      addi $t0, $t0, 1 ;t0 = 10
0010100110      sll $v0, $t0, $zero ;v0 = 10
1110000011      halt
```

Instruction and Data Memory Test

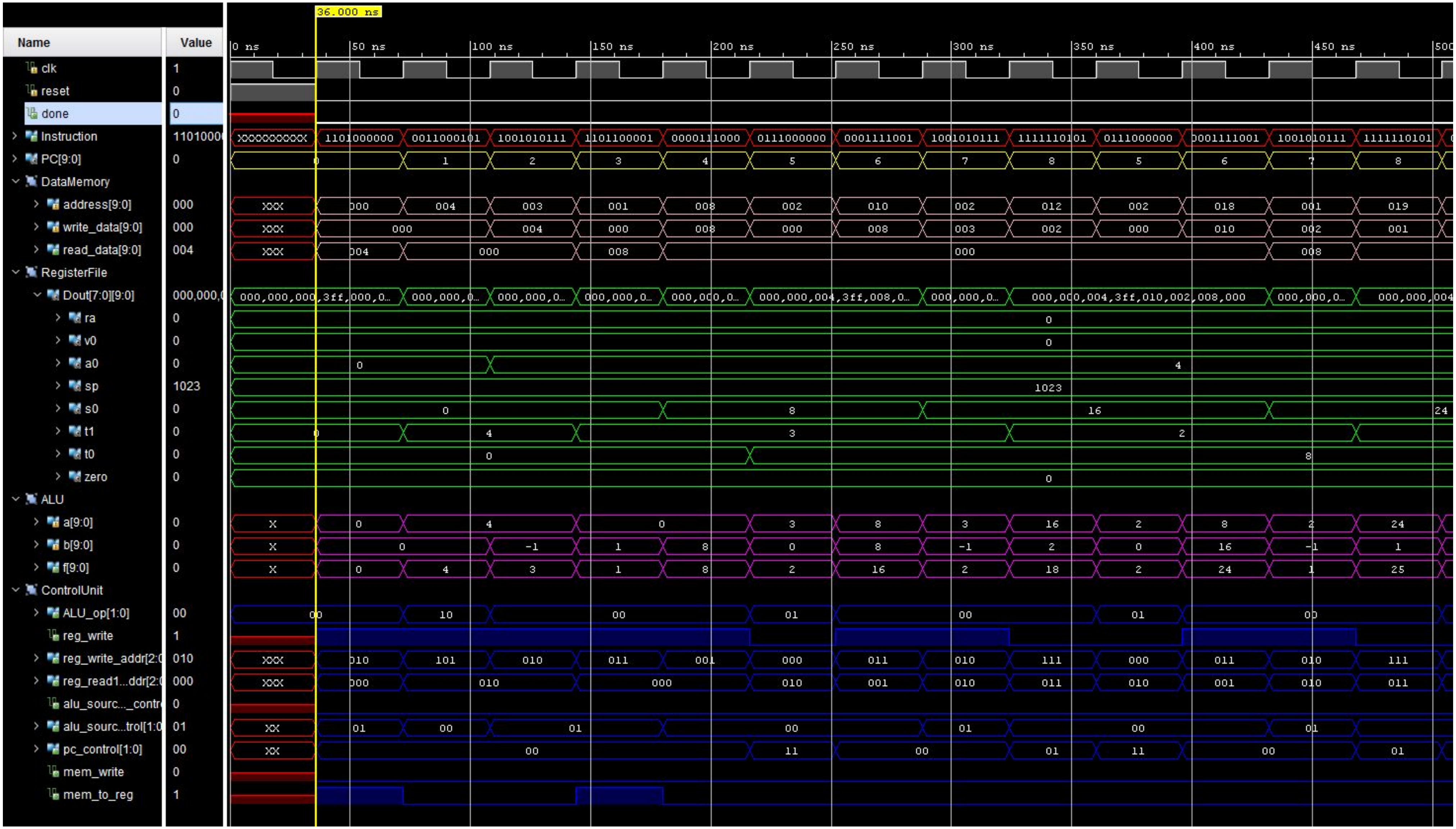


Program 1



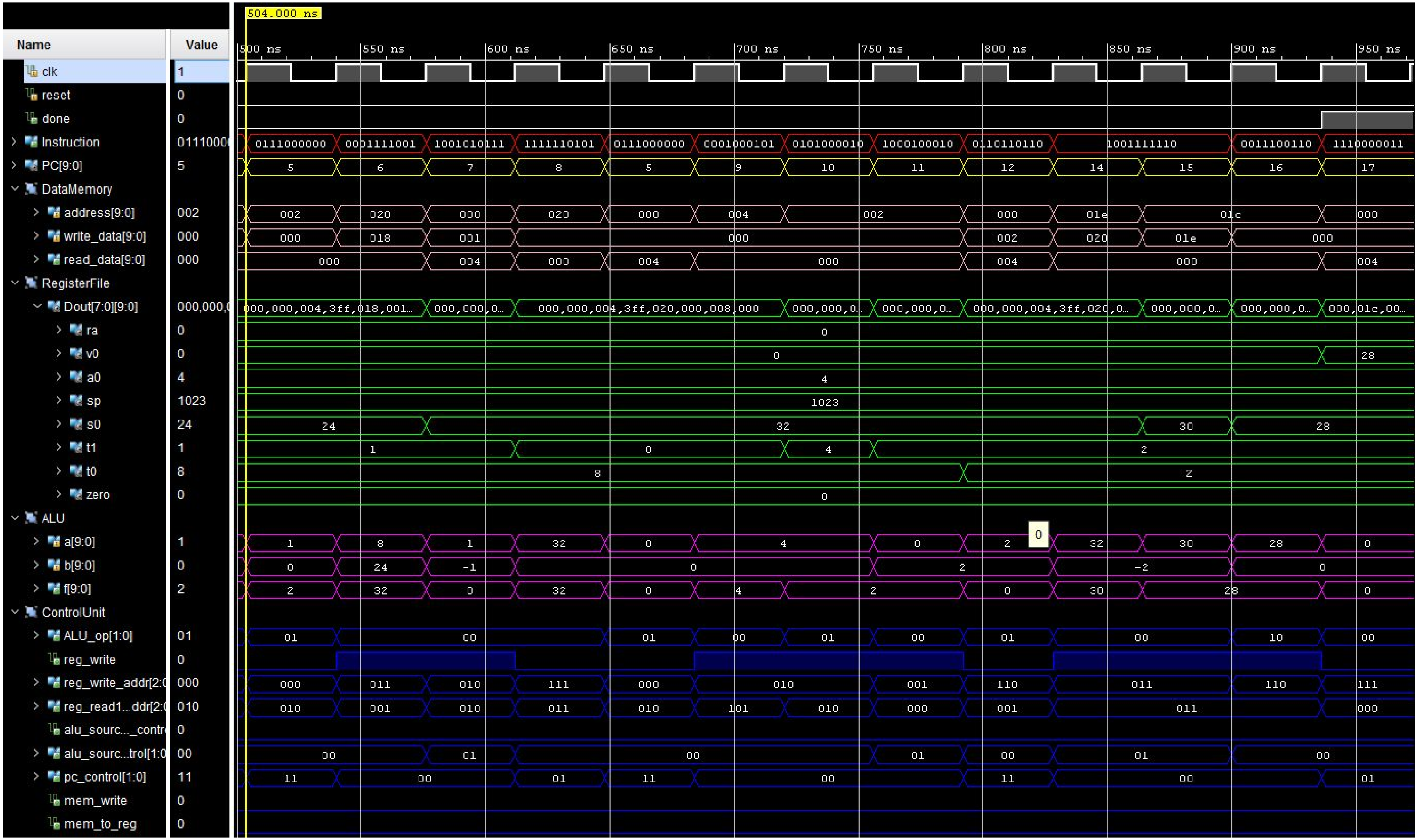
Program 2 (1 of 2)

X = 4, Y = 8, F = 28

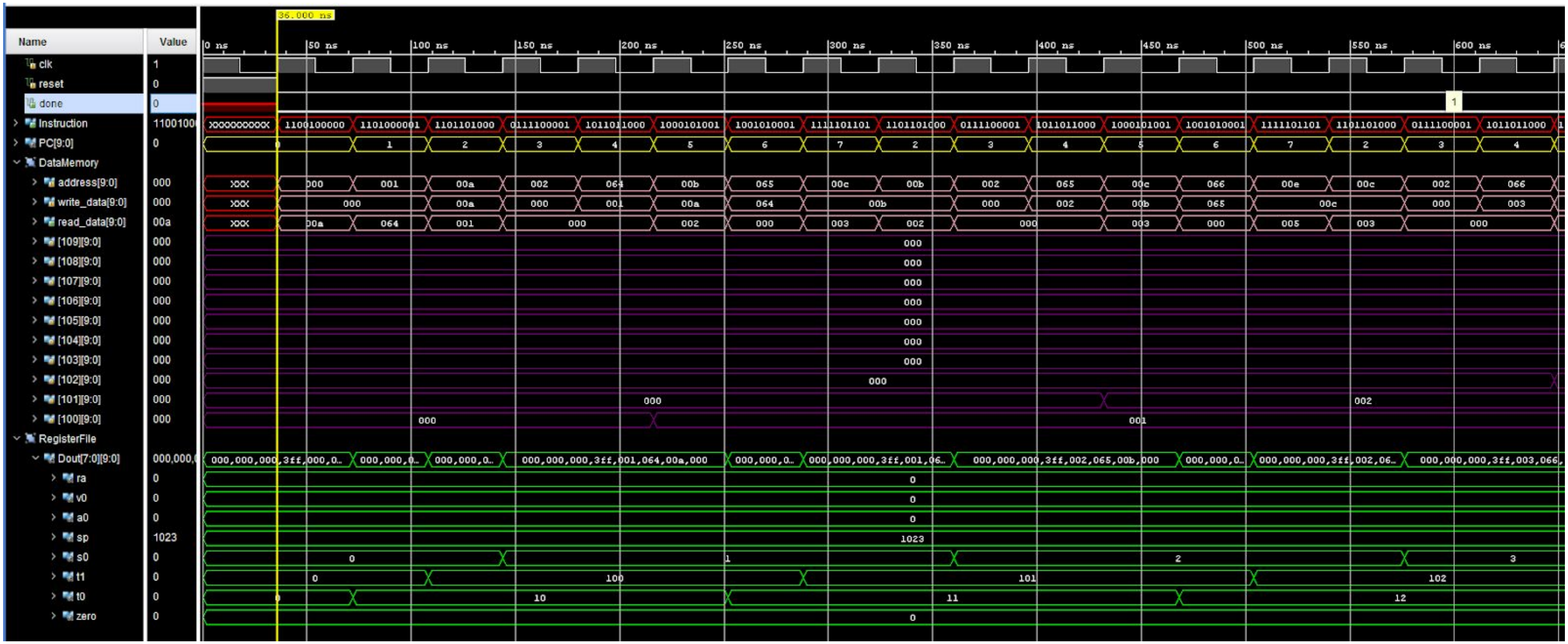


Program 2 (2 of 2)

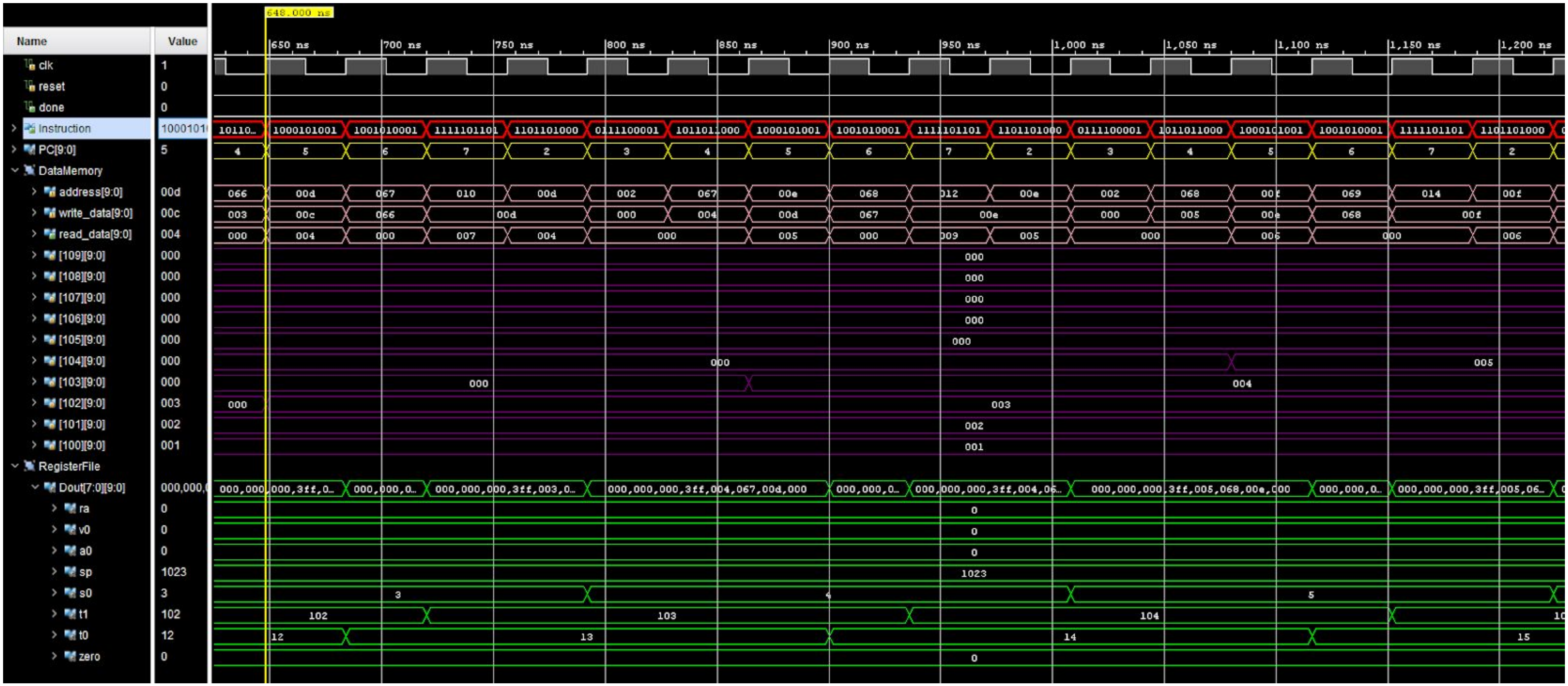
X = 4, Y = 8, F = 28



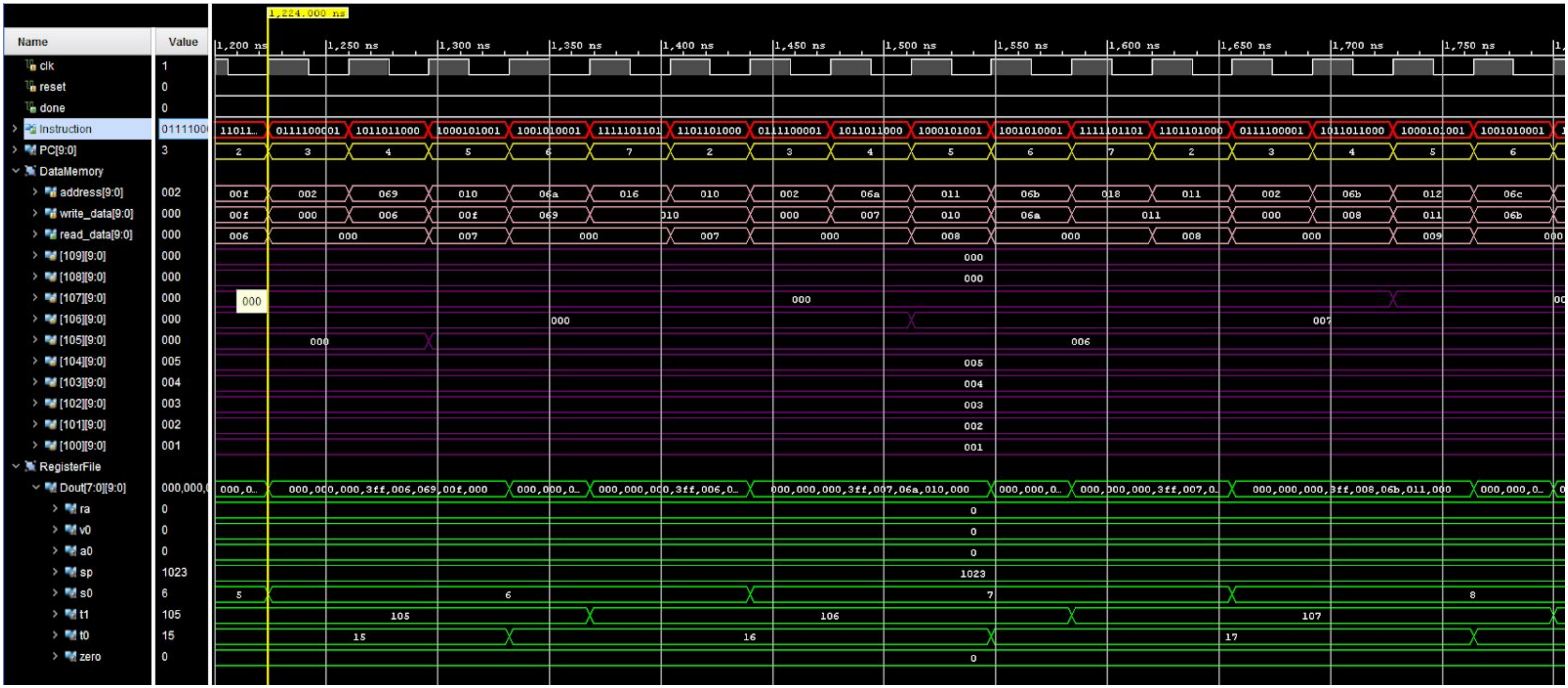
Program 3 (1 of 5)



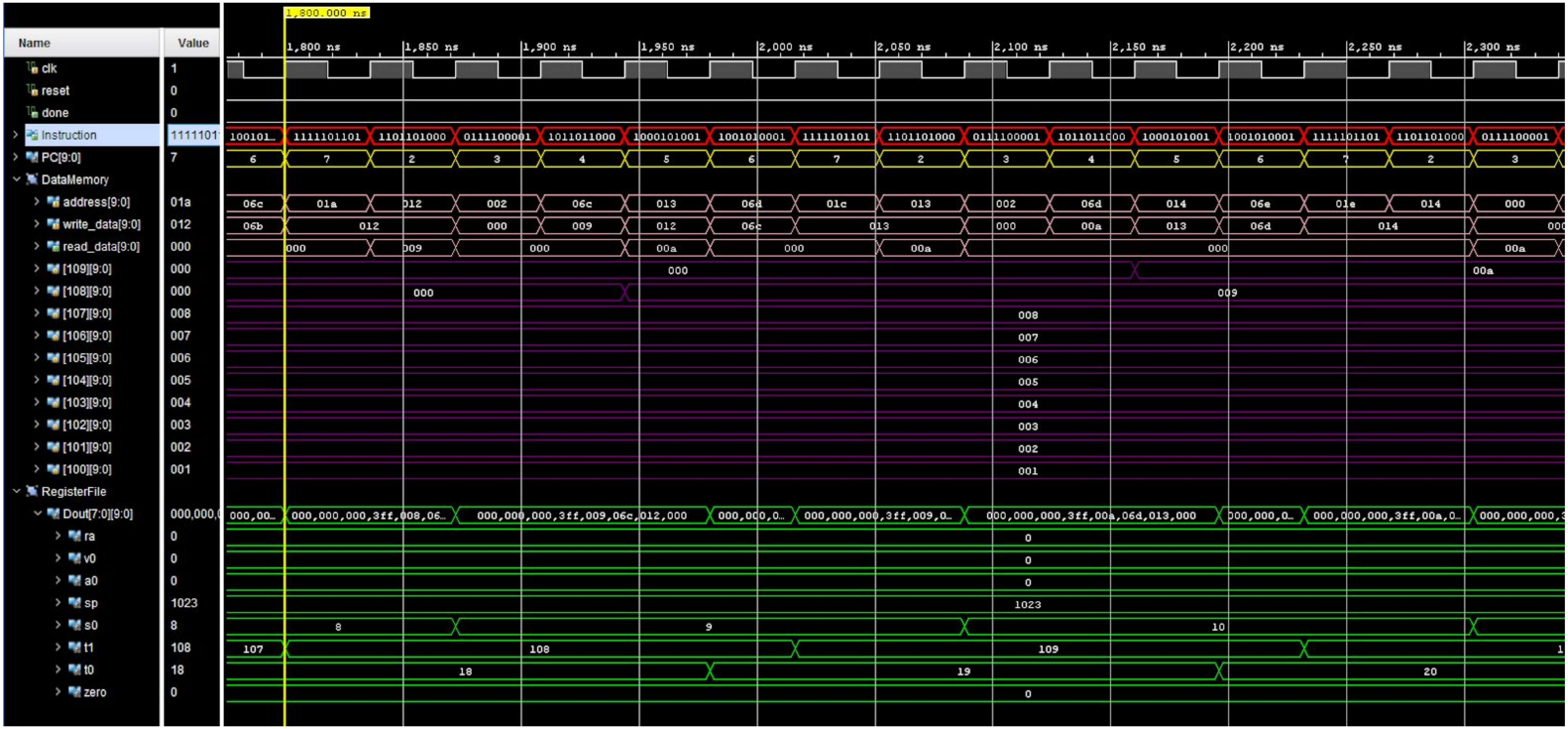
Program 3 (2 of 5)



Program 3 (3 of 5)



Program 3 (4 of 5)



Program 3 (5 of 5)