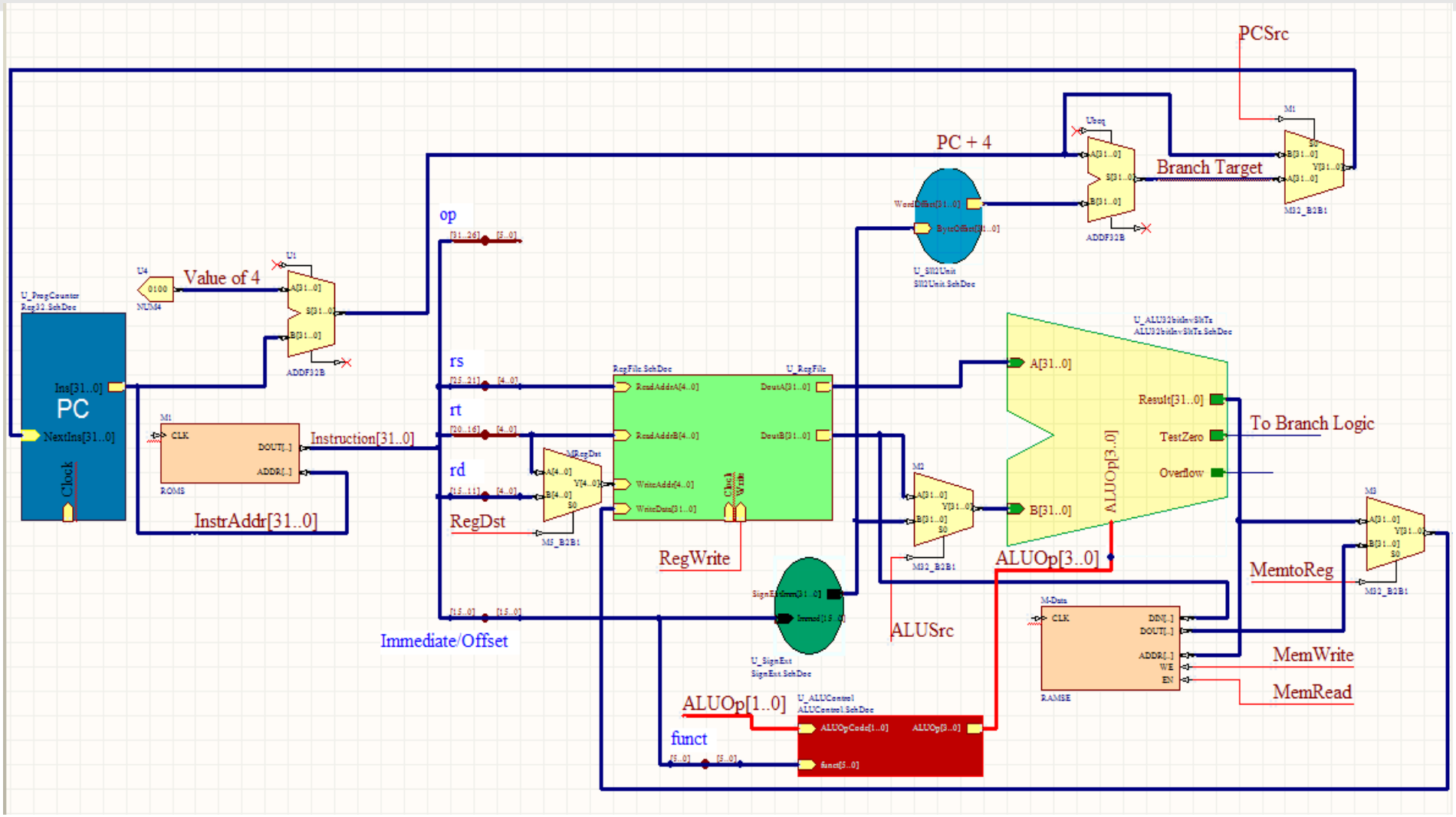
CSE 2021
COMPUTER ORGANIZATION

HUGH CHESSER
CSEB 1012U

Combined Datapath



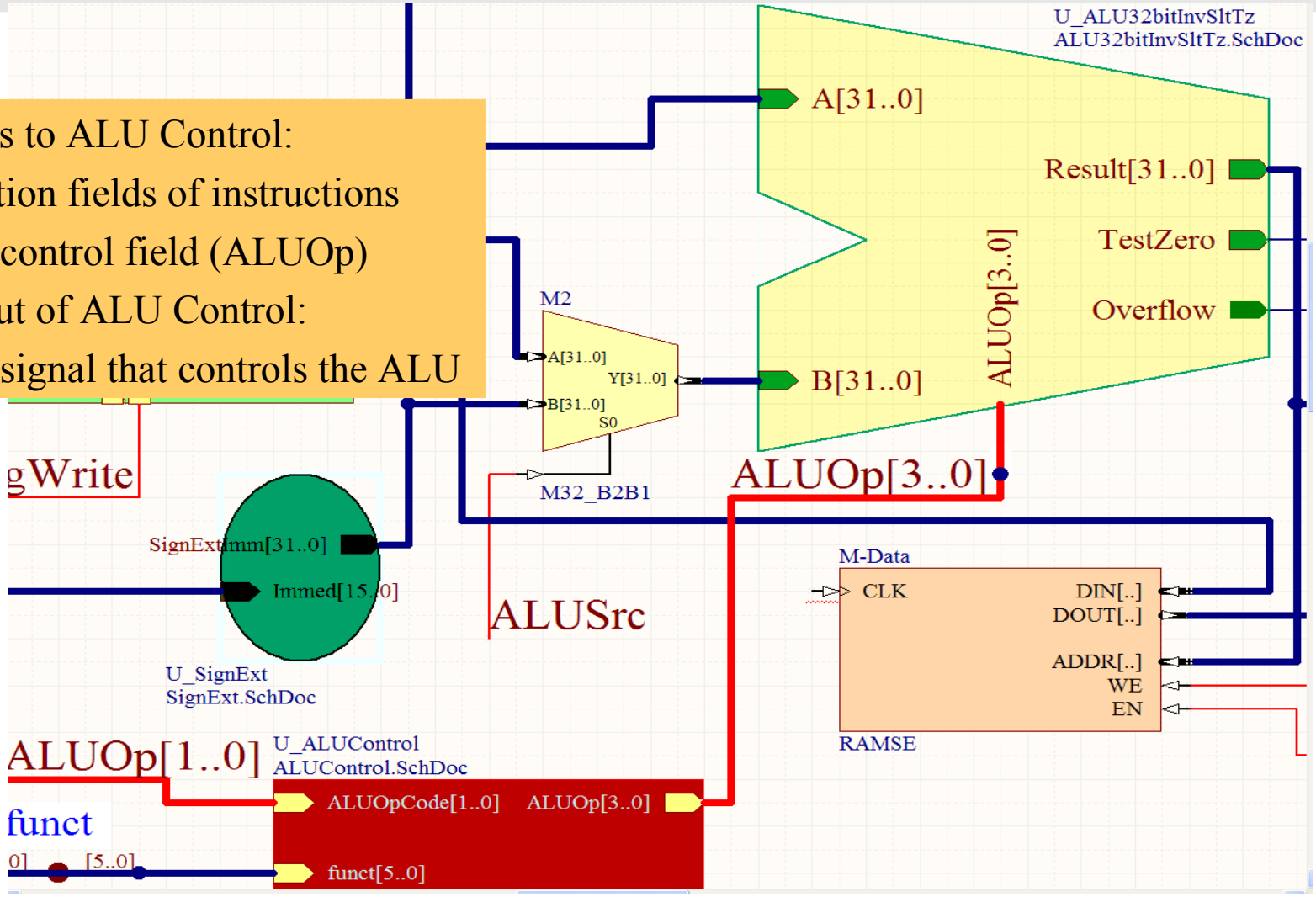
Control: ALU Control Unit (1)

— Inputs to ALU Control:

1. Function fields of instructions
2. 2-bit control field (ALUOp)

— Output of ALU Control:

1. 4-bit signal that controls the ALU



Control: ALU Control Unit (4)

Instruction (funct)	Inputs: ALUOp (ALUOp1 – ALUOp0)	Inputs: Function Field (F5 – F0)	Desired ALU action	Outputs Operation (Op3 – Op0)
lw (I)	0 X (0 0)	X X X X X X	**add**	**0 0 1 0**
sw (I)	0 X (0 0)	X X X X X X	**add**	**0 0 1 0**
beq (I)	X 1 (0 1)	X X X X X X	**sub**	**0 1 1 0**
add (32)	1 X (1 0)	X X 0 0 0 0	**add**	**0 0 1 0**
sub (34)	1 X (1 0)	X X 0 0 1 0	**sub**	**0 1 1 0**
and (36)	1 X (1 0)	X X 0 1 0 0	**and**	**0 0 0 0**
or (37)	1 X (1 0)	X X 0 1 0 1	**or**	**0 0 0 1**
slt (42)	1 X (1 0)	X X 1 0 1 0	**slt**	**0 1 1 1**

Simplified Expressions:

$$Op0 = \overline{ALUOp1} \cdot (F0 + F3)$$
$$Op1 = \overline{ALUOp1} + \overline{F2}$$
$$Op2 = ALUOp0 + ALUOp1 \cdot F1$$

Agenda

Topics:

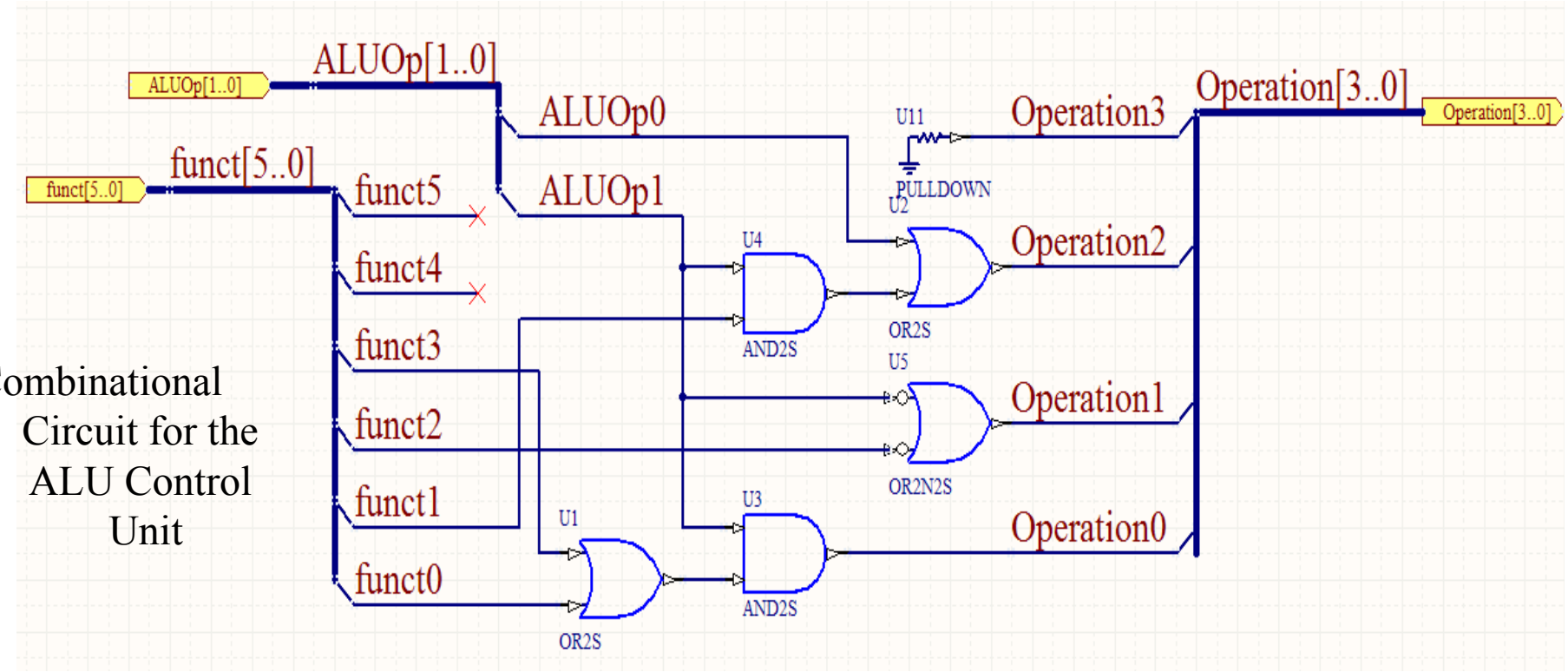
1. A single cycle implementation (complete this)
2. iverilog Example
3. Lab D – double precision version

Patterson: Section 4.3, 4.4

Reminder: No class next Wednesday, Quiz 2 is following Wednesday, Lab K (Verilog) next week

Control: ALU Control Unit (5)

$$Op0 = ALUOp1 \cdot (F0 + F3)$$
$$Op1 = \overline{ALUOp1} + \overline{F2} = \overline{ALUOp1 \cdot F2}$$
$$Op2 = ALUOp0 + ALUOp1 \cdot F1$$



Combinational Circuit for the ALU Control Unit

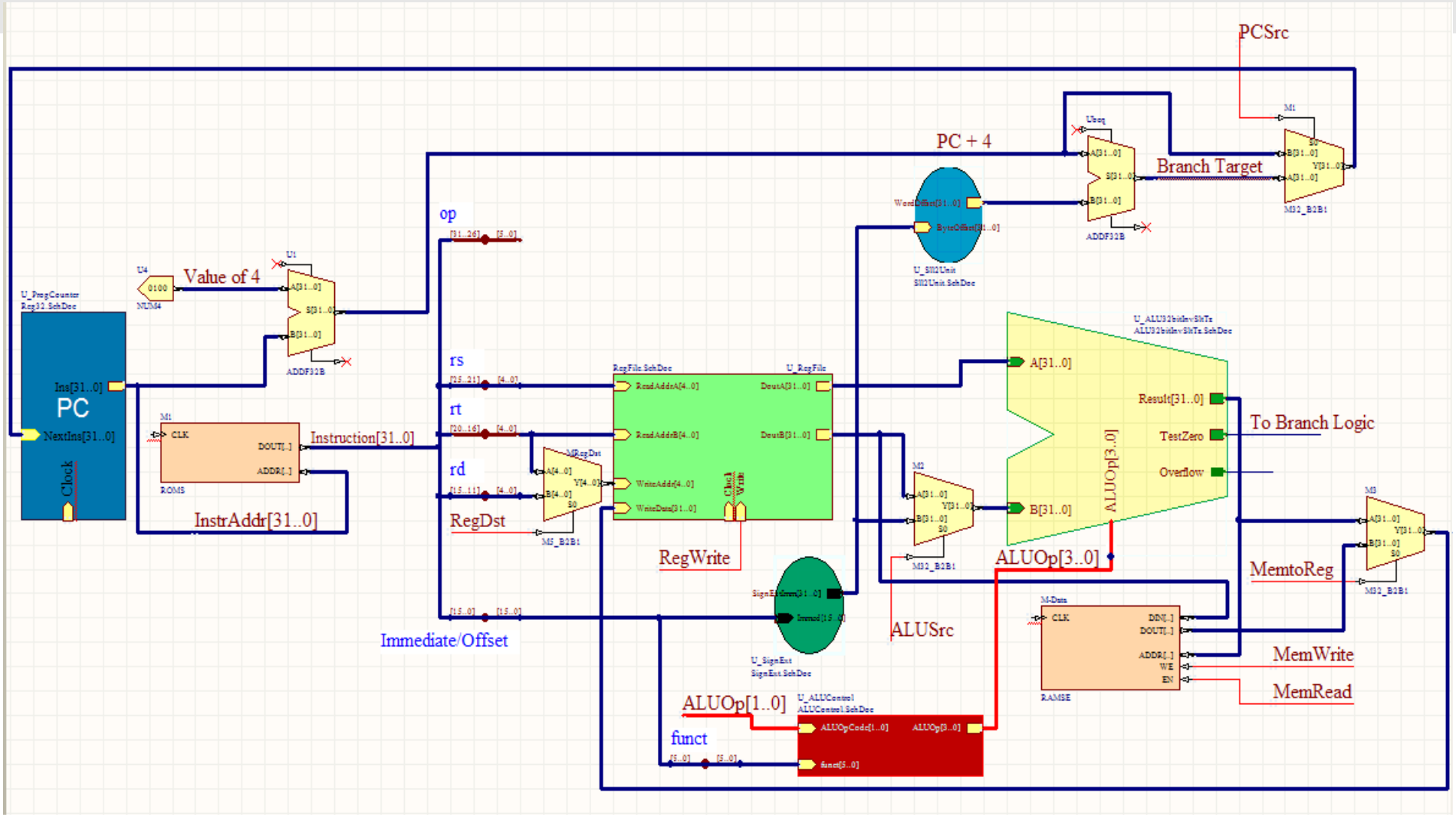
Main Control (1)

R-format:	**op (31 – 26)**	**rs (25 – 21)**	**rt (20 – 16)**	**rd (15 – 11)**	**shamt (10 – 6)**	**funct (5 – 0)**
I-format:	**op (31 – 26)**	**rs (25 – 21)**	**rt (20 – 16)**	**Immediate / Offset (15 – 0)**		

1. Opcode is contained in bits 31 – 26.
2. Registers specified by rs (bits 25 – 21) and rt (bits 20 – 16) are always read
3. Base register (w/ base address) for lw/sw instruction is specified by rs (bits 25–21)
4. 16-bit offset for **`beq`**, **`lw`**, and **`sw`** is always specified in bits 15 – 0.
5. Destination register is specified in one of the two places:
   - For R-type instructions (add/sub/and/or), destination register is specified by bits (15 – 11)
   - For lw instruction, destination register is specified by bits (20 – 16)

Using information (1 – 5), we can add the instruction labels and additional MUX's to the datapath that we have constructed.

Main Control (2)



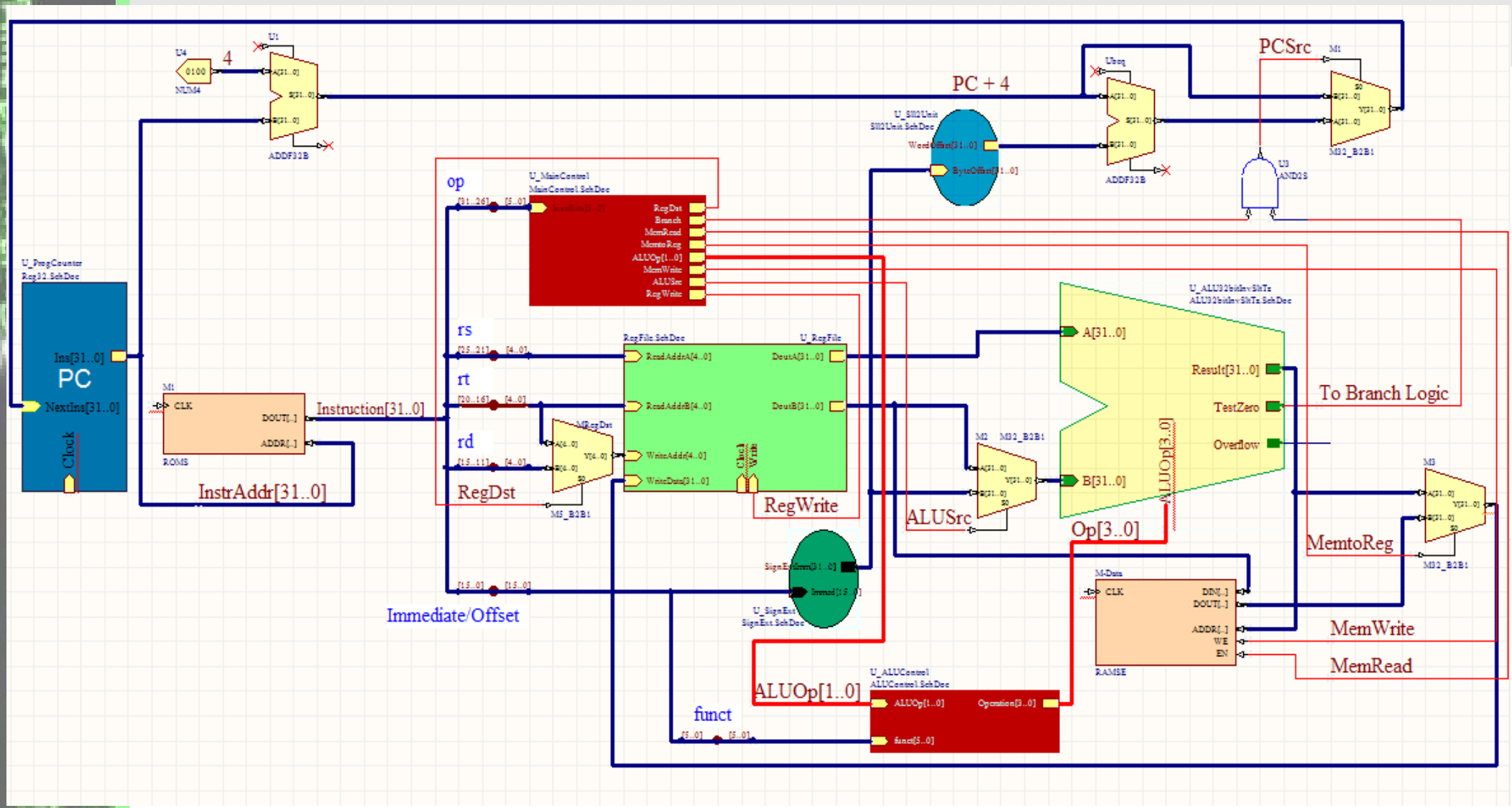
Number of control lines is 11 (4 for MUX's, 4 for ALU control, 2 for data memory, 1 for register file)

Main Control (3)

Control Input	Effect when Deasserted (0)	Effect when asserted (1)
RegDst	Destination register number comes from bits 20 – 16 of the instruction `(sw)`	Destination register number comes from bits 15 – 11 of the instruction `(add,sub,or,and,slt)`
Regwrite	None	Data on the "write data" input is written on the register specified on the "write register" input `(lw, add,sub,or,and)`
ALUSrc	Second operand to ALU comes from the second register file output `(add,sub,or,and,beq,slt)`	Second operand to ALU is sign extended, lower 16 bits of instruction(lw,sw)
MemRead	None	Data from memory location specified by "address" input is placed on the "read data" output `(lw)`
MemWrite	None	Data from "write data" input replaces memory location specified by "address" input (`sw`)
MemtoReg	Data from the output of ALU is fed into "write data" input of the register file `(add,sub,or,and,slt)`	Data from the "read data" output of data memory is fed into "write data" input of the register file `(lw)`
PCSrc	PC is replaced by the output of adder which adds 4 to the existing content of PC (except for `beq`)	PC is replaced by the output of adder which computes branch target by adding existing content of PC with 2-bit right shifted offset (`beq`)

Next step in the design of datapath is to add a control unit that generates the control inputs to MUX's

Main Control (4)



Main Control (5)

Control Unit Inputs:

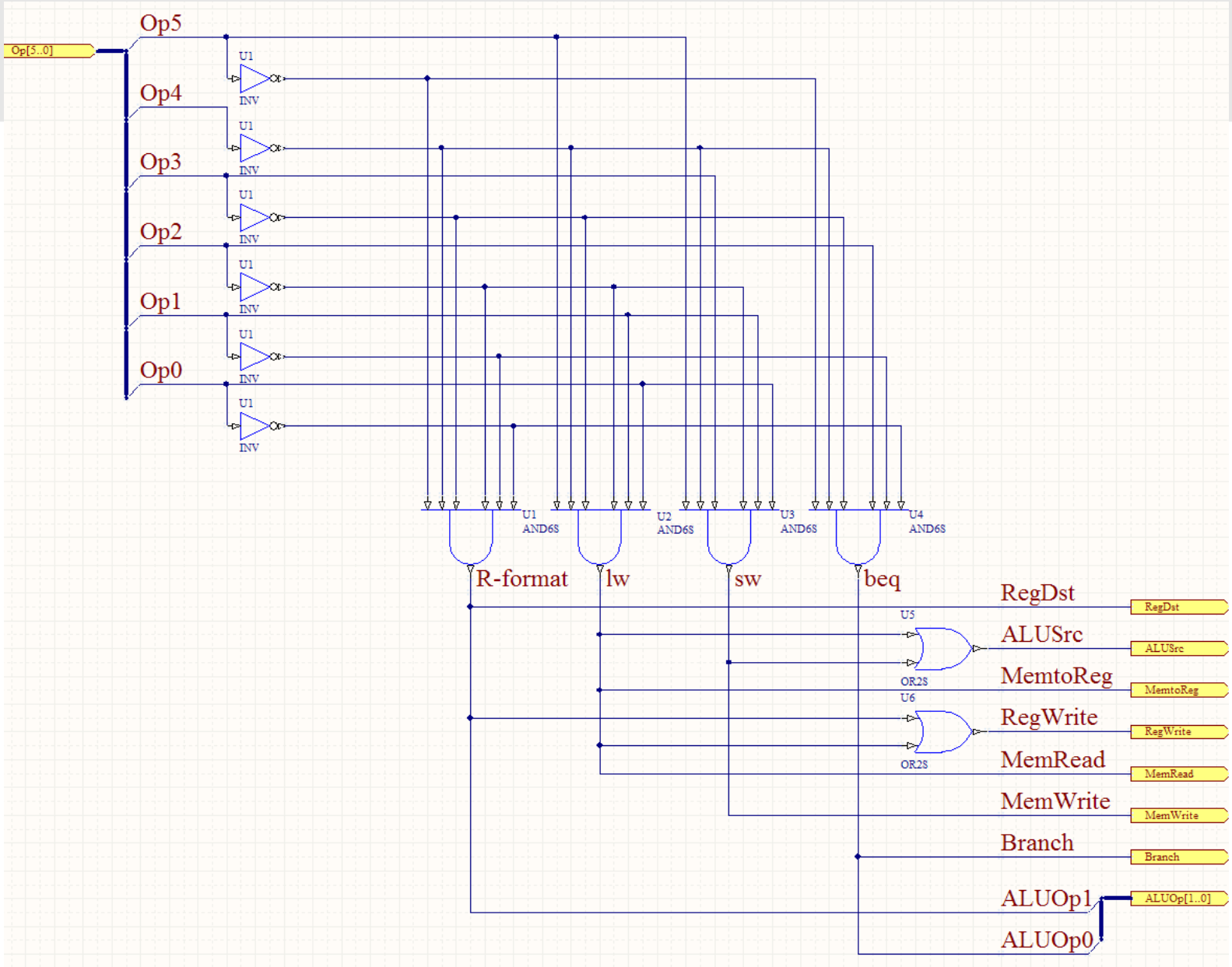
Instruction	Opcode in Decimal	Opcode in Binary					
		Op5	Op4	Op3	Op2	Op1	Op0
R-format	0_{ten}	**0**	**0**	**0**	**0**	**0**	**0**
`lw`	35_{ten}	**1**	**0**	**0**	**0**	**1**	**1**
`sw`	43_{ten}	**1**	**0**	**1**	**0**	**1**	**1**
`beq`	4_{ten}	**0**	**0**	**0**	**1**	**0**	**0**

Outputs of Control Unit:

Instruction	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
R-format	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**0**
`lw`	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**
`sw`	**X**	**1**	**X**	**0**	**0**	**1**	**0**	**0**	**0**
`beq`	**X**	**0**	**X**	**0**	**0**	**0**	**1**	**0**	**1**

...above 2 tables constitute the truth table

Main Control (6)

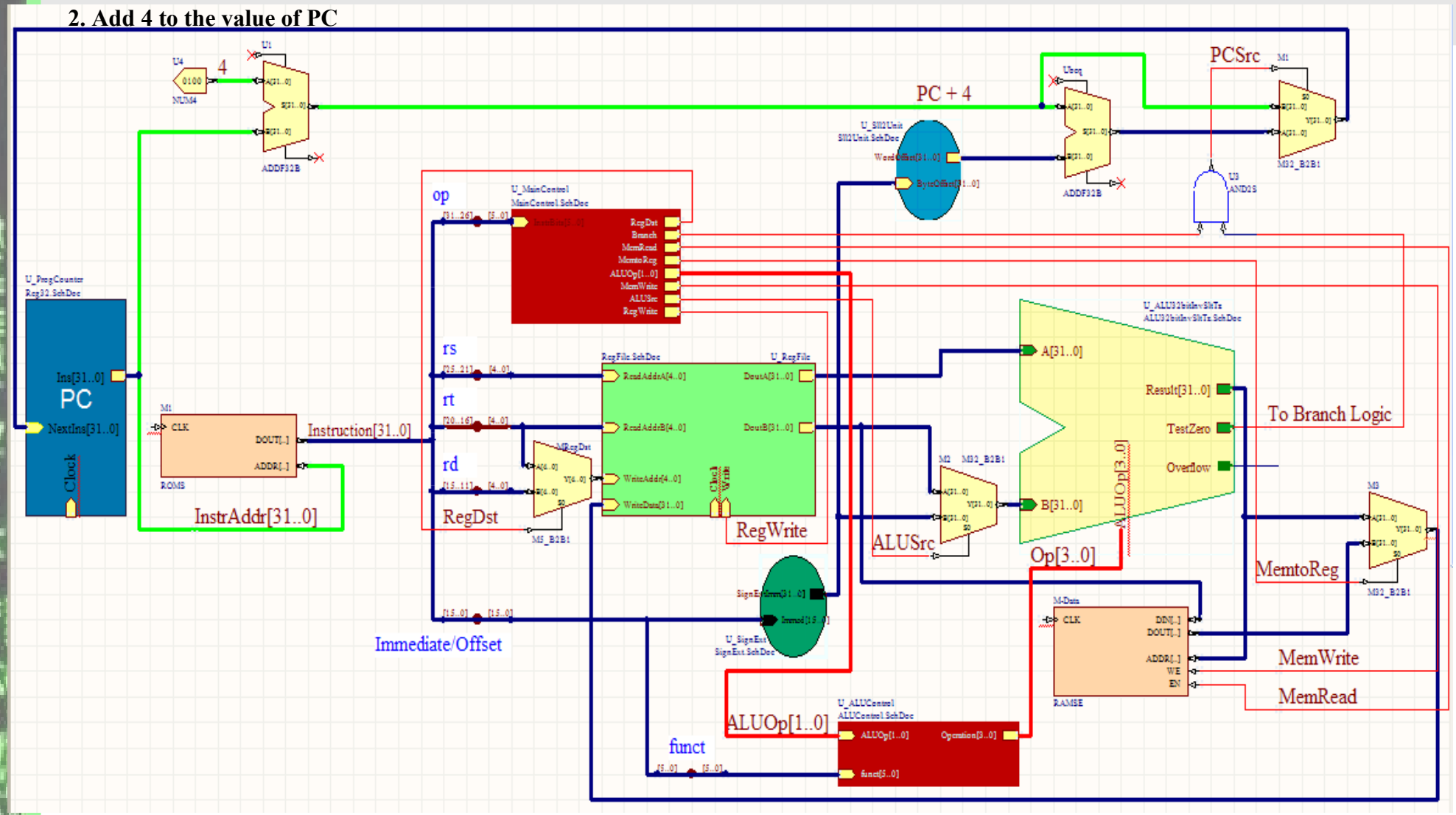


Example: R-type Instruction (step1: fetch instruction & increment PC)

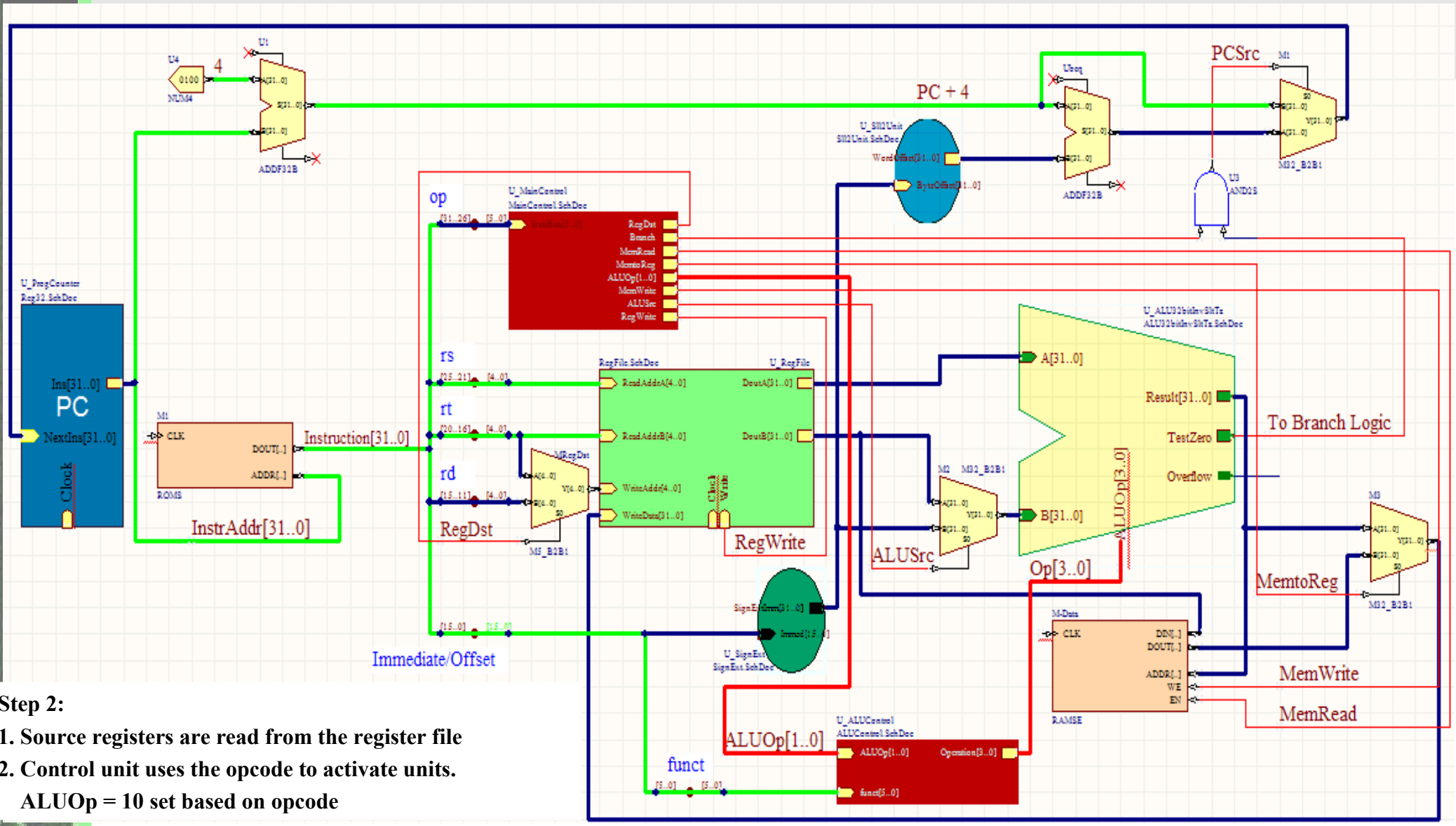
Step 1

1. Fetch instruction

2. Add 4 to the value of PC



Example: R-type Instruction (step2: Read two source registers)



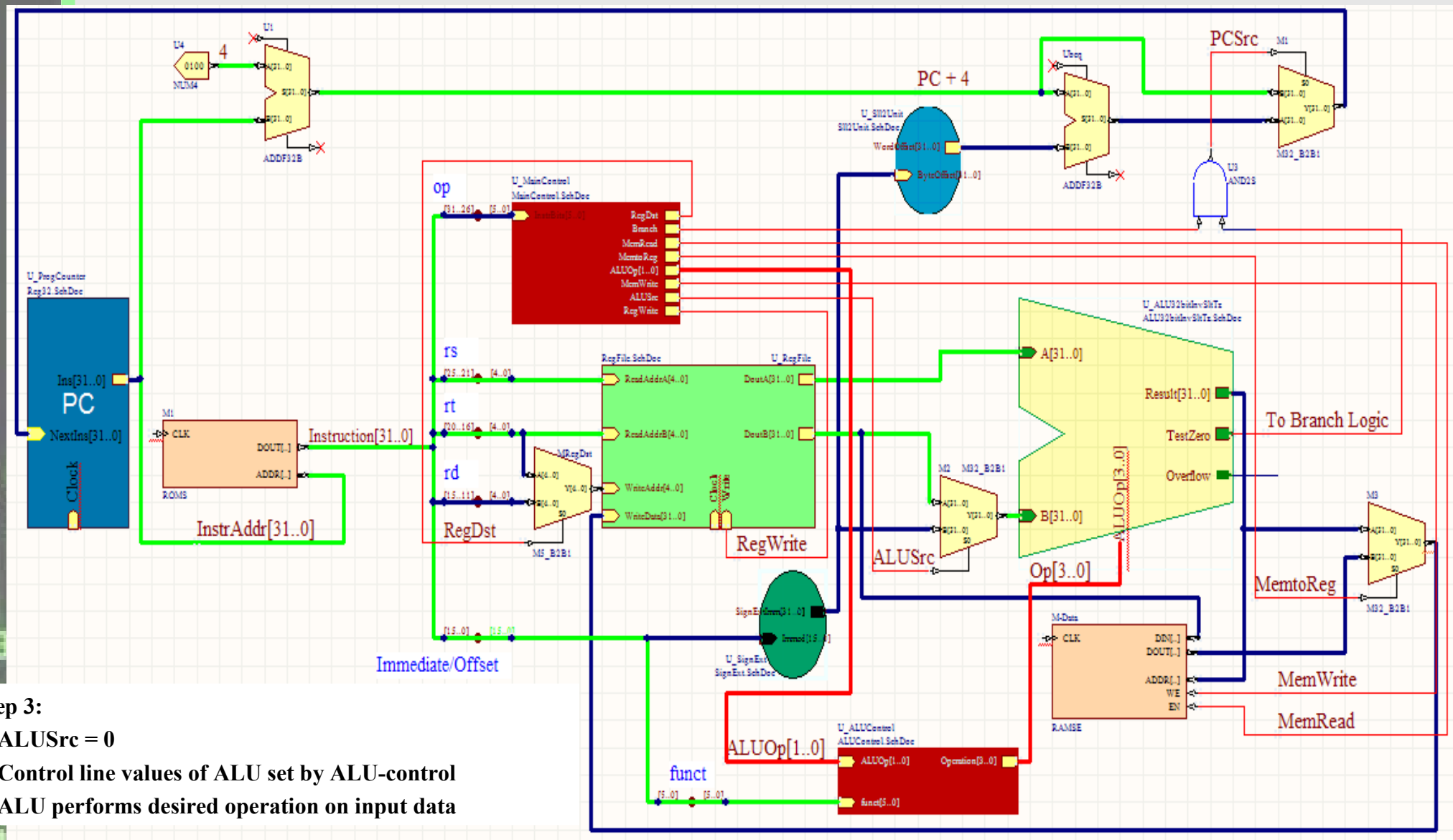
Step 2:

1. Source registers are read from the register file

2. Control unit uses the opcode to activate units.

ALUOp = 10 set based on opcode

Example: R-type Instruction (step3: ALU operates on operands)



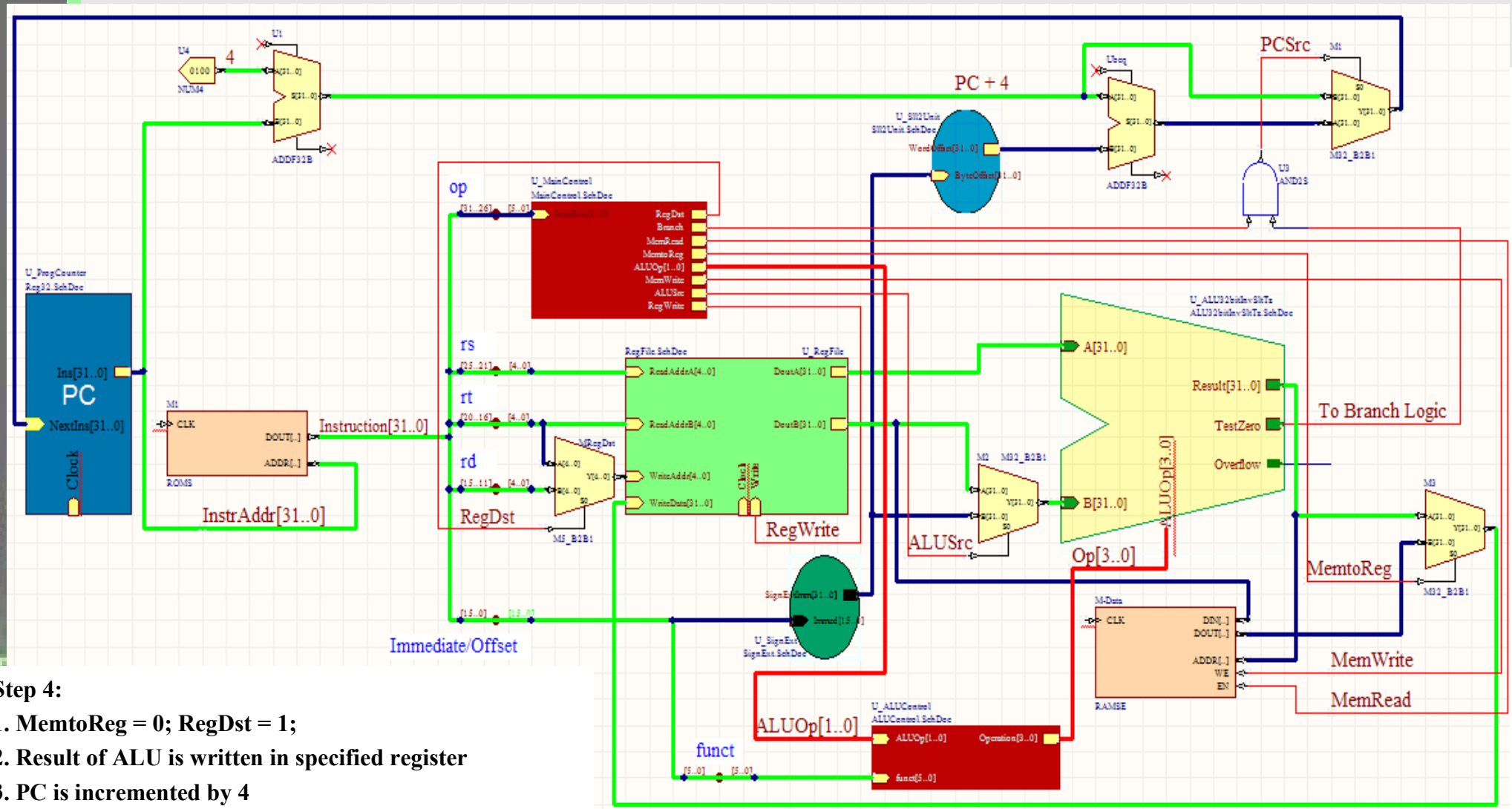
Step 3:

1. ALUSrc = 0

2. Control line values of ALU set by ALU-control

2. ALU performs desired operation on input data

Example: R-type Instruction (step 4: Write result in destination register)



Step 4:

1. MemtoReg = 0; RegDst = 1;

2. Result of ALU is written in specified register

3. PC is incremented by 4

Why is single-cycle implementation not used?

Assuming no delay at adder, sign extension unit, shift left unit, PC, control unit, and MUX:

- Load cycle requires 5 functional units:
  instruction fetch, register access, ALU, data memory access, register access
- Store cycle requires 4 functional units:
  instruction fetch, register access, ALU, data memory access
- R-type instruction cycle requires 4 functional units:
  instruction fetch, register access, ALU, register access
- Path for a branch instruction requires 3 functional units:
  instruction fetch, register access, ALU
- Path for a jump instruction requires 1 functional unit:
  instruction fetch

Using a clock cycle of equal duration for each instruction is a waste of resources.

iverilog Example

C:\iverilog\Q1.v - Notepad++

File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Wir

Q1.v testbench.v

```
1  module eq2_sop
2      (
3      input wire[1:0] a, b,
4      output wire aeqb
5      );
6
7      // internal signal declaration
8      wire p0, p1, p2, p3;
9
10     //sum of product terms
11     assign aeqb = p0 | p1 | p2 | p3;
12     // product terms
13     assign p0 = (~a[1] & ~b[1]) & (~a[0] & ~b[0]);
14     assign p1 = (~a[1] & ~b[1]) & (a[0] & b[0]);
15     assign p2 = (a[1] & b[1]) & (~a[0] & ~b[0]);
16     assign p3 = (a[1] & b[1]) & (a[0] & b[0]);
17
18 endmodule
```

C:\iverilog\testbench.v - Notepad++

File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ?

Q1.v testbench.v

```
1  module testbench;
2      reg [1:0] test_a, test_b;
3      wire test_c;
4
5      eq2_sop uut_eq2(test_a, test_b, test_c);
6
7      initial
8      begin
9          #200
10         test_a = 2'b10;
11         test_b = 2'b10;
12         #1 $display($time, " a = %d, b = %d, c = %d",test_a, test_b, test_c);
13         #200
14         test_a = 2'b00;
15         test_b = 2'b10;
16         #1 $display($time, " a = %d, b = %d, c = %d",test_a, test_b, test_c);
17     end
18 endmodule
```

C:\WINDOWS\system32\cmd.exe

```
C:\iverilog>iverilog Q1.v testbench.v

C:\iverilog>vvp a.out
                201 a = 2, b = 2, c = 1
                402 a = 0, b = 2, c = 0

C:\iverilog>
```

Lab D – Double Precision Read/Write

```
# CSE2021 Lab D
# Name: removed
# csexxxxx  s/n yyy yyy yyy
# October nn, 2012
#
# TABS ARE SET TO 4

            .data
HEAD:       .word   0                       # pointer to first node
message:    .asciiz  "Enter a double precision number (-1 to stop): \n"
            .text
            ###############################################################
            # main procedure
            #
            # 12-10-23: modified to read double precision values - H.Chesser
            # 12-10-23: modified to prompt user for values
            ###############################################################
main:       sw      $ra,    0($sp)
            addi    $sp,    $sp,    -4      #opening main
            # Read & process loop
            # Read int values from user one at a time and process them
            # Ends on input -1.0e0
prompt:     li      $v0,    4
            la      $a0,    message
            syscall
read:       li      $v0,    7               #Syscall 7 for double read
            syscall
            li.d        $f10,   -1.0e0      #sentinel
            c.eq.d      $f0,    $f10        # double comparison
            bc1t        endRead             #break on sentinel
            lw      $a0     HEAD($zero)     #a0 points to first node
#           add     $a1,    $v0,    $zero   #f0 has read int
            jal     insdCell
            sw      $v0,    HEAD($zero)     #HEAD points to last node
            j       prompt                  #continue
endRead:    ### End of reading and processing ints
            lw      $a0,    HEAD($zero)
            jal     printdList
            addi    $sp,    4               #wrapping up main
            lw      $ra,    0($sp)
            jr      $ra
            # End of main
```

Lab D – Double Precision Read/Write (cont'd)

```
        ############################################################
        # insdCell
        # Creates a new node and stores the passed parameter in it.
        # Parameters:
        #   $a0 = pointer to next node
        #   $f0 = double to be stored
        # Returns:
        #   $v0 = address of created node
        #
        #12-10-23 - Modified to insert double precision values - H.Chesser
        ############################################################
insdCell: add     $t0, $zero, $a0
        add     $t1, $zero, $a1
        li      $a0,    12              #allocate 12 bytes - link address + double
        li      $v0,    9
        syscall
        sw      $t0,    0($v0)          #save pointer
        sdc1    $f0,    4($v0)          #save double data
        jr $ra
```

```
        ############################################################
        # printdList
        # Prints the content of the linked list.
        # Parameters:
        #   $a0 = pointer first node
        #
        # 12-10-23 - Modified to print double values - H.Chesser
        ############################################################
printdList: add     $t0,    $zero   $a0
        beq     $t0,    $zero   endPrint
        ldc1    $f12,   4($t0)          #get data
        li      $v0,    3               #Syscall 3 for double print
        syscall                         #print
        li      $a0,    10              #printnl
        li      $v0,    11
        syscall
        lw      $a0,    0($t0)
        j       printdList              #continue
endPrint:   jr $ra
```