Tyler Thompson
An Nguyen

ECE 3570 Lab 4
10-bit Instruction Set Architecture CPU - Pipelined CPU
April 4th, 2018

Critical Path Delay

Fetch/Decode Stage: 7.61ns
Execute/Memory Stage: 13.40ns
Writeback Stage: 4.92ns

Answers to Design Questions

A. The main registers in the register file were changed to write at the first half of the clock cycle and read at the second half. A module was added in the fetch/decode stage so that branch instructions resolve in the fetch/decode stage.

B. The pipeline stage with the longest latency is the execute/memory stage with a latency of 13.40ns. Because of this the cycle time and minimum clock period of the CPU is **14ns**.

C. A module was added in the fetch/decode stage so that branch instructions resolve in the fetch/decode stage instead of the writeback stage. The addition of the forwarding unit allows all instructions to execute without the need of nops.

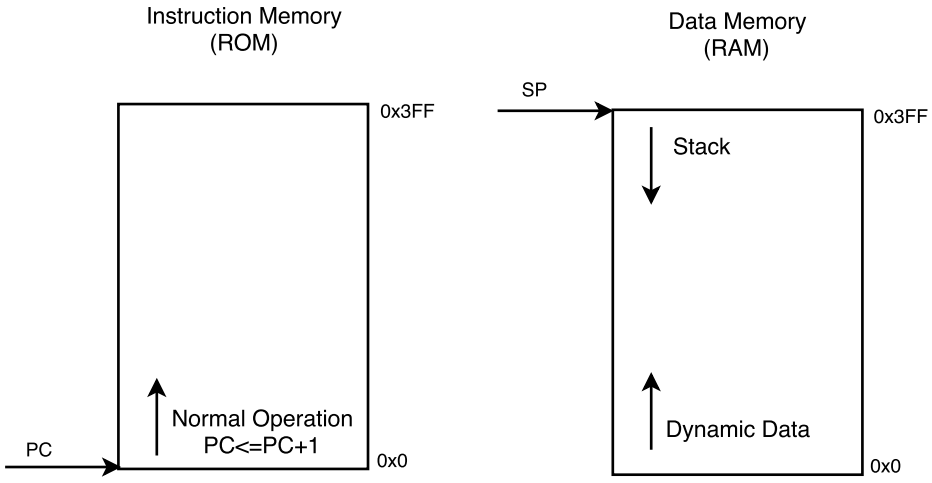
D. Program 1 - (19n^2) -9 (Assuming array starts in the worst case of being sorted backwards. I.e. n = 10, IC = 1891)

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

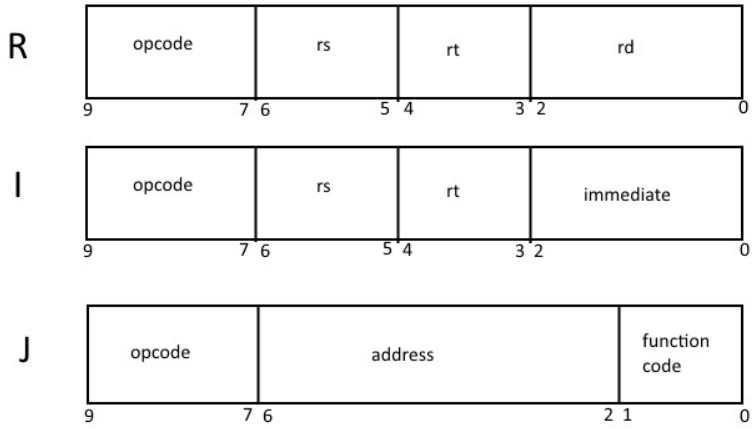
   Program 3 - 10 + 6 * Size Of Array (i.e. Array Size = 10, IC = 70)

E. Pipelined Clock Cycle Time = 14ns
   Non-Pipelined Clock Cycle Time = 18ns
   Speedup = 1.29

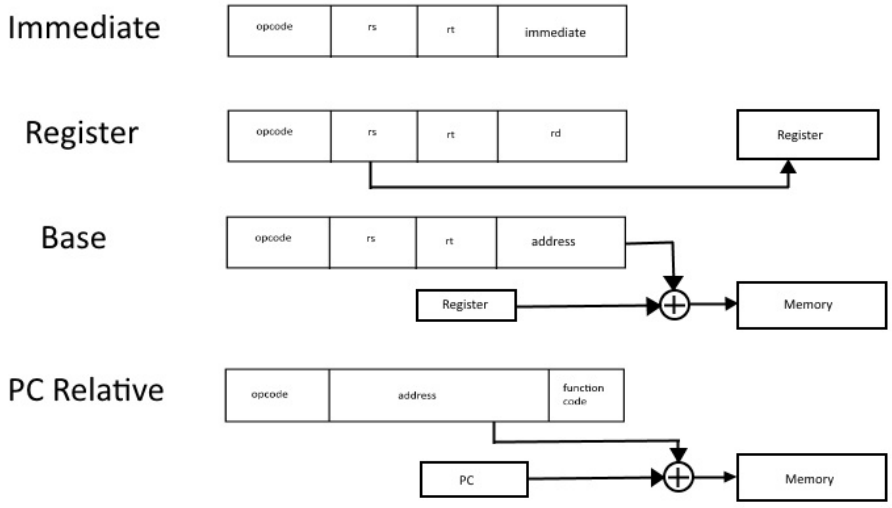
Memory Allocation



Instruction Formats



Addressing Modes



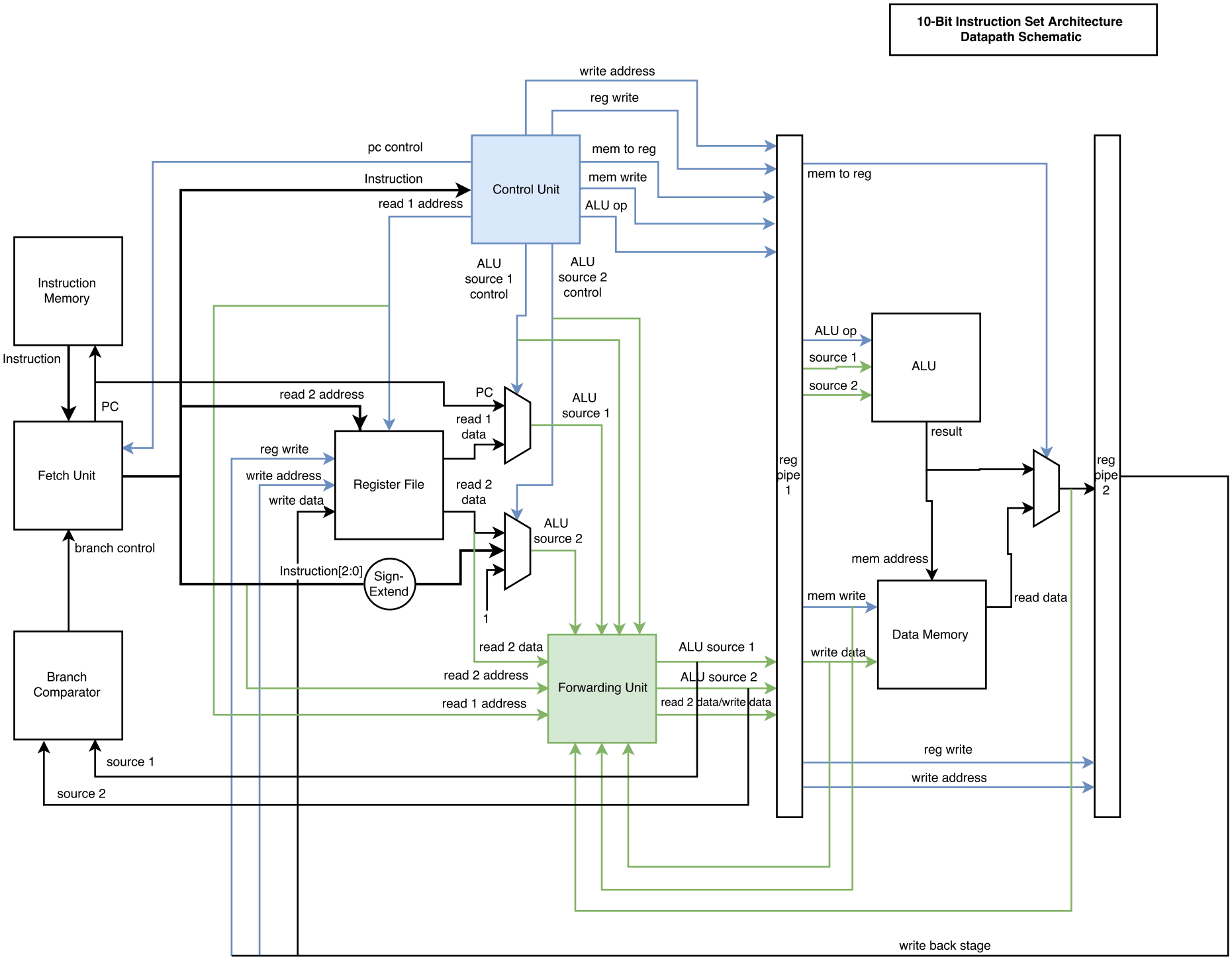
Registers		
Name	**Number**	**Use**
$Zero	0	The constant 0
$t0	1	Temporary
$t1	2	Temporary
$s0	3	Saved Temporary
$sp	4	Stack Pointer
$a0	5	Argument
$v0	6	Function Return
$ra	7	Return Address

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 ISA Instruction Reference Sheet

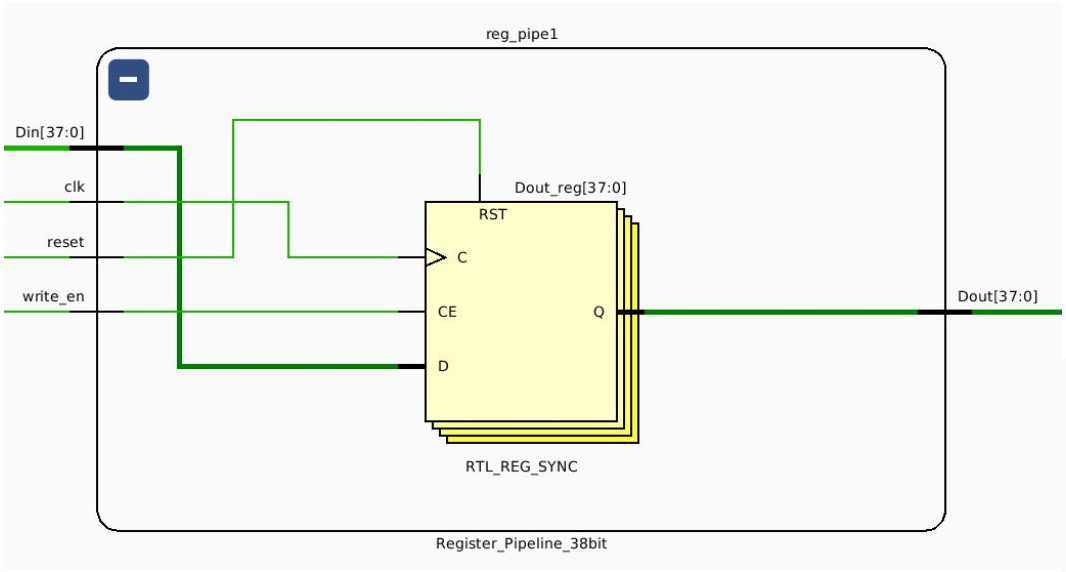
Instructions						
Name	**Description**	**Format**	**RTL**	**Syntax**	**OP Code (hex)**	**Function Code (hex)**
add	Add two values in registers together	R	if MEM[PC] == ADD rs rt rd [rs] <= [rt] + [rd] PC <= PC + 1	add $rs, $rt, $rd	0x0	
addi	Add a 3-bit signed contant to a value in a register	I	if MEM[PC] == 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

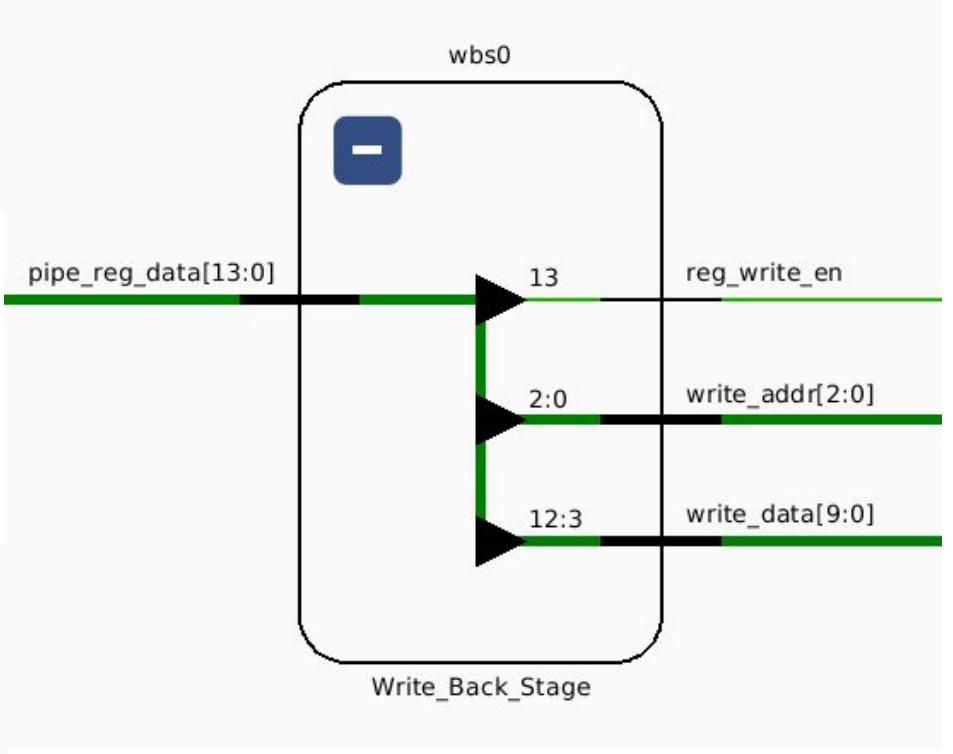

10-Bit Instruction Set Architecture
Datapath Schematic
write address
reg write
pc control
Instruction
read 1 address
Control Unit
mem to reg
mem write
ALU op
ALU
source 1
control
ALU
source 2
control
Instruction
Memory
Instruction
PC
read 2 address
PC
read 1
data
ALU
source 1
reg write
write address
write data
Register File
read 2
data
ALU
source 2
Fetch Unit
branch control
Instruction[2:0]
Sign-
Extend
1
Branch
Comparator
read 2 data
read 2 address
read 1 address
Forwarding Unit
ALU source 1
ALU source 2
read 2 data/write data
source 1
source 2
reg
pipe
1
mem to reg
ALU op
source 1
source 2
ALU
result
mem address
mem write
write data
Data Memory
read data
reg
pipe
2
reg write
write address
write back stage

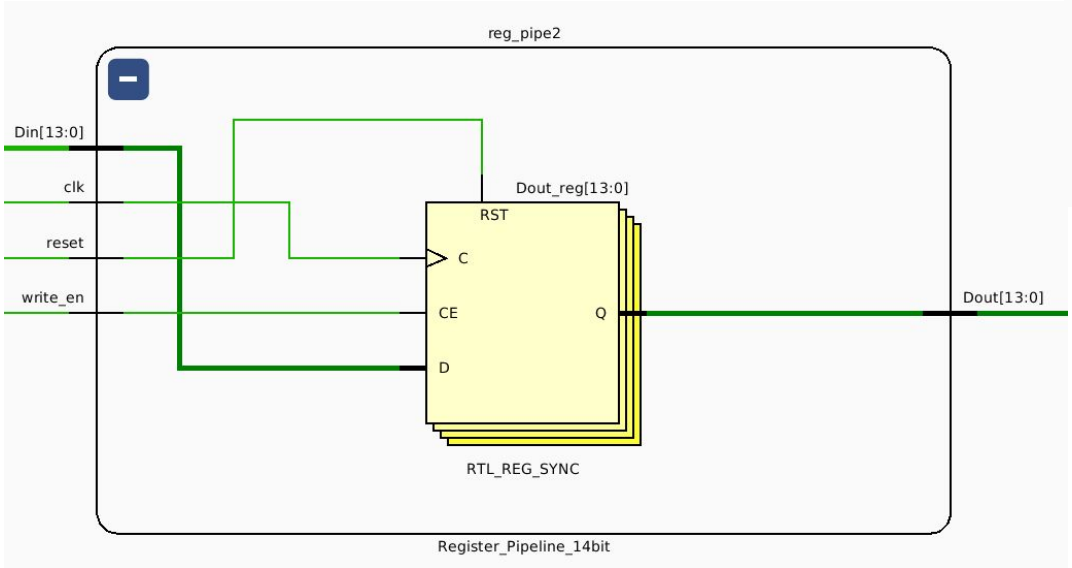
```
; Tyler Thompson
; ECE3570 Lab3b
; mem[0] = arraysize -3
; mem[1] = array address
; mem[2] = i
; mem[3] = j
; mem[4] = 11

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
```

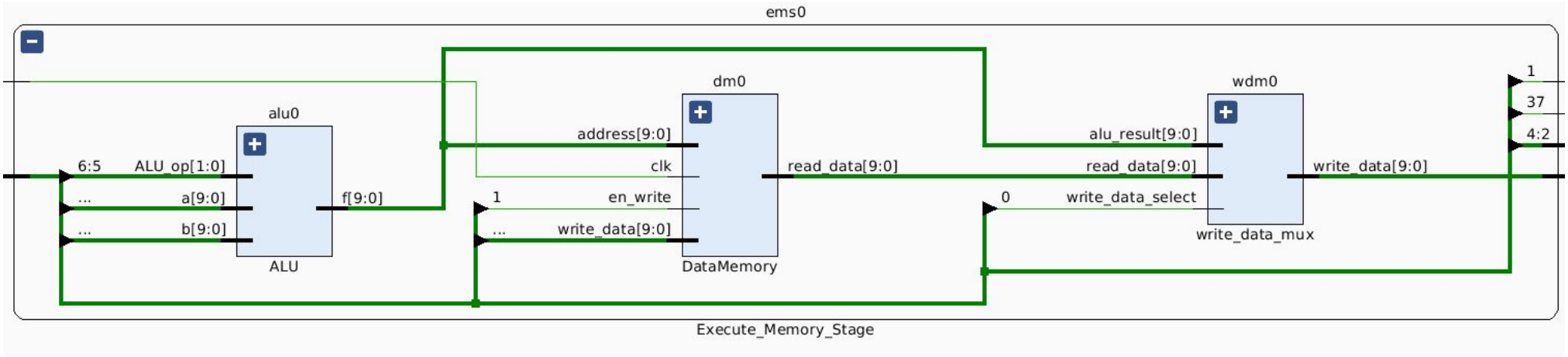
```
; Tyler Thompson
; ECE3570 Lab3b
; 3/13/2018
; Program 2 re-written for Lab3b
; F = ( X*Y ) - 4
; Assume X is in address 0 of data memory
; Assume Y is in address 1 of data memory
; F will be placed in $v0 at the end of the program


1101000000      lw $t1, 0($zero)        ;$t1=x
0011000101      sll $a0, $t1, $zero     ;store original x in $a0
0101000001      cmp $t0, $t1, $zero     ;test if x is negative
1001100010      addi $s0, $zero, 2
0111101110      beq $s0, $t0, -2        ;if x is negative
1001010100      tcp, $t1, $t1           ;make x positive
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
```

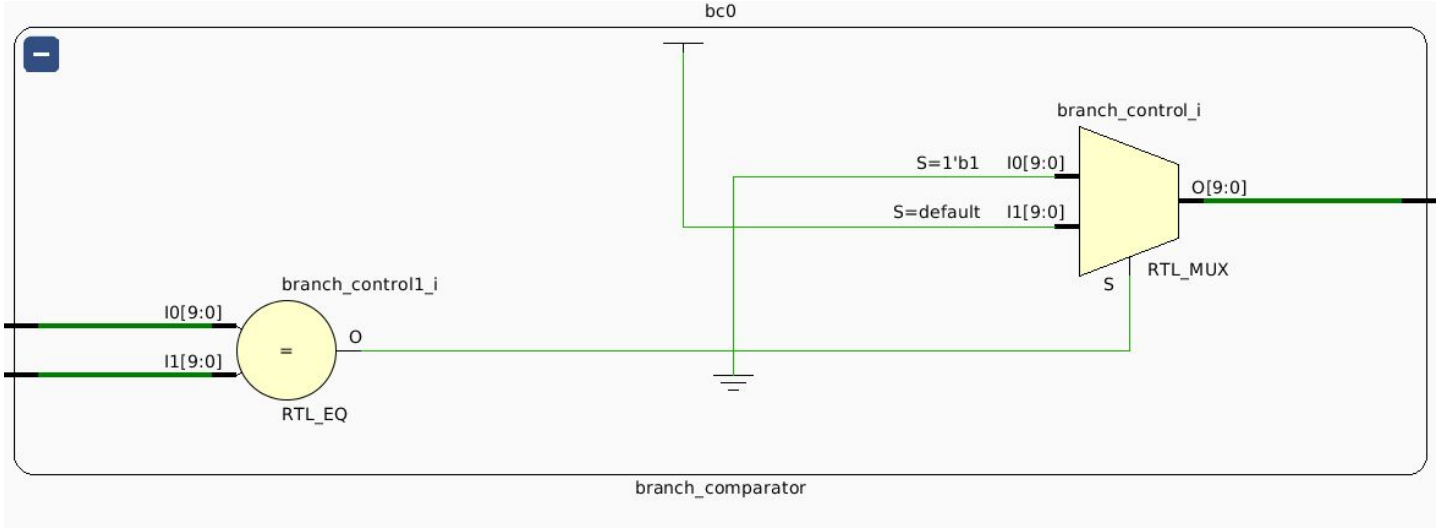
```
; Tyler Thompson
; ECE3570 Lab3b
; 3/13/2018
; Program 3 re-written for Lab3b
; assume data memory address 0 point to base address of A
; assume data memory address 1 point to base address of new_A
; The number 10 is returned in $v0 at the end of the program

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
```

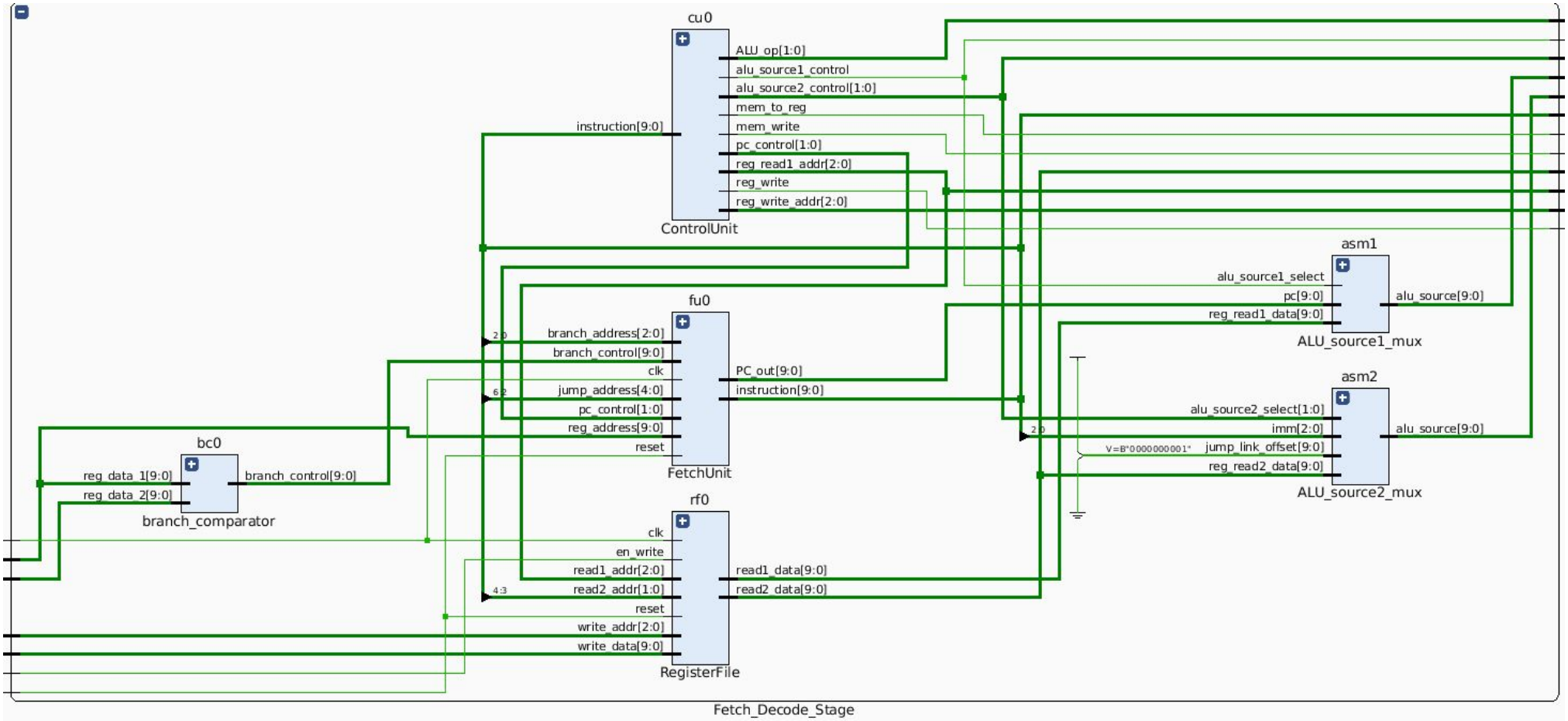

reg_pipe1
Din[37:0]
clk
reset
write_en
Dout_reg[37:0]
RST
C
CE
D
Q
Dout[37:0]
RTL_REG_SYNC
Register_Pipeline_38bit

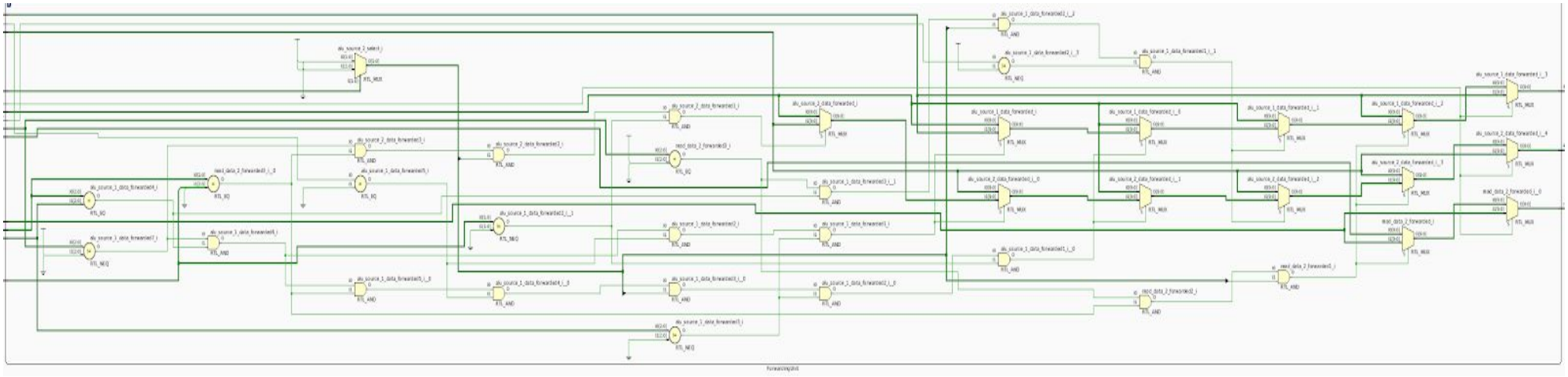

wbs0
pipe_reg_data[13:0]
13
reg_write_en
2:0
write_addr[2:0]
12:3
write_data[9:0]
Write_Back_Stage

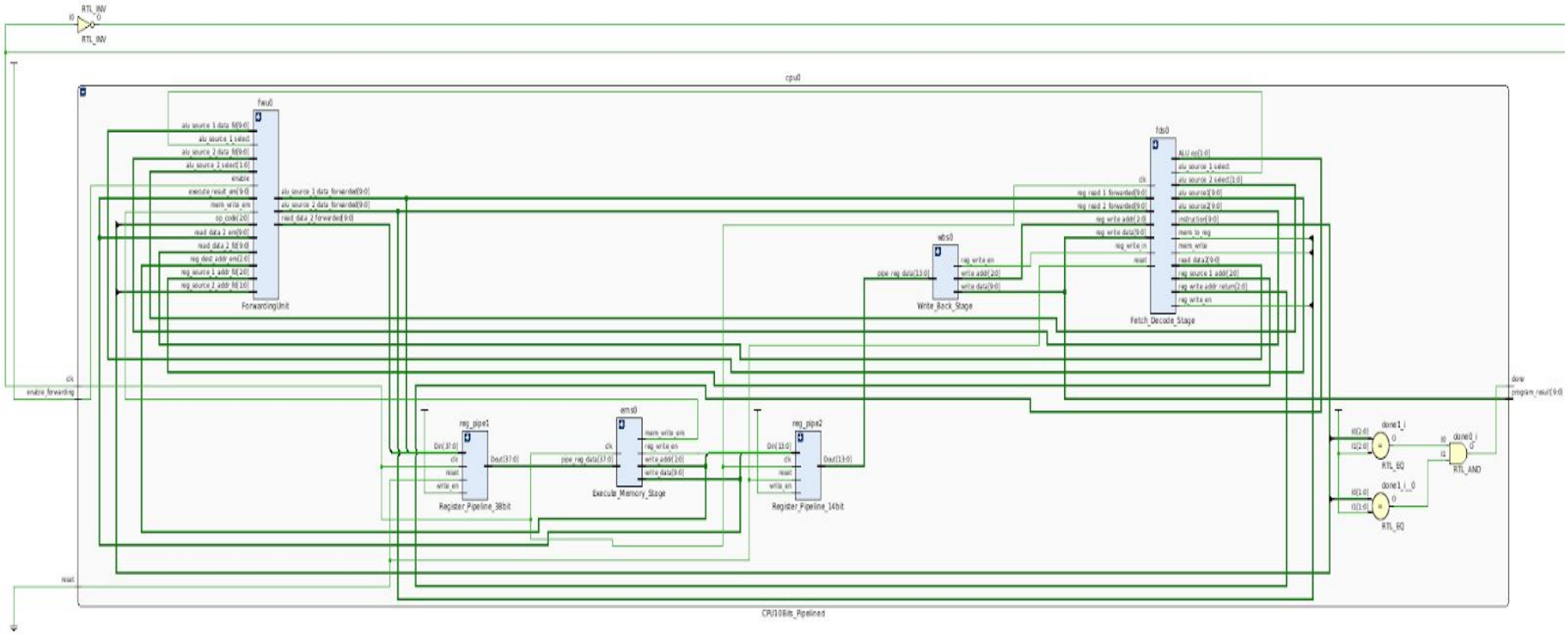

reg_pipe2
Din[13:0]
clk
reset
write_en
Dout_reg[13:0]
RST
C
CE
D
Q
Dout[13:0]
RTL_REG_SYNC
Register_Pipeline_14bit

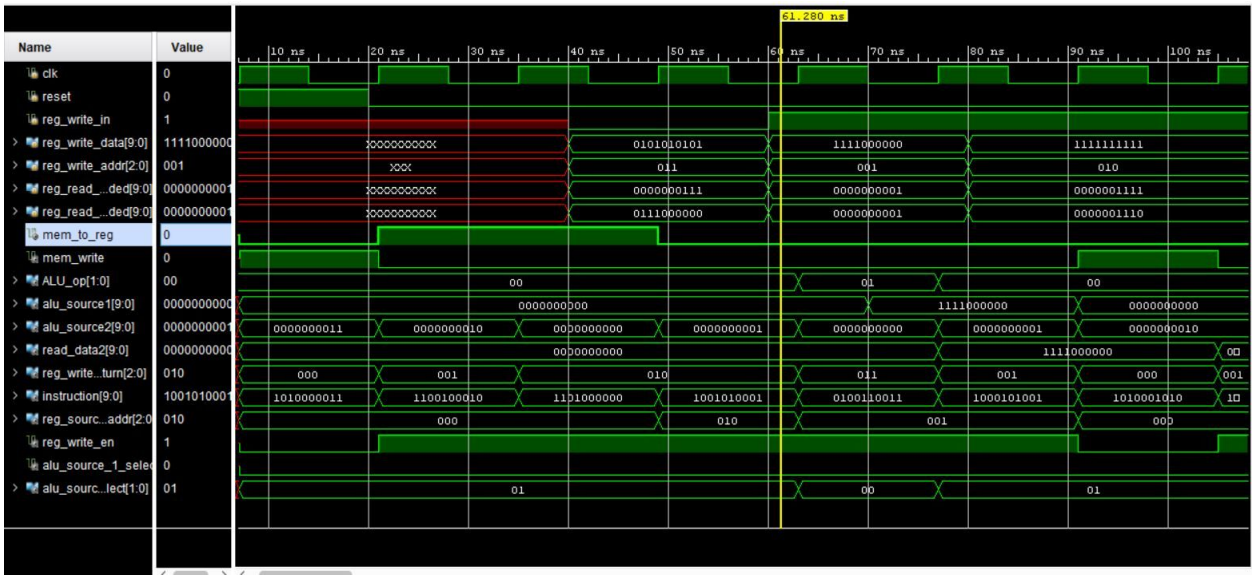

ems0
alu0
6:5
ALU_op[1:0]
...
a[9:0]
...
b[9:0]
ALU
f[9:0]
dm0
address[9:0]
clk
1
en_write
...
write_data[9:0]
DataMemory
read_data[9:0]
wdm0
alu_result[9:0]
read_data[9:0]
0
write_data_select
write_data_mux
write_data[9:0]
1
37
4:2
Execute_Memory_Stage


bc0
branch_control_i
S=1'b1
I0[9:0]
S=default
I1[9:0]
O[9:0]
S
RTL_MUX
branch_control1_i
I0[9:0]
I1[9:0]
=
O
RTL_EQ
branch_comparator

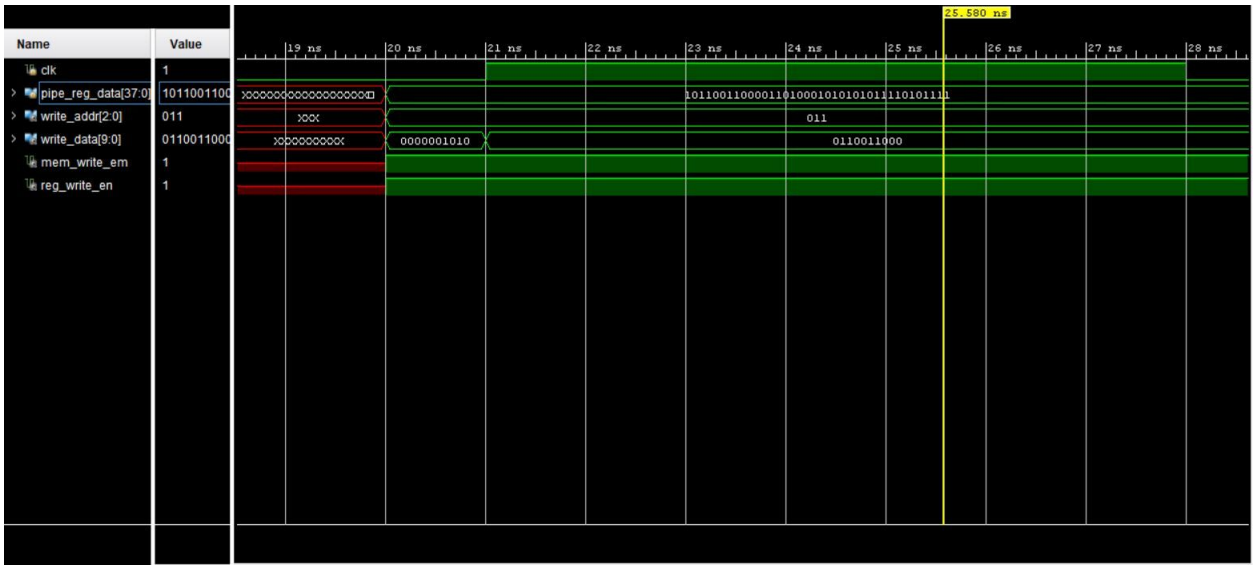

cu0
ALU_op[1:0]
alu_source1_control
alu_source2_control[1:0]
mem_to_reg
mem_write
pc_control[1:0]
reg_read1_addr[2:0]
reg_write
reg_write_addr[2:0]
instruction[9:0]
ControlUnit
asm1
alu_source1_select
pc[9:0]
reg_read1_data[9:0]
alu_source[9:0]
ALU_source1_mux
fu0
branch_address[2:0]
branch_control[9:0]
clk
jump_address[4:0]
pc_control[1:0]
reg_address[9:0]
reset
PC_out[9:0]
instruction[9:0]
FetchUnit
asm2
alu_source2_select[1:0]
imm[2:0]
V=B"0000000001"
jump_link_offset[9:0]
reg_read2_data[9:0]
alu_source[9:0]
ALU_source2_mux
bc0
reg_data_1[9:0]
reg_data_2[9:0]
branch_control[9:0]
branch_comparator
rf0
clk
en_write
read1_addr[2:0]
read2_addr[1:0]
reset
write_addr[2:0]
write_data[9:0]
read1_data[9:0]
read2_data[9:0]
RegisterFile
Fetch_Decode_Stage



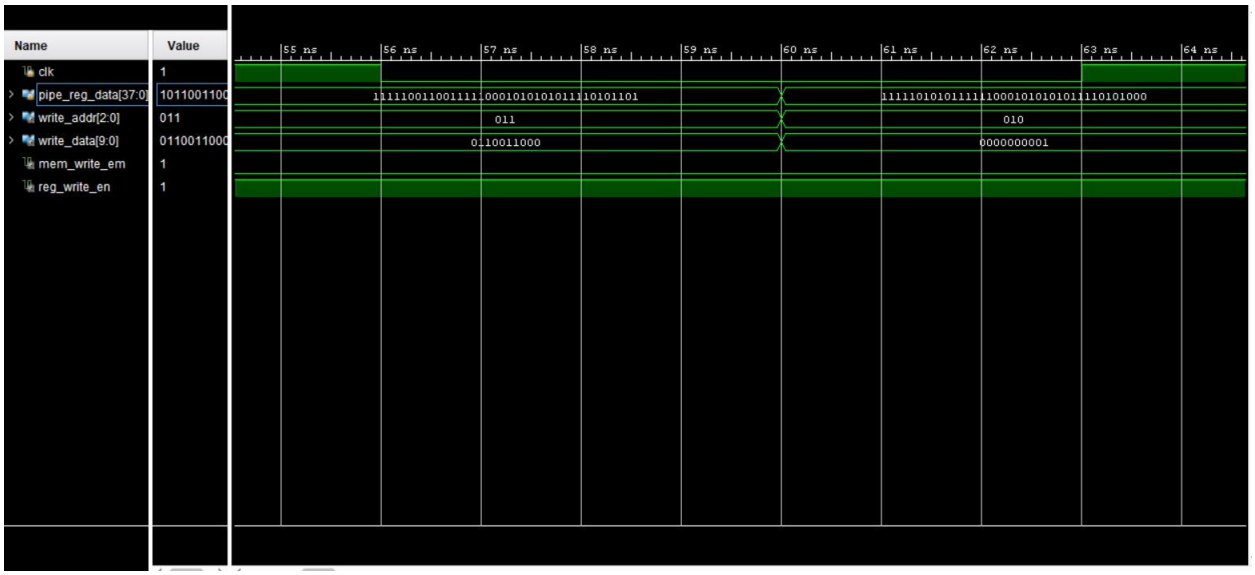

RTL_INV
RTL_INV
cpu0
fwu0
ForwardingUnit
fds0
Fetch_Decode_Stage
wbs0
Write_Back_Stage
reg_pipe1
Register_Pipeline_38bit
ems0
Execute_Memory_Stage
reg_pipe2
Register_Pipeline_14bit
done1_i
RTL_EQ
done0_i
RTL_AND
done1_i_0
RTL_EQ
clk
enable_forwarding
reset
done
program_result[9:0]
CPU10Bit_Pipelined



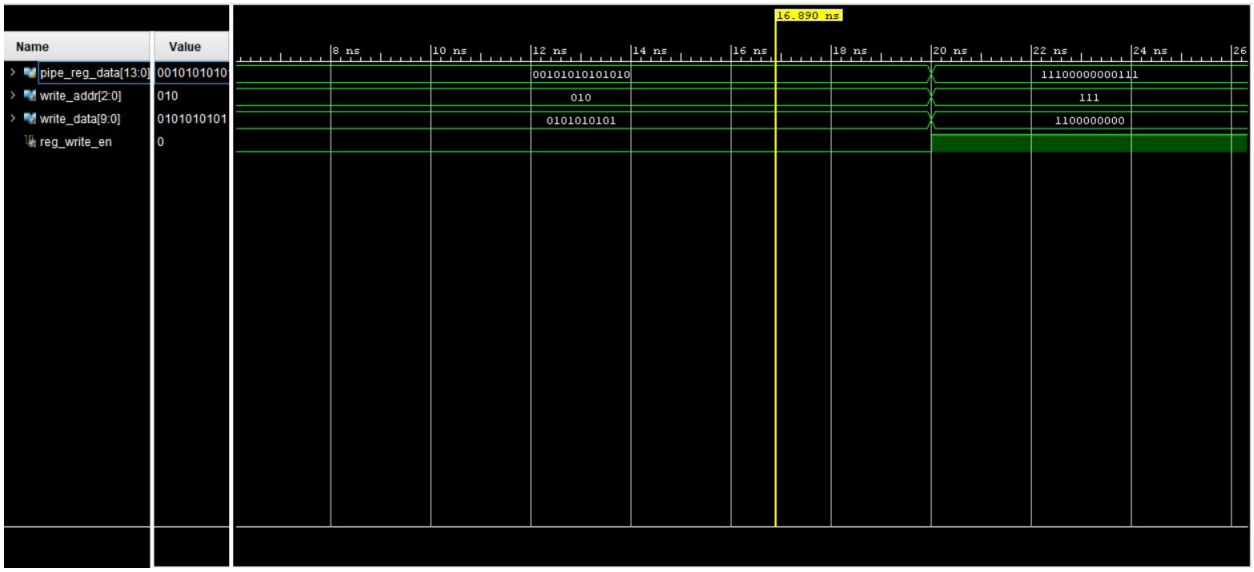
Fetch Decode



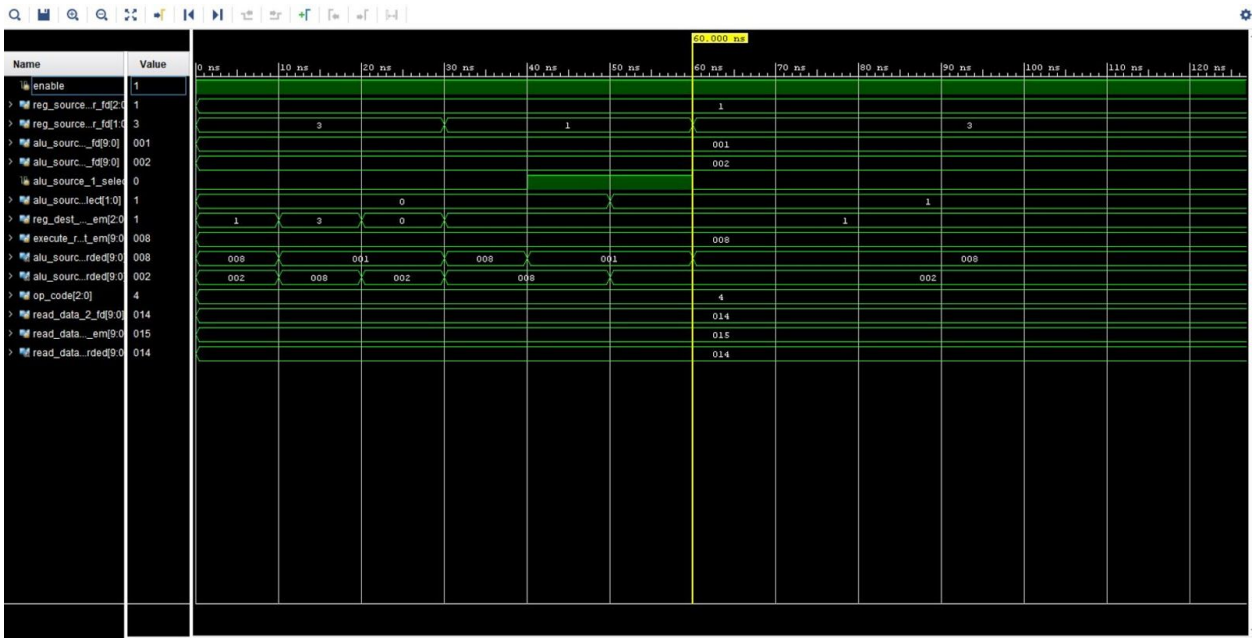
E/M



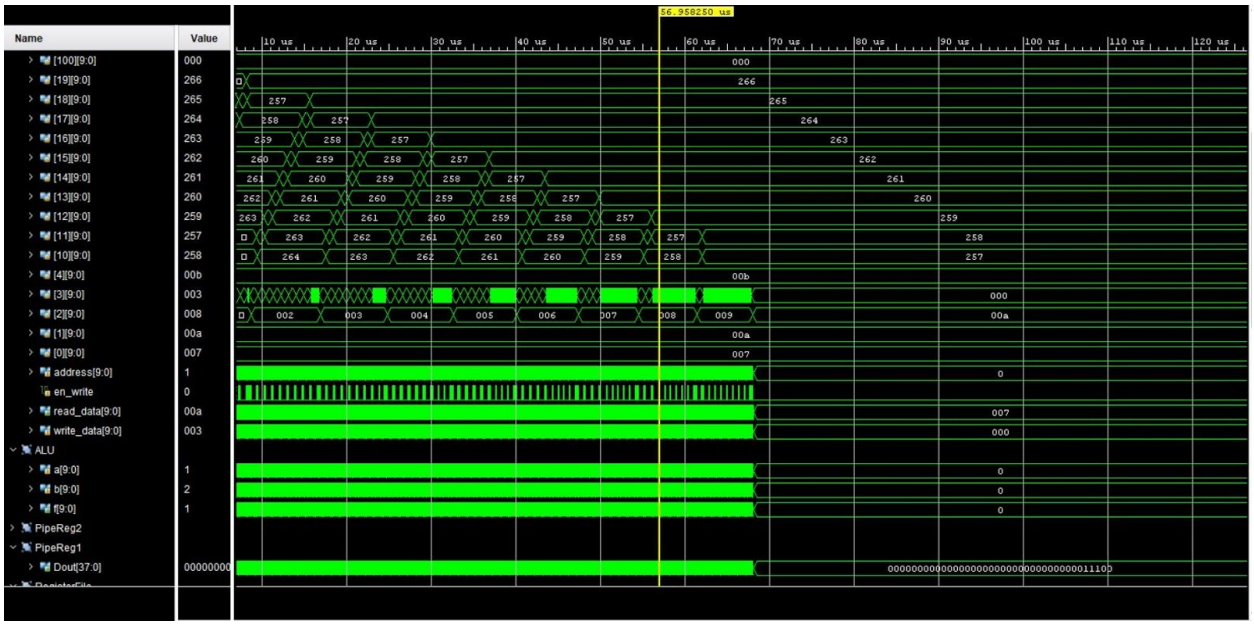
E/M



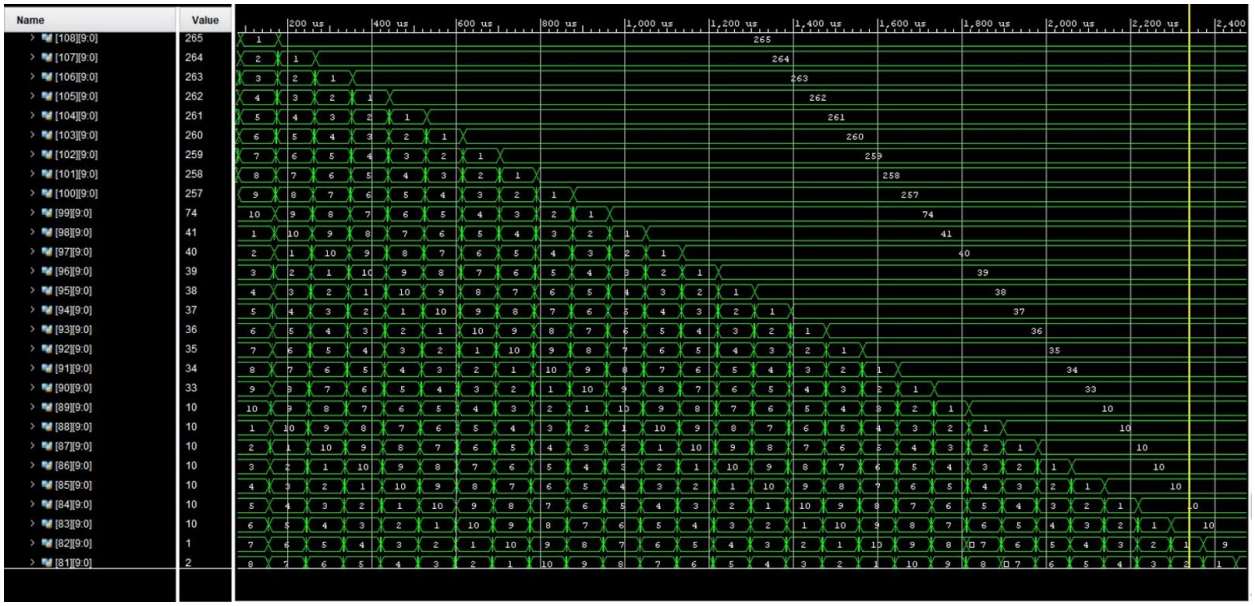
WriteBack



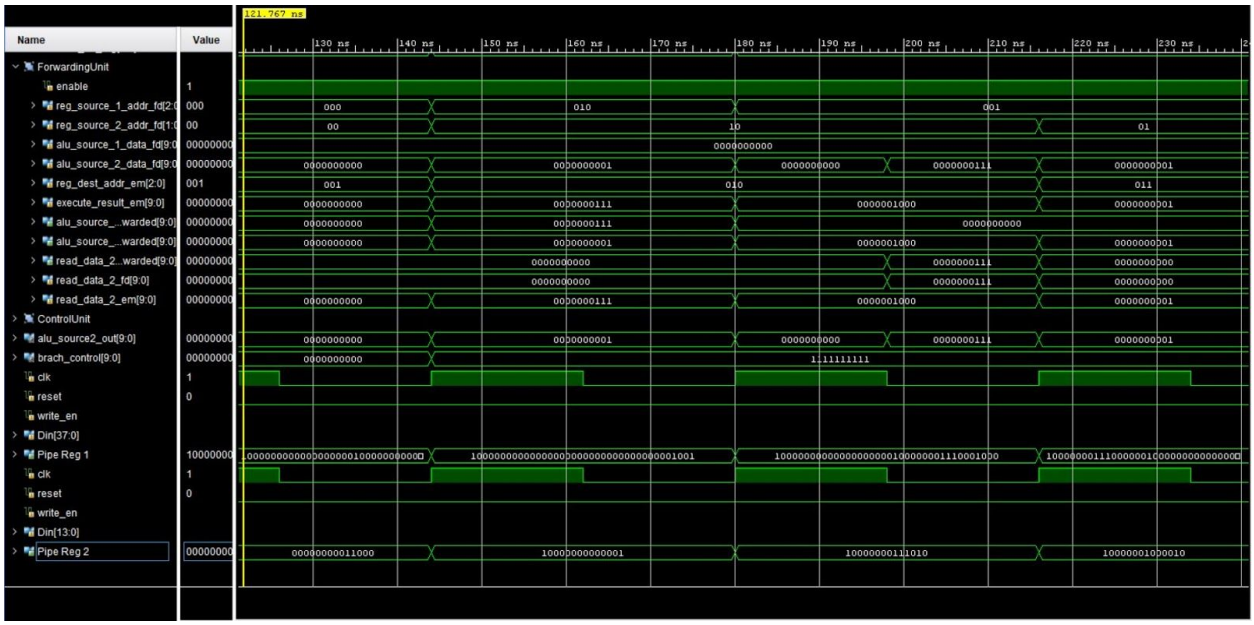
Forwarding Unit Test



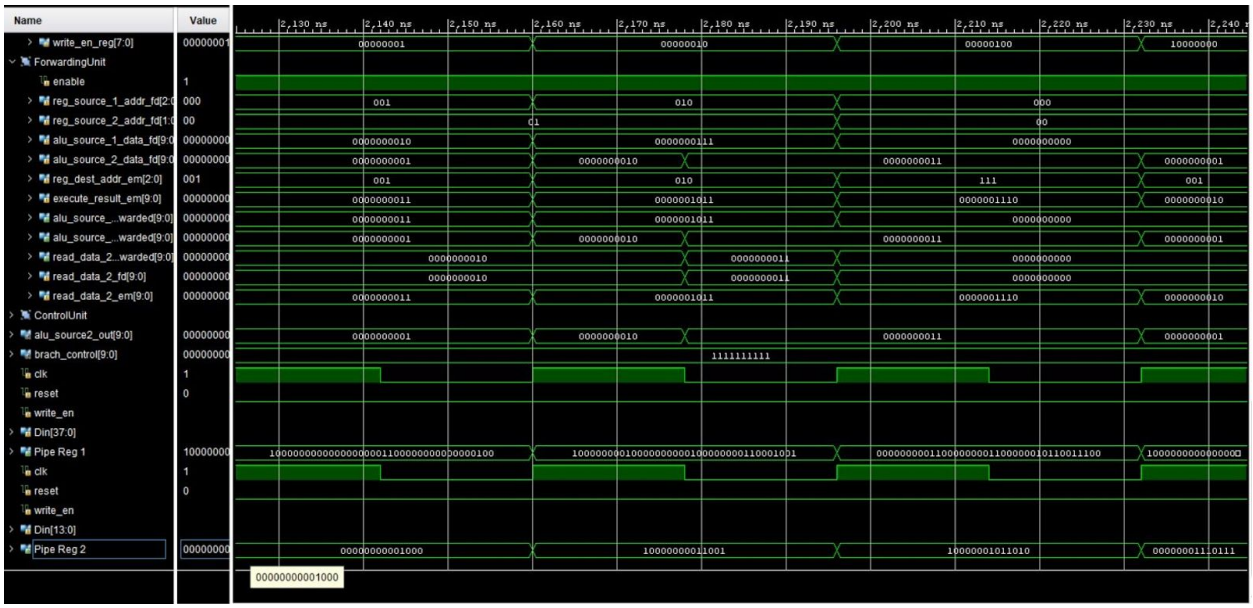
Program 1 10 Numbers



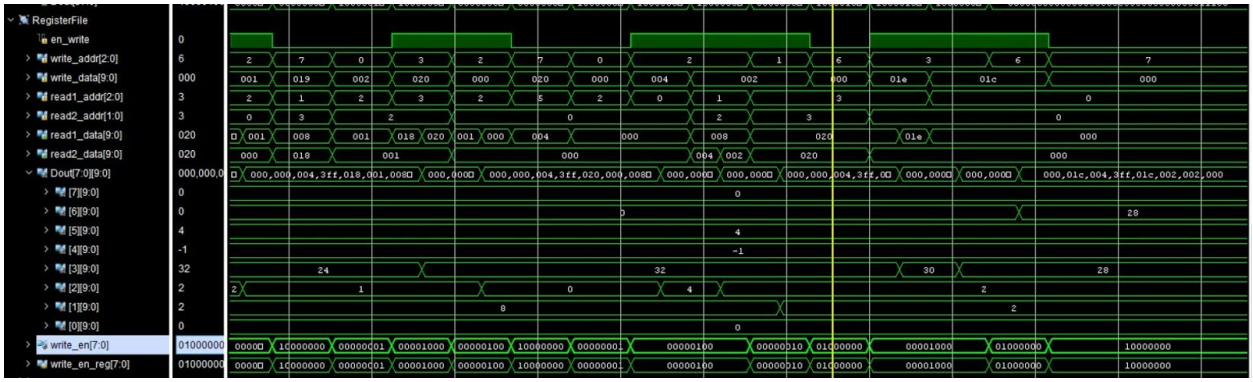
Program 1 100 Numbers



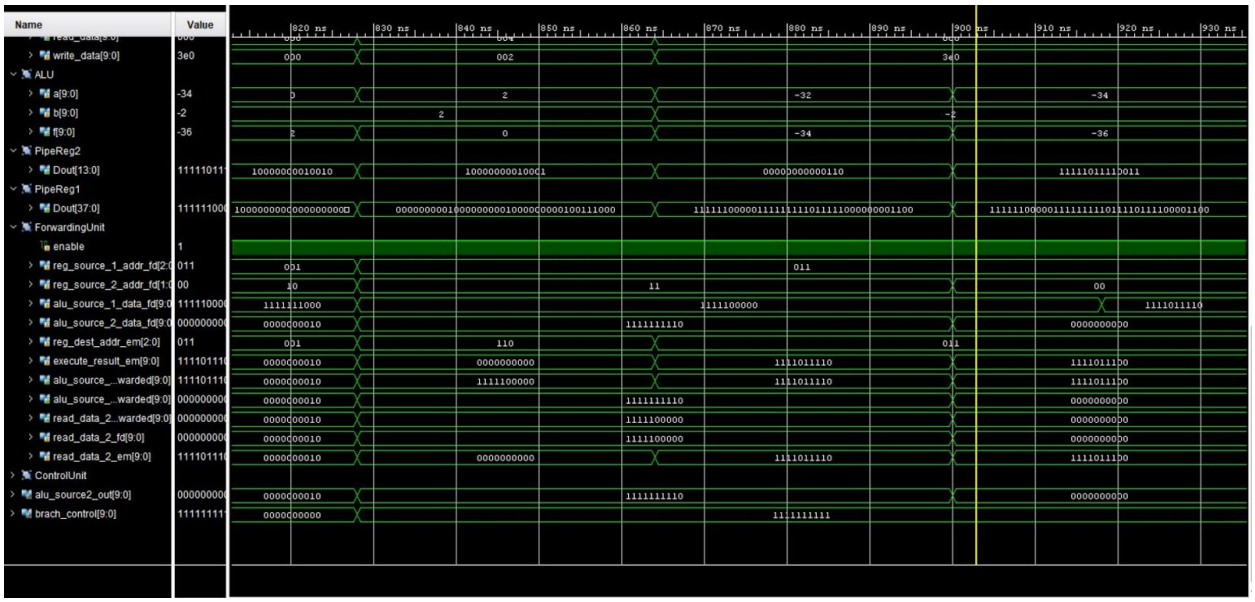
Program 1 Forwarding Unit and Pipe Reg Steps 1-4



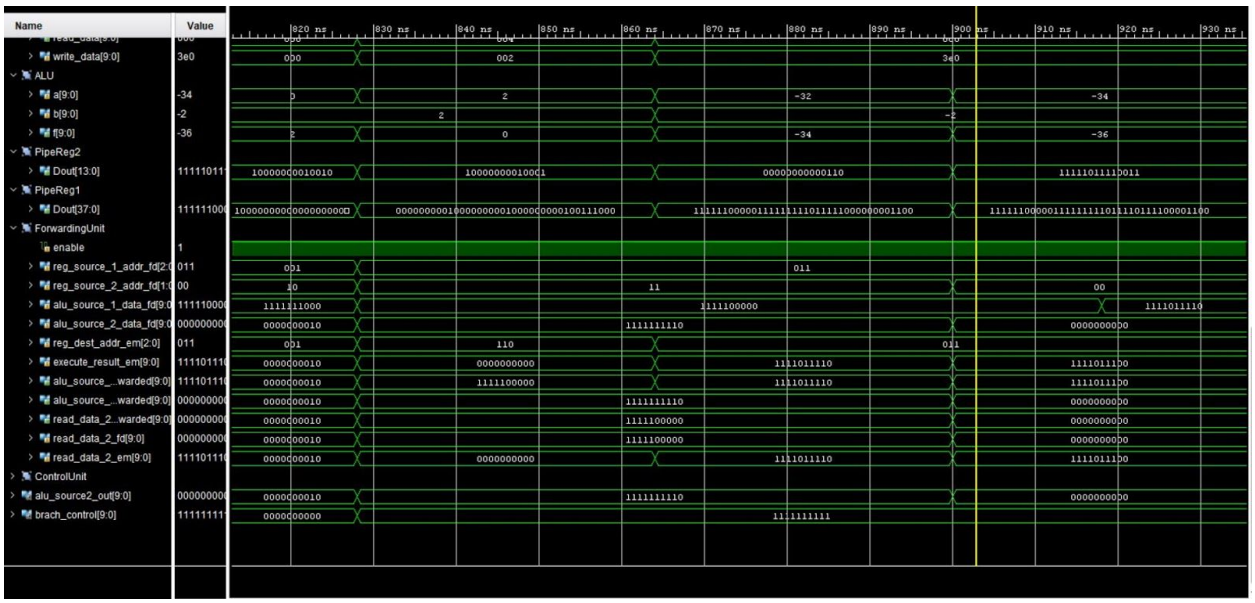
Program 1 Steps 35,36 - > 11, 12 Forwarding Unit and Pipe Reg



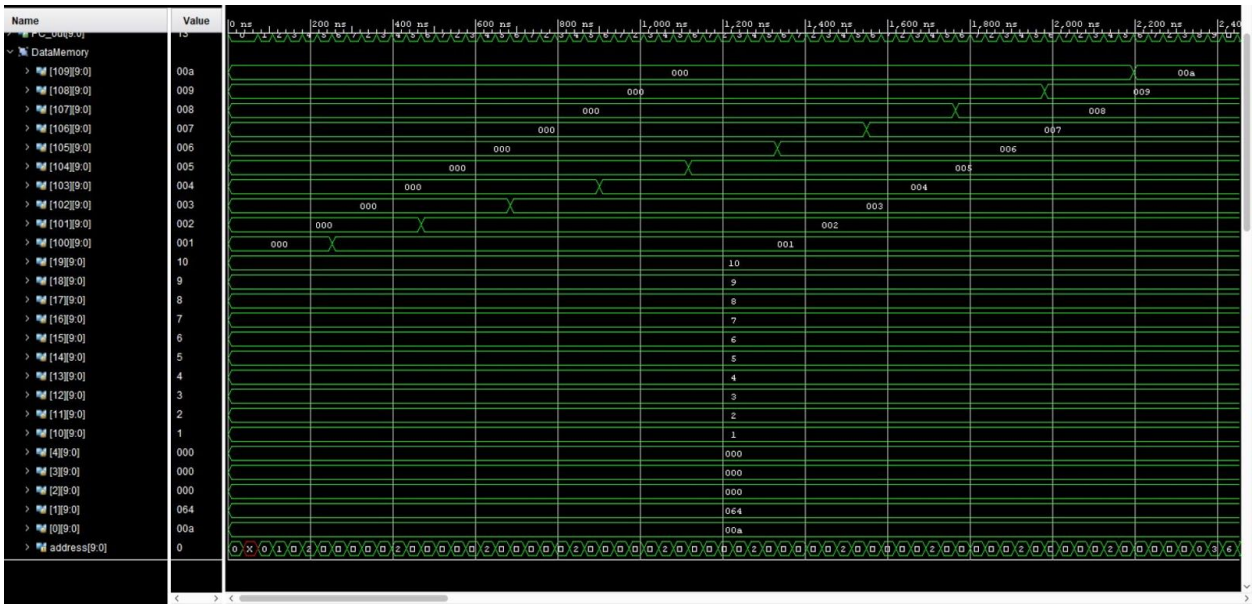
Program 2 8 * 4



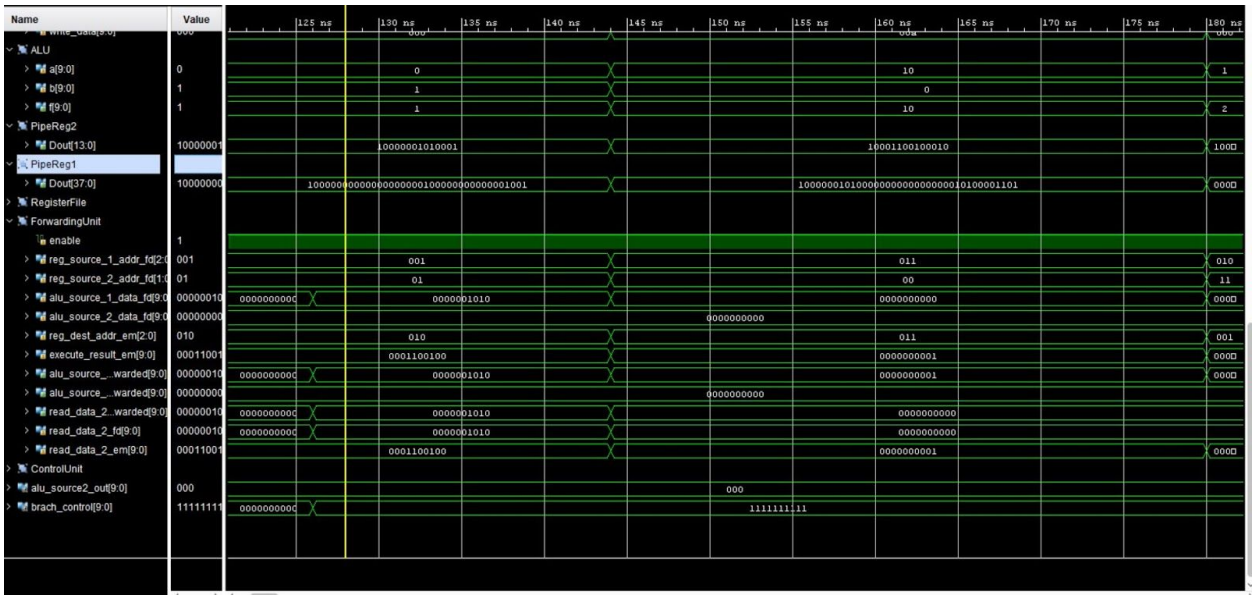
Program 2 4 * -8



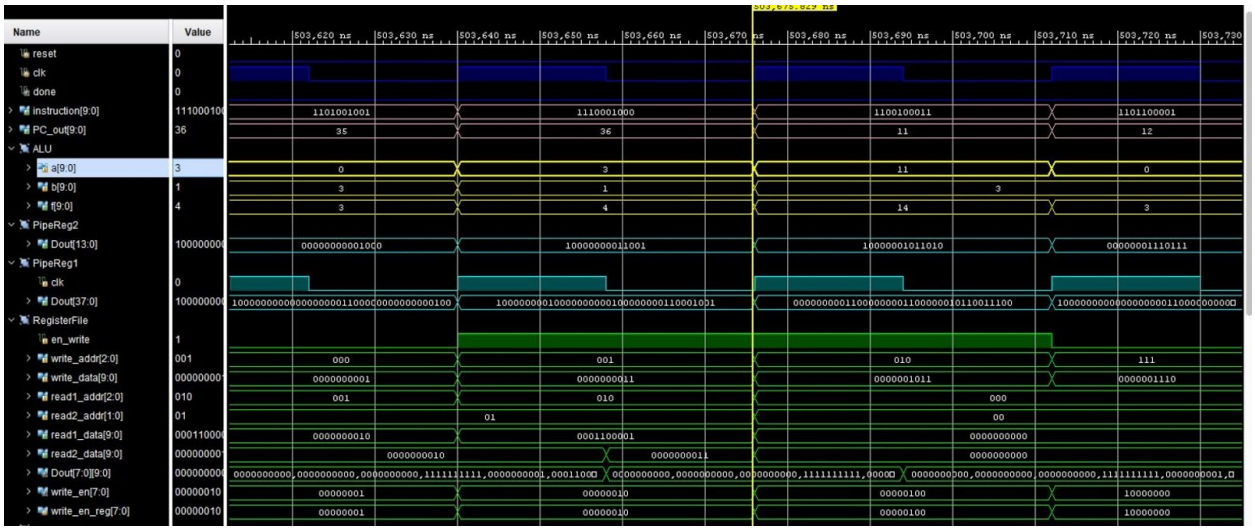
Program 2 Forwarding Unit and Pipe Regs 8 *4



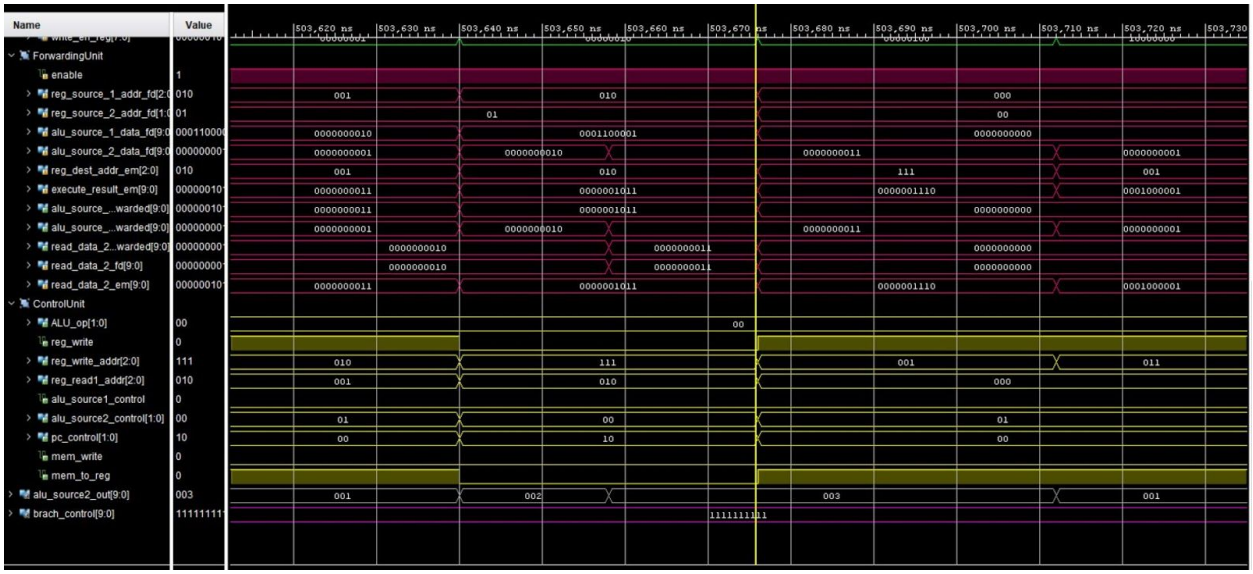
Program 3 Memory



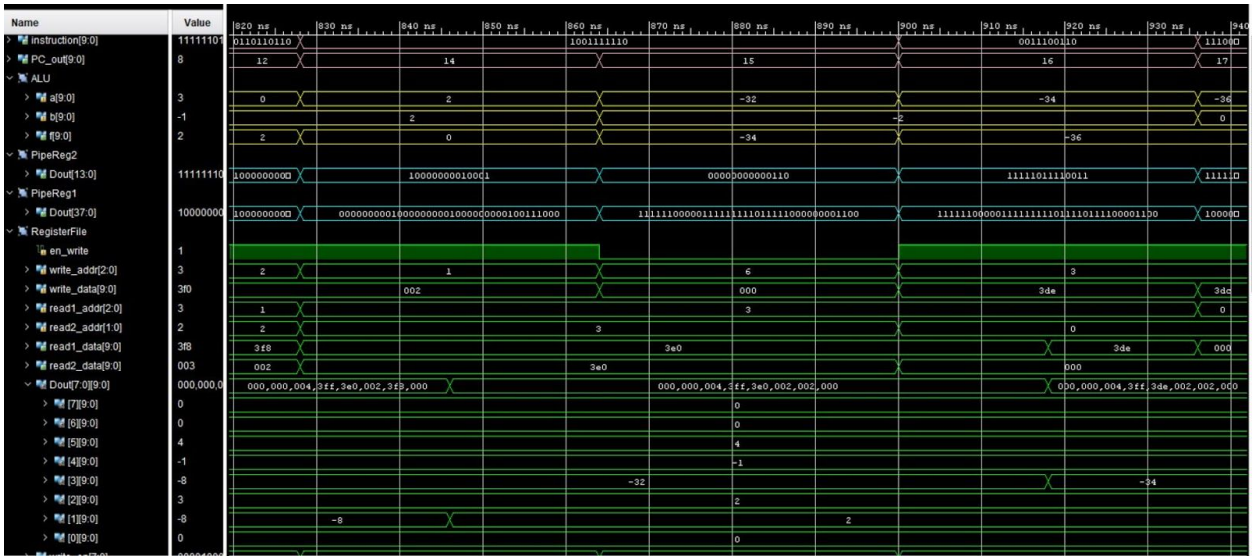
Program 3 Forwarding Unit and Pipe Regs



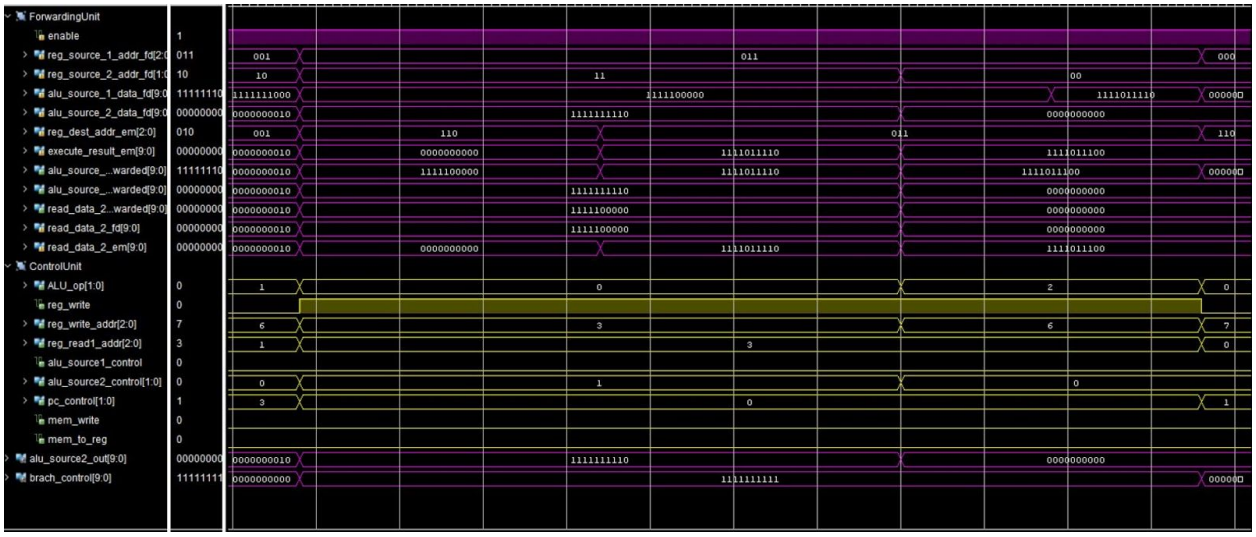
Program 1 (100 Numbers) Pt 1



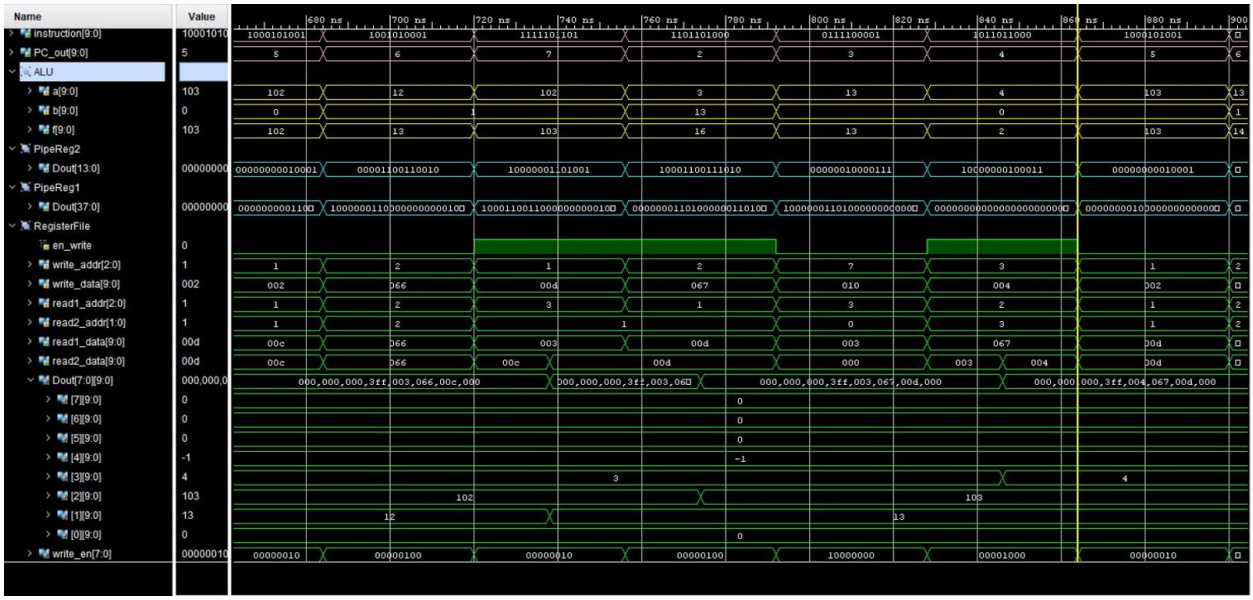
Program 1 (100 Numbers) Pt 2



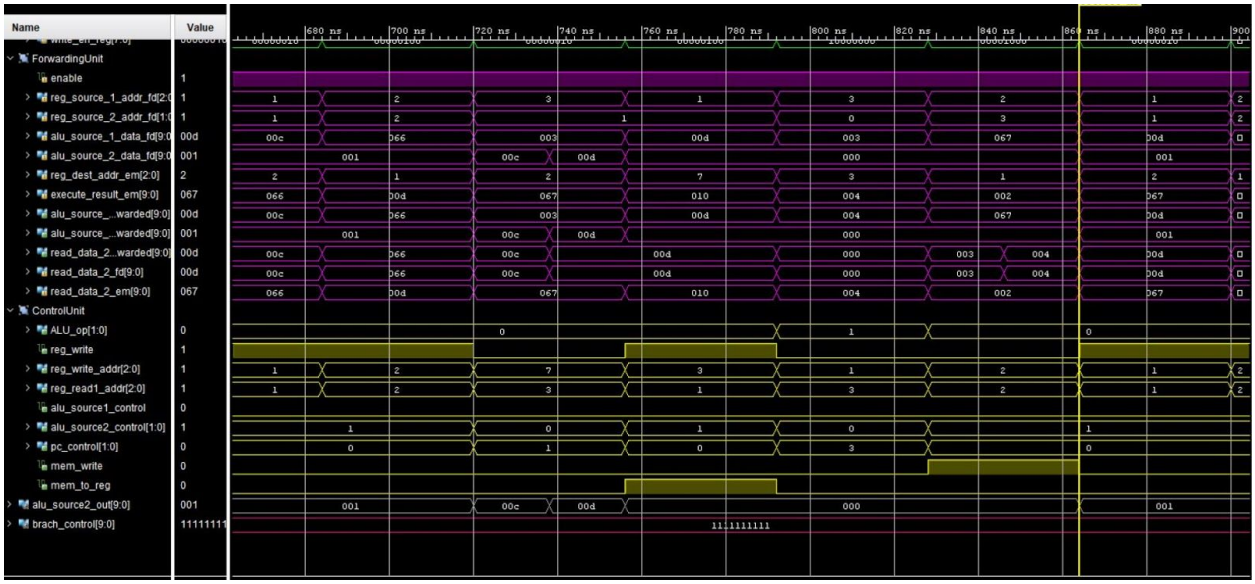
Program 2 (4*-8 -4) Pt 1



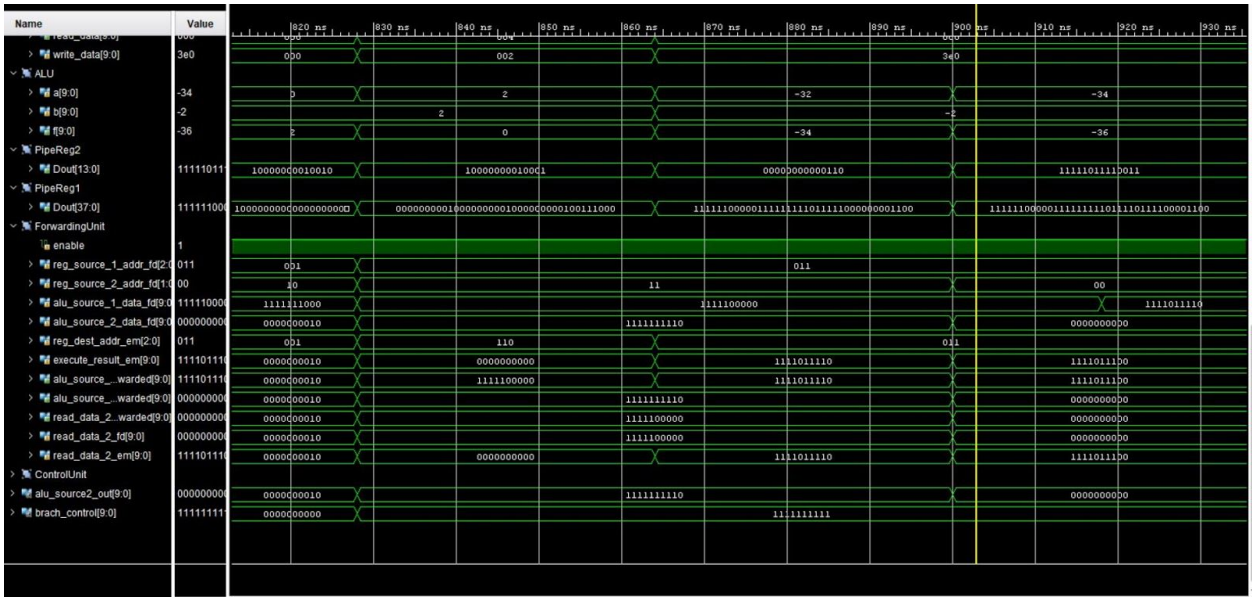
Program 2 (4*-8 -4) Pt 2



Program 3 Pt 1



Program 3 Pt 2


Name
Value
write_data[9:0]
3e0
ALU
a[9:0]
-34
b[9:0]
-2
f[9:0]
-36
PipeReg2
Dout[13:0]
PipeReg1
Dout[37:0]
ForwardingUnit
enable
reg_source_1_addr_fd[2:0]
reg_source_2_addr_fd[1:0]
alu_source_1_data_fd[9:0]
alu_source_2_data_fd[9:0]
reg_dest_addr_em[2:0]
execute_result_em[9:0]
alu_source_...warded[9:0]
alu_source_...warded[9:0]
read_data_2...warded[9:0]
read_data_2_fd[9:0]
read_data_2_em[9:0]
ControlUnit
alu_source2_out[9:0]
brach_control[9:0]
820 ns
830 ns
840 ns
850 ns
860 ns
870 ns
880 ns
890 ns
900 ns
910 ns
920 ns
930 ns