COMPUTER ORGANIZATION EECS 2021

LAB "M" REPORT
BUILDING THE CPU

The work in this report is my own. I have read and understood York University academic dishonesty policy and I did not violate the senate dishonesty policy in writing this report.

X ____________________

Camillo John (CJ) D'Alimonte
212754396
Lab Section 2
11/18/14
Professor Aboelaze

Abstract:

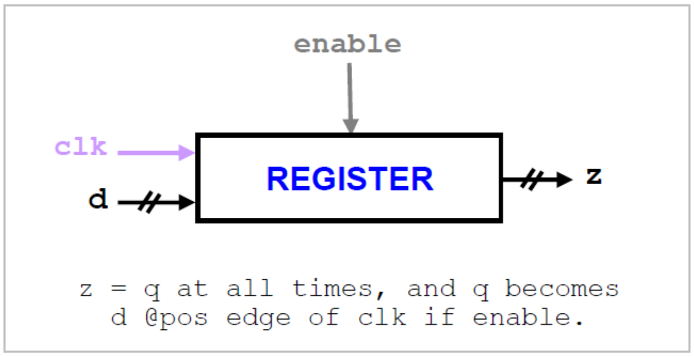
This laboratory/experiment introduced me to some of the more complex applications that can be built through the Verilog language, namely a model of a CPU. I learned how to program different modules, such as instruction encoders and decoders as well as modules that execute various commands such as write backs. The data path that was designed for my CPU is able to handle the following commands: and, or, add, sub, slt, addi, lw, sw, beq, and j. I was able to design Verilog code from a given circuit diagram. By the end of the lab, I had learned some of the more complex applications that can be programmed in the Verilog language.

Equipment Used:

This lab was done through the UNIX environment at the Lassonde Lab. I used the pico editor in UNIX to write my code and I used the built-in Verilog compiler to run and test my programs.

Methods/Procedures:

For LabM1, I needed to create a program that instantiated and tested a 32-bit register. In order to test the enable control input, I had to supply a command-line argument. I needed to hard coding several test values for the clock and data inputs (clk and d). The code was provided to us in the lab manual.


enable
clk
d
REGISTER
z
z = q at all times, and q becomes d @pos edge of clk if enable.

Pseudo Code LabM1.v	reg [31:0] d; reg clk, enable, flag; wire [31:0] z; register #(32) mine(z, d, clk, enable); flag → ("enable=%b", enable); d → 15; clk = 0; #1 $display (clk, d, z); d → 20; clk = 1; #1 $display (clk, d, z); d → 25; clk = 0; #1

	$display (clk, d, z); d → 30; clk = 1; #1 $display (clk, d, z);

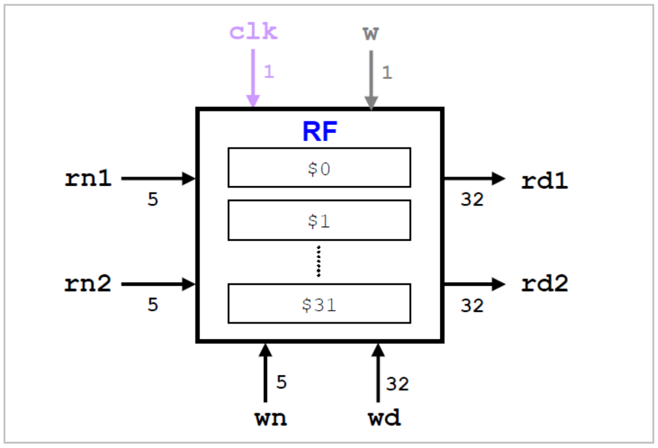
For LabM2, I needed to generalize the above tester by generating random values for d every two units of time as well as set the clock cycle time to 5 units (a code template was provided to us). In order to capture the values of the signals, I used the monitor command rather than display in order to capture changes as they occur.

Pseudo Code
LabM2.v
	reg [31:0] d, e;
	reg clk, enable, flag;
	wire [31:0] z;
	register #(32) mine(z, d, clk, enable);
	
	initial
	begin
	 clk = 0;
	 flag →(enable);
	
	 repeat(20)
	 begin
	  #2 d → $random;
	 end
	
	always
	begin
	 #5 clk → ~clk;
	 if (clk → 1 && enable → 1)
	  e = d;
	
	$monitor (clk, d, z, e);

For LabM3, I needed to simulate a register file. In order to do this, I needed to design a program that set each register to the square of its register number. Pseudo code was provided to us:

```
flag = $value$plusargs("w=%b", w);
  for (i = 0; i < 32; i = i + 1)
              begin
             clk = 0;
           wd = i * i;
             wn = i;
             clk = 1;
               #1;
               end
```

The program had to generate random values for rn1 and rn2 and then output the corresponding contents of rd1 and rd2. This was done in a loop that repeated 10 times.

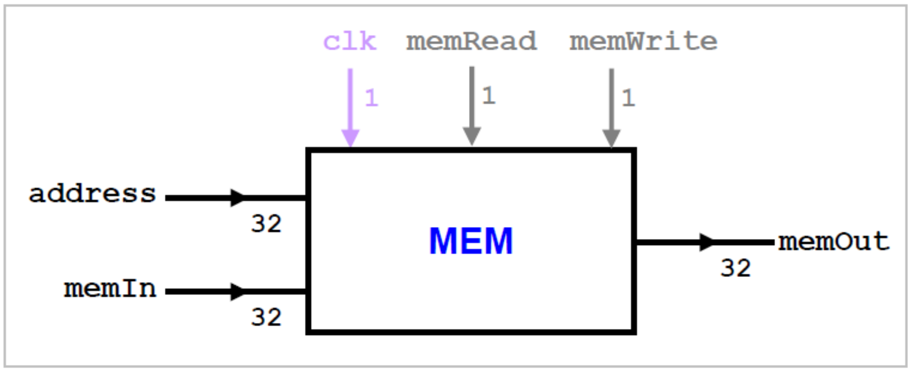


Pseudo Code LabM3.v	reg [31:0]wd;
	reg clk, w, flag;
	reg [4:0] rn1, rn2, wn;
	wire [31:0] rd1, rd2;
	integer i;
	
	rf myRF(rd1, rd2, rn1, rn2, wn, wd, clk, w);
	 clk → 0;
	 flag →(w);
	
	 for (i → 0; i < 32; i ++)
	 begin
	  clk → 0;
	  wd → i * i;
	  wn → i;
	  clk → 1;
	
	 repeat (10)
	 begin
	  rn1 = $random % 32;
	  rn2 = $random % 32;
	$display ($time, rn1, rn2, rd1, rd2);

For LabM4, I needed to simulate memory. I needed to instantiate mem as follows:

```
mem data(memOut, address, memIn, clk, read, write);
```

I needed to store the value of 32'h12345678 at address 16 and the value 32'h89abcdef at address 24. Both of these are word addresses.



Pseudo Code LabM4.v	reg [31:0] address, memIn;
	reg clk, read, write, flag;
	wire [31:0] memOut;
	integer i;
	
	mem data(memOut, address, memIn, clk, read, write);
	
	clk → 0;
	write → 1;
	memIn → 32'h12345678;
	address → 16;
	clk → 1;
	
	clk → 0;
	write → 1;
	memIn → 32'h89abcdef;
	address → 24;
	clk → 1;
	
	write → 0;
	read → 1;
	address → 16;
	
	repeat (3)
	begin
	$display (address, memOut);
	address = address + 4;

For LabM5, I needed to create a memory map in MIPS in order to initialize my memory. My program, LabM5.v, would display the program in ram.dat (my memory initialization program). Much of the code was provided to us.

<u>MIPS Code:</u>

```
      .text
main: add   $t5,  $0,   $0      # index
add   $s0,  $0,   $0            # sum
add   $a0,  $0,   $0            # or reduction
loop: lw    $t0,  0x50($t5)     # loop: t0 = array[t5]
beq   $t0,  $0,   done          # if (t0 == 0) done
add   $s0,  $s0,  $t0
or    $a0,  $a0,  $t0
addi  $t5,  $t5,  4             # t5++
j loop
done: sw    $s0,  0x20($0)      # done: save s0
      sw    $a0,  0x24($0)      # save a0
```

Pseudo Code LabL5.v	reg clk, read, write;
	reg [31:0] address, memIn;
	wire [31:0] memOut;
	
	mem data(memOut, address, memIn, clk, read, write);
	
	address → 32'h80; write → 0; read → 1;
	
	repeat (11)
	$display (address, memOut);
	address → address + 4;

Pseudo Code ram.dat	@20 00000000 @24 00000000 @50 00000001 @54 00000003 @58 00000005 @5C 00000007 @60 00000009 @64 0000000B @68 00000000 @80 00006820 @84 00008020 @88 00002020 @8C 8da80050 @90 11000004 @94 02088020 @98 00882025

	@9C 21ad0004 @A0 08000023 @A4 ac100020 @A8 ac040024

For LabM6, I needed to modify LabM5 so that it displayed the memory content in a format that was instruction-aware. For example, if the instruction was an I-type, then the output for the contents was its opCode, two registers, and immediate, as four separate outputs.

Pseudo Code
LabM6.v
	reg clk, read, write;
	reg [31:0] address, memIn;
	wire [31:0] memOut;
	
	mem data(memOut, address, memIn, clk, read, write);
	
	initial
	begin
	 address → 32'h80; write → 0; read → 1;
	
	 repeat (11)
	
	  if (memOut[31:26] → 0)
	   $display (memOut[31:26], memOut[25:21], memOut[20:16], memOut[15:11], memOut[10:6], memOut[5:0]);
	  else if (memOut[31:26] → 2)
	   $display (memOut[31:26], memOut[25:0]);
	  else
	   $display (memOut[31:26], memOut[25:21], memOut[20:16], memOut[15:0]);
	
	  address → address + 4;

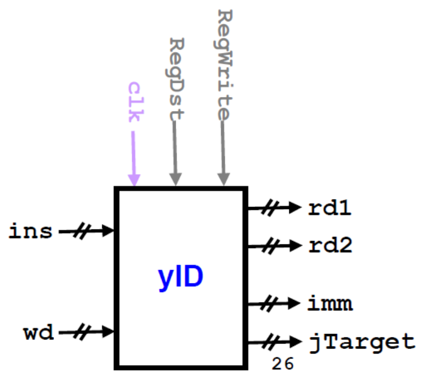
For LabM7, I needed to begin building the data path of the CPU. The first step towards this was by implementing a circuit for the **i**nstruction **f**etch of the memory. I needed to create a program that tested and initialized my yIF module.

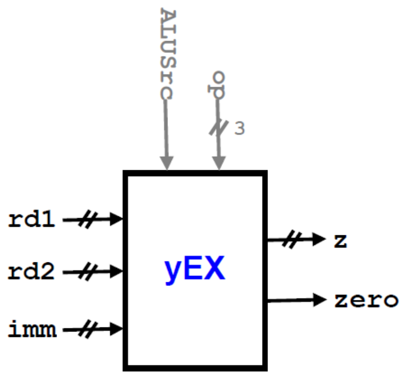
Pseudo Code
LabM7.v
	reg clk;
	reg [31:0] PCin;
	wire [31:0] ins, PCp4;
	
	yIF myIF(ins, PCp4, PCin, clk);
	 PCin → 128;

	repeat (11)
	begin
	clk → 1; #1
	clk → 0; #1
	$display (ins);
	
	PCin → PCp4;

Pseudo Code
yIF.v
	// Input and Output was Defined in the Lab Manual
	
	yAlu my_alu(PCp4, ex, mem_addr, 32'd4, 3'b010);
	register #(32) my_reg(mem_addr, PCin, clk, 1'b1);
	mem my_mem(ins, mem_addr, memIn, clk, 1'b1, 1'b0);

For LabM8, we attempted to build the components for instruction decoding and for instruction execution. Their block diagrams are included below.





I had to create two different modules for them, yID and yEX respectively, and use my LabM8.v file to instantiate and test their functionality.

Pseudo Code
LabM8.v
	reg [31:0] PCin;
	reg clk, RegDst, RegWrite, ALUSrc;
	reg [2:0] op;
	wire [31:0] wd, rd1, rd2, imm, ins, PCp4, z;
	wire [25:0] jTarget;
	wire zero;
	
	yIF myIF(ins, PCp4, PCin, clk);
	yID myID(rd1, rd2, imm, jTarget, ins, wd, RegDst, RegWrite, clk);
	yEX myEX(z, zero, rd1, rd2, imm, op, ALUSrc);
	assign wd = z;

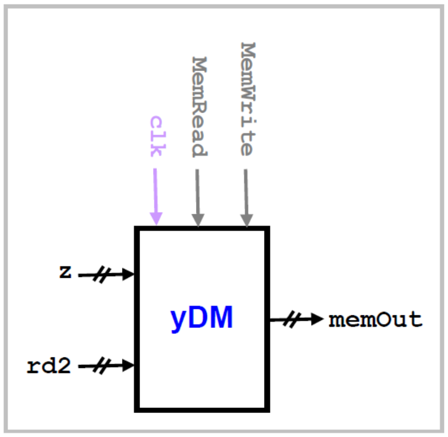
	PCin = 128;
	repeat (11)
	
	 clk = 1;
	  RegDst = 0; RegWrite = 0; ALUSrc = 1; op = 3'b010;
	  if (ins[31:26] → 0)
	   RegDst = 1; RegWrite = 1; ALUSrc = 0;
	  else if (ins[31:26] → 2)
	   RegDst = 0; RegWrite = 0; ALUSrc = 1;
	   if (ins[31:26] → 6'h23) // load word
	    RegDst = 0; RegWrite = 1; ALUSrc = 1;
	   else if (ins[31:26] → 6'h2b) // store word
	    RegDst = 0; RegWrite = 0; ALUSrc = 1;
	   else if (ins[31:26] → 6'h4) // beq
	    RegDst = 0; RegWrite = 0; ALUSrc = 0;
	  clk = 0; #1
	  $display (ins, rd1, rd2, imm, jTarget, z, zero);
	  PCin = PCp4;

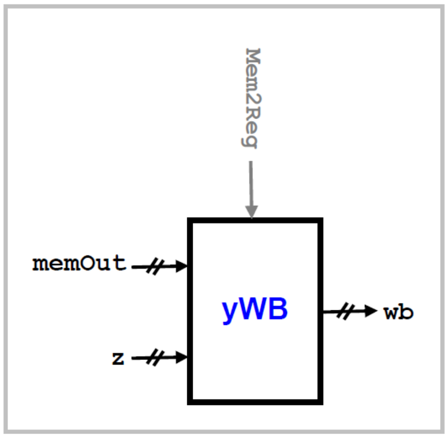
Pseudo Code
yEX.v
	output [31:0] z;
	output zero;
	input [31:0] rd1, rd2, imm;
	input [2:0] op;
	input ALUSrc;
	wire [31:0] b;
	
	yMux #(32) my_mux(b, rd2, imm, ALUSrc);
	yAlu my_alu(z, zero, rd1, b, op);

Pseudo Code
yID.v
	output [31:0] rd1, rd2, imm;
	output [25:0] jTarget;
	input [31:0] ins, wd;
	input RegDst, RegWrite, clk;
	wire [4:0] wn, rn1, rn2;
	
	assign rn1 → ins[25:21];
	assign rn2 →ins[20:16];
	assign imm[15:0] → ins[15:0];
	yMux #(16) se(imm[31:16], 16'b0000000000000000,
	16'b1111111111111111, ins[15]);
	assign jTarget → ins[25:0];
	yMux #(5) my_mux(wn, rn2, ins[15:11], RegDst);
	rf myRF(rd1, rd2, rn1, rn2, wn, wd, clk, RegWrite);

For LabM9, I had to complete the data path with the last two components:

- yDM (data memory): a data memory unit that reads from address z or writes rd2 to that address based on two control signals and a clock.
- yWB (write back): a 2-to-1 mux that selects either memOut or z based on whether the control signal Mem2Reg is 1 or 0, respectively.





Pseudo Code
LabM9.v
	reg [31:0] PCin;
	reg clk, RegDst, RegWrite, ALUSrc, Mem2Reg, MemRead, MemWrite;
	reg [2:0] op;
	wire [31:0] wd, rd1, rd2, imm, ins, PCp4, z, memOut, wb;
	wire [25:0] jTarget;
	wire zero;
	
	yIF myIF(ins, PCp4, PCin, clk);
	yID myID(rd1, rd2, imm, jTarget, ins, wd, RegDst, RegWrite, clk);
	yEX myEX(z, zero, rd1, rd2, imm, op, ALUSrc);
	yDM myDM(memOut, z, rd2, clk, MemRead, MemWrite);
	yWB myWB(wb, z, memOut, Mem2Reg);
	
	assign wd = wb;
	
	PCin = 128;
	 repeat (11)
	 begin
	  clk = 1; #1
	  RegDst = 0; RegWrite = 0; ALUSrc = 1; MemRead = 0; MemWrite = 0; Mem2Reg = 0; op = 3'b010;
	
	  if (ins[31:26] → 0)

```
 begin
  // R
  RegDst = 1; RegWrite = 1; ALUSrc = 0;
  MemRead = 0; MemWrite = 0; Mem2Reg = 0;
 end
 else if (ins[31:26] → 2)
 begin
  RegDst = 0; RegWrite = 0; ALUSrc = 1;
  MemRead = 0; MemWrite = 0; Mem2Reg = 0;
 else
  if (ins[31:26] → 6'h23) // load word
  begin
   RegDst = 0; RegWrite = 1; ALUSrc = 1;
   MemRead = 1; MemWrite = 0; Mem2Reg = 1;
  else if (ins[31:26] → 6'h2b) // store word
  begin
   RegDst = 0; RegWrite = 0; ALUSrc = 1;
   MemRead = 0; MemWrite = 1; Mem2Reg = 0;
  else if (ins[31:26] → 6'h4) // beq
  begin
   RegDst = 0; RegWrite = 0; ALUSrc = 0;
   MemRead = 0; MemWrite = 0; Mem2Reg = 0;
 clk = 0; #1
 $display (ins, rd1, rd2, z, zero, wb);

 PCin = PCp4;
```

Pseudo Code
yDM.v

```
output [31:0] memOut;
input [31:0] exeOut, rd2;
input clk, MemRead, MemWrite;
mem my_mem(memOut, exeOut, rd2, clk, MemRead,
MemWrite);
```

Pseudo Code
yWB.v

```
output [31:0] wb;
input [31:0] exeOut, memOut;
input Mem2Reg;

yMux #(32) my_mux (wb, exeOut, memOut, Mem2Reg);
```

For LabM10, I had to modify LabM9 to p*repare for the next ins* section in order to incorporate branches and jumps: rather than always setting PCin to PCp4.

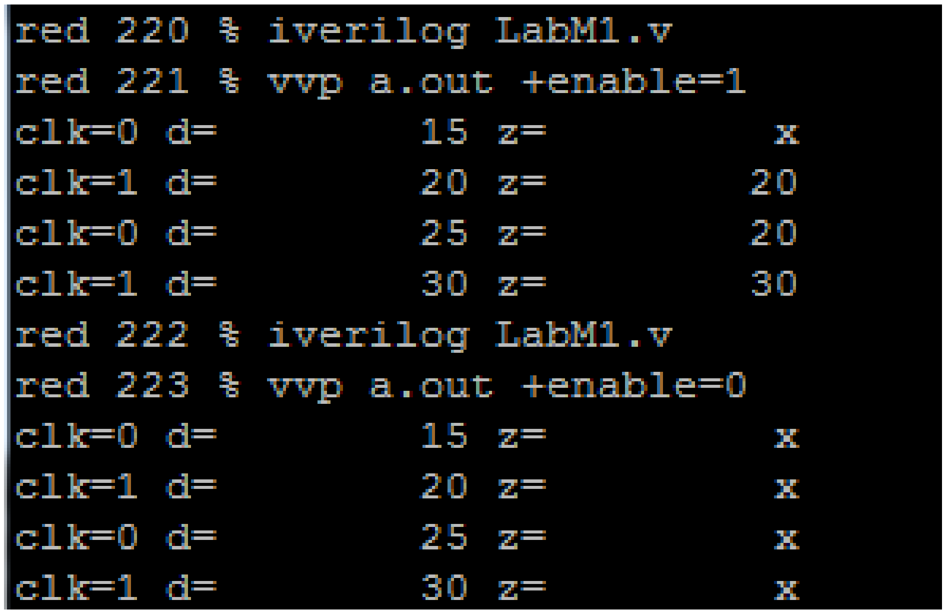
Pseudo Code
LabM10.v

```
reg [31:0] PCin;
reg clk, RegDst, RegWrite, ALUSrc, Mem2Reg, MemRead,
MemWrite;
reg beq, j;
reg [1*8:0] ins_type;
reg [2:0] op;
wire [31:0] wd, rd1, rd2, imm, ins, PCp4, z, memOut, wb;
wire [25:0] jTarget;
wire zero;

yIF myIF(ins, PCp4, PCin, clk);
yID myID(rd1, rd2, imm, jTarget, ins, wd, RegDst, RegWrite, clk);
yEX myEX(z, zero, rd1, rd2, imm, op, ALUSrc);
yDM myDM(memOut, z, rd2, clk, MemRead, MemWrite);
yWB myWB(wb, z, memOut, Mem2Reg);

 if (ins[31:26] == 6'h0)

  RegDst = 1; RegWrite = 1; ALUSrc = 0;
  MemRead = 0; MemWrite = 0; Mem2Reg = 0;
  ins_type = "R";

  if (ins[5:0] → 6'h25)
   op = 3'b001;
 end
 else if (ins[31:26] → 6'h2)
  RegDst = 0; RegWrite = 0; ALUSrc = 1;
  MemRead = 0; MemWrite = 0; Mem2Reg = 0;
  j = 1;
  ins_type = "J";
    ins_type = "I";
  if (ins[31:26] → 6'h23) // load word
  begin
   RegDst = 0; RegWrite = 1; ALUSrc = 1;
   MemRead = 1; MemWrite = 0; Mem2Reg = 1;

  else if (ins[31:26] → 6'h2b)
  begin
   RegDst = 0; RegWrite = 0; ALUSrc = 1;
```

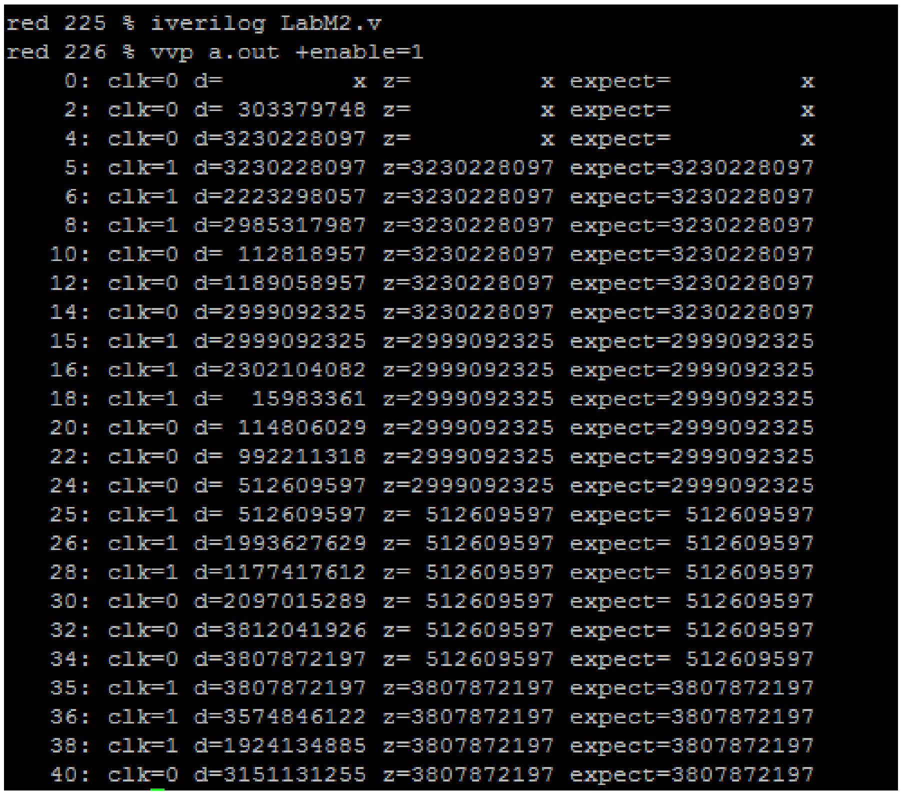
	MemRead = 0; MemWrite = 1; Mem2Reg = 0;
	else if (ins[31:26] → 6'h4)
	
	RegDst = 0; RegWrite = 0; ALUSrc = 0;
	MemRead = 0; MemWrite = 0; Mem2Reg = 0;
	beq = 1;
	else if (ins[31:26] → 6'h8) // addi
	RegDst = 0; RegWrite = 1; ALUSrc = 1;
	MemRead = 0; MemWrite = 0; Mem2Reg = 0;
	clk = 0; #1
	$display(ins, rd1, rd2, z, zero, wb, ins_type, PCin);
	if (beq && zero → 1)
	PCin = PCp4 + (imm << 2);
	else if (j)
	PCin = jTarget << 2;
	else
	PCin = PCp4;

Results:

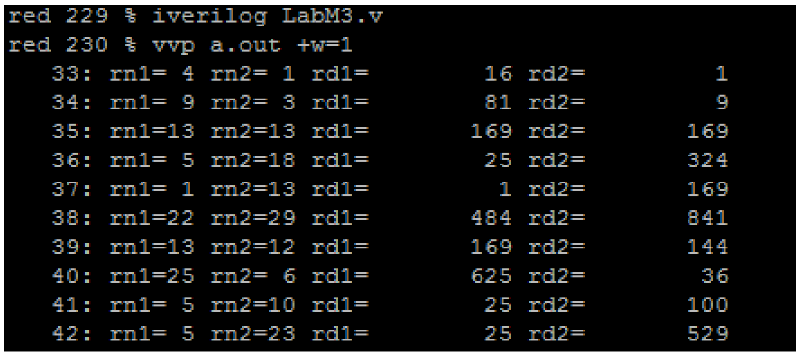
The following terminal screen-shots provide a view of the different results I had during my trial runs.



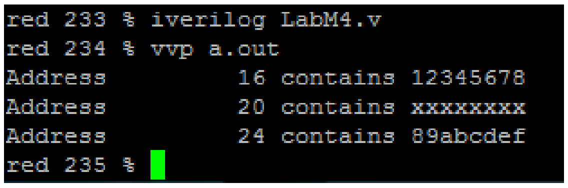
```
red 220 % iverilog LabM1.v
red 221 % vvp a.out +enable=1
clk=0 d=        15 z=        x
clk=1 d=        20 z=       20
clk=0 d=        25 z=       20
clk=1 d=        30 z=       30
red 222 % iverilog LabM1.v
red 223 % vvp a.out +enable=0
clk=0 d=        15 z=        x
clk=1 d=        20 z=        x
clk=0 d=        25 z=        x
clk=1 d=        30 z=        x
```



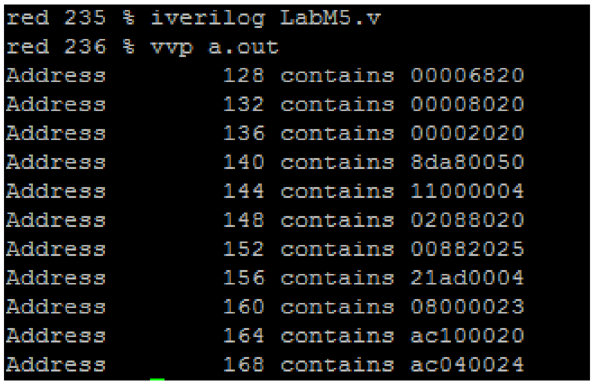
```
red 225 % iverilog LabM2.v
red 226 % vvp a.out +enable=1
    0: clk=0 d=         x z=         x expect=         x
    2: clk=0 d= 303379748 z=         x expect=         x
    4: clk=0 d=3230228097 z=         x expect=         x
    5: clk=1 d=3230228097 z=3230228097 expect=3230228097
    6: clk=1 d=2223298057 z=3230228097 expect=3230228097
    8: clk=1 d=2985317987 z=3230228097 expect=3230228097
   10: clk=0 d= 112818957 z=3230228097 expect=3230228097
   12: clk=0 d=1189058957 z=3230228097 expect=3230228097
   14: clk=0 d=2999092325 z=3230228097 expect=3230228097
   15: clk=1 d=2999092325 z=2999092325 expect=2999092325
   16: clk=1 d=2302104082 z=2999092325 expect=2999092325
   18: clk=1 d=  15983361 z=2999092325 expect=2999092325
   20: clk=0 d= 114806029 z=2999092325 expect=2999092325
   22: clk=0 d= 992211318 z=2999092325 expect=2999092325
   24: clk=0 d= 512609597 z=2999092325 expect=2999092325
   25: clk=1 d= 512609597 z= 512609597 expect= 512609597
   26: clk=1 d=1993627629 z= 512609597 expect= 512609597
   28: clk=1 d=1177417612 z= 512609597 expect= 512609597
   30: clk=0 d=2097015289 z= 512609597 expect= 512609597
   32: clk=0 d=3812041926 z= 512609597 expect= 512609597
   34: clk=0 d=3807872197 z= 512609597 expect= 512609597
   35: clk=1 d=3807872197 z=3807872197 expect=3807872197
   36: clk=1 d=3574846122 z=3807872197 expect=3807872197
   38: clk=1 d=1924134885 z=3807872197 expect=3807872197
   40: clk=0 d=3151131255 z=3807872197 expect=3807872197
```



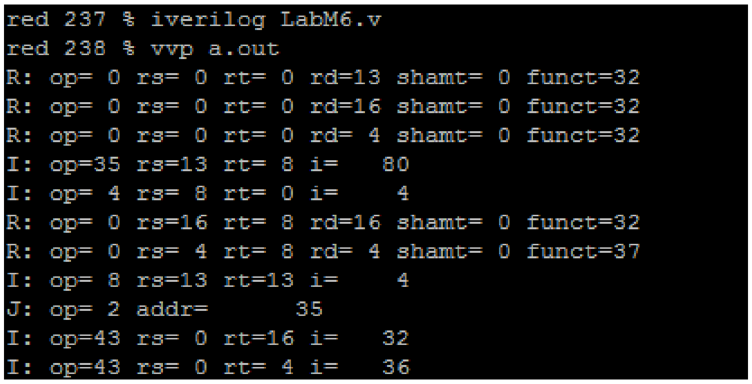
```
red 229 % iverilog LabM3.v
red 230 % vvp a.out +w=1
   33: rn1= 4 rn2= 1 rd1=        16 rd2=         1
   34: rn1= 9 rn2= 3 rd1=        81 rd2=         9
   35: rn1=13 rn2=13 rd1=       169 rd2=       169
   36: rn1= 5 rn2=18 rd1=        25 rd2=       324
   37: rn1= 1 rn2=13 rd1=         1 rd2=       169
   38: rn1=22 rn2=29 rd1=       484 rd2=       841
   39: rn1=13 rn2=12 rd1=       169 rd2=       144
   40: rn1=25 rn2= 6 rd1=       625 rd2=        36
   41: rn1= 5 rn2=10 rd1=        25 rd2=       100
   42: rn1= 5 rn2=23 rd1=        25 rd2=       529
```



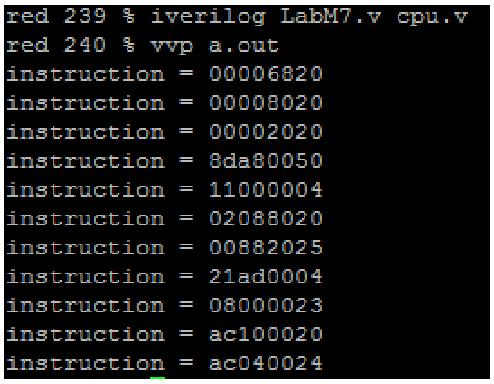
```
red 233 % iverilog LabM4.v
red 234 % vvp a.out
Address         16 contains 12345678
Address         20 contains xxxxxxxx
Address         24 contains 89abcdef
red 235 %
```



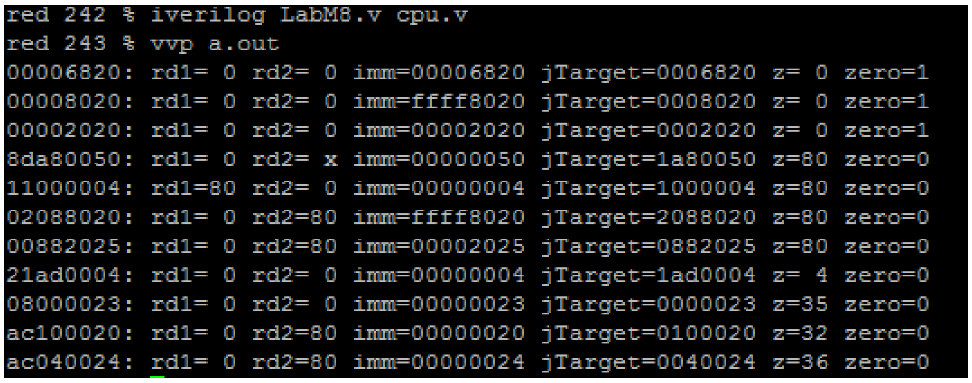
```
red 235 % iverilog LabM5.v
red 236 % vvp a.out
Address        128 contains 00006820
Address        132 contains 00008020
Address        136 contains 00002020
Address        140 contains 8da80050
Address        144 contains 11000004
Address        148 contains 02088020
Address        152 contains 00882025
Address        156 contains 21ad0004
Address        160 contains 08000023
Address        164 contains ac100020
Address        168 contains ac040024
```



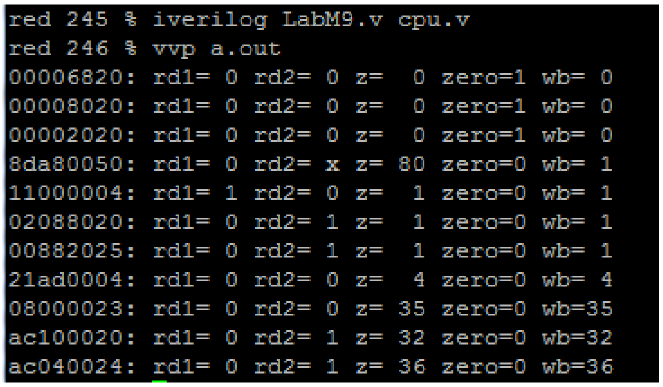
```
red 237 % iverilog LabM6.v
red 238 % vvp a.out
R: op= 0 rs= 0 rt= 0 rd=13 shamt= 0 funct=32
R: op= 0 rs= 0 rt= 0 rd=16 shamt= 0 funct=32
R: op= 0 rs= 0 rt= 0 rd= 4 shamt= 0 funct=32
I: op=35 rs=13 rt= 8 i=   80
I: op= 4 rs= 8 rt= 0 i=    4
R: op= 0 rs=16 rt= 8 rd=16 shamt= 0 funct=32
R: op= 0 rs= 4 rt= 8 rd= 4 shamt= 0 funct=37
I: op= 8 rs=13 rt=13 i=    4
J: op= 2 addr=      35
I: op=43 rs= 0 rt=16 i=   32
I: op=43 rs= 0 rt= 4 i=   36
```



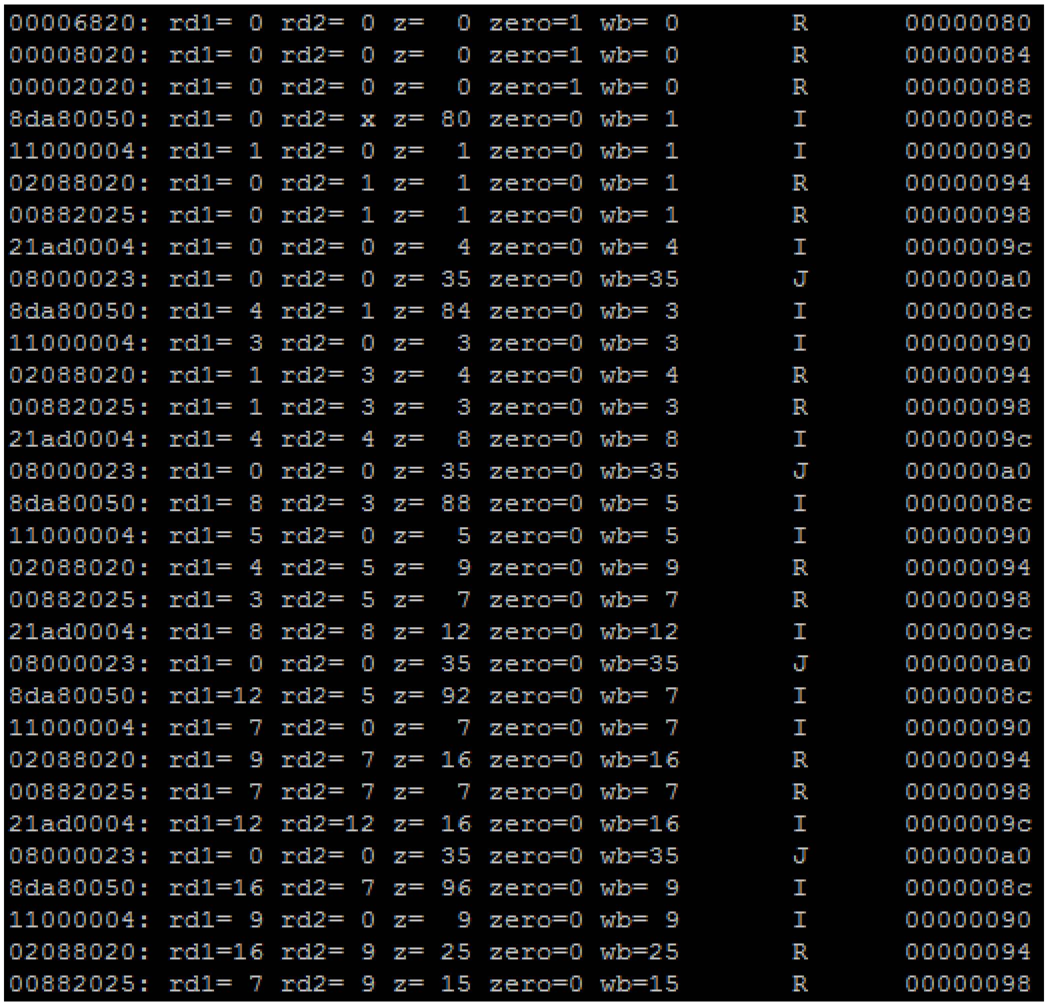
```
red 239 % iverilog LabM7.v cpu.v
red 240 % vvp a.out
instruction = 00006820
instruction = 00008020
instruction = 00002020
instruction = 8da80050
instruction = 11000004
instruction = 02088020
instruction = 00882025
instruction = 21ad0004
instruction = 08000023
instruction = ac100020
instruction = ac040024
```



```
red 242 % iverilog LabM8.v cpu.v
red 243 % vvp a.out
00006820: rd1= 0 rd2= 0 imm=00006820 jTarget=0006820 z= 0 zero=1
00008020: rd1= 0 rd2= 0 imm=ffff8020 jTarget=0008020 z= 0 zero=1
00002020: rd1= 0 rd2= 0 imm=00002020 jTarget=0002020 z= 0 zero=1
8da80050: rd1= 0 rd2= x imm=00000050 jTarget=1a80050 z=80 zero=0
11000004: rd1=80 rd2= 0 imm=00000004 jTarget=1000004 z=80 zero=0
02088020: rd1= 0 rd2=80 imm=ffff8020 jTarget=2088020 z=80 zero=0
00882025: rd1= 0 rd2=80 imm=00002025 jTarget=0882025 z=80 zero=0
21ad0004: rd1= 0 rd2= 0 imm=00000004 jTarget=1ad0004 z= 4 zero=0
08000023: rd1= 0 rd2= 0 imm=00000023 jTarget=0000023 z=35 zero=0
ac100020: rd1= 0 rd2=80 imm=00000020 jTarget=0100020 z=32 zero=0
ac040024: rd1= 0 rd2=80 imm=00000024 jTarget=0040024 z=36 zero=0
```



```
red 245 % iverilog LabM9.v cpu.v
red 246 % vvp a.out
00006820: rd1= 0 rd2= 0 z=  0 zero=1 wb= 0
00008020: rd1= 0 rd2= 0 z=  0 zero=1 wb= 0
00002020: rd1= 0 rd2= 0 z=  0 zero=1 wb= 0
8da80050: rd1= 0 rd2= x z= 80 zero=0 wb= 1
11000004: rd1= 1 rd2= 0 z=  1 zero=0 wb= 1
02088020: rd1= 0 rd2= 1 z=  1 zero=0 wb= 1
00882025: rd1= 0 rd2= 1 z=  1 zero=0 wb= 1
21ad0004: rd1= 0 rd2= 0 z=  4 zero=0 wb= 4
08000023: rd1= 0 rd2= 0 z= 35 zero=0 wb=35
ac100020: rd1= 0 rd2= 1 z= 32 zero=0 wb=32
ac040024: rd1= 0 rd2= 1 z= 36 zero=0 wb=36
```



```
00006820: rd1= 0 rd2= 0 z=  0 zero=1 wb= 0       R       00000080
00008020: rd1= 0 rd2= 0 z=  0 zero=1 wb= 0       R       00000084
00002020: rd1= 0 rd2= 0 z=  0 zero=1 wb= 0       R       00000088
8da80050: rd1= 0 rd2= x z= 80 zero=0 wb= 1       I       0000008c
11000004: rd1= 1 rd2= 0 z=  1 zero=0 wb= 1       I       00000090
02088020: rd1= 0 rd2= 1 z=  1 zero=0 wb= 1       R       00000094
00882025: rd1= 0 rd2= 1 z=  1 zero=0 wb= 1       R       00000098
21ad0004: rd1= 0 rd2= 0 z=  4 zero=0 wb= 4       I       0000009c
08000023: rd1= 0 rd2= 0 z= 35 zero=0 wb=35       J       000000a0
8da80050: rd1= 4 rd2= 1 z= 84 zero=0 wb= 3       I       0000008c
11000004: rd1= 3 rd2= 0 z=  3 zero=0 wb= 3       I       00000090
02088020: rd1= 1 rd2= 3 z=  4 zero=0 wb= 4       R       00000094
00882025: rd1= 1 rd2= 3 z=  3 zero=0 wb= 3       R       00000098
21ad0004: rd1= 4 rd2= 4 z=  8 zero=0 wb= 8       I       0000009c
08000023: rd1= 0 rd2= 0 z= 35 zero=0 wb=35       J       000000a0
8da80050: rd1= 8 rd2= 3 z= 88 zero=0 wb= 5       I       0000008c
11000004: rd1= 5 rd2= 0 z=  5 zero=0 wb= 5       I       00000090
02088020: rd1= 4 rd2= 5 z=  9 zero=0 wb= 9       R       00000094
00882025: rd1= 3 rd2= 5 z=  7 zero=0 wb= 7       R       00000098
21ad0004: rd1= 8 rd2= 8 z= 12 zero=0 wb=12       I       0000009c
08000023: rd1= 0 rd2= 0 z= 35 zero=0 wb=35       J       000000a0
8da80050: rd1=12 rd2= 5 z= 92 zero=0 wb= 7       I       0000008c
11000004: rd1= 7 rd2= 0 z=  7 zero=0 wb= 7       I       00000090
02088020: rd1= 9 rd2= 7 z= 16 zero=0 wb=16       R       00000094
00882025: rd1= 7 rd2= 7 z=  7 zero=0 wb= 7       R       00000098
21ad0004: rd1=12 rd2=12 z= 16 zero=0 wb=16       I       0000009c
08000023: rd1= 0 rd2= 0 z= 35 zero=0 wb=35       J       000000a0
8da80050: rd1=16 rd2= 7 z= 96 zero=0 wb= 9       I       0000008c
11000004: rd1= 9 rd2= 0 z=  9 zero=0 wb= 9       I       00000090
02088020: rd1=16 rd2= 9 z= 25 zero=0 wb=25       R       00000094
00882025: rd1= 7 rd2= 9 z= 15 zero=0 wb=15       R       00000098
```

<u>Discussion:</u>

The following are the answers to the questions listed in the lab manual.

Q4: We should have four printed lines, each containing the values of clk, d, and z. Clk alternates between 0 and 1 twice. The value of b should be 15 20 25 30. The value of z on each of these lines should be x because we have not initialized a value for enable.

Q5: Yes the register behaves as expected in both test cases. When we set enable to 1, the values of d are equal to those of z. When we set enable to 0, z is equal to x for all four lines. This is because z has no value set when the inputted value is 0.

Q10: Yes, the registers behaved as expected. This behaviour is the value of z changing when clk is equal to 1.

Q14: My program doesn’t run as expected when my w is set. Our rn1 and rn2 are set to random numbers between 0 and 31, which are then multiplied by themselves (their square) and then outputted through rd1 and rd2.

Q15: All our output is equal t0 x (the value of rd1, rd2), therefore the registers perform as expected.

Q20: I predict that my program will output three lines. It will start with the address (16 20 24) and then hat they contain (16 →12345678) (24 →89abcdef). For 20, the value should be xxxxxxxx.

Q21: Yes, my program behaves the way I expected.

Q22: If I were to try to pass an un-aligned address, the output will be xxxxxxxx. Addresses should be divisible by 4. In this case, they are not, thus, making our addresses inaccessible.

Q33: The clk input of mem can be deemed irrelevant due to the fact that it’s only used during memwrite. This conclusion was drawn from Lab4 where writing it would be saved on the edge of clk, but when we read it, we only set the address. The value does not change from pc reg to mem.

Q50: The output is in fact consistent with the data path (We go instruction by instruction; jump branch works the same).

Q59: The loop must repeat 43 times rather than 11 because there is a loop inside the jump statement, therefore, causing it to run more than 11 times.

Conclusion:

This lab taught me how to build a variety of components that make up a CPU. My CPU’s data path has all the computational units needed for executing the above instructions but must rely on an external agent to supply the correct control signals for each instruction. As well as learning how to build different modules, this lab continued to teach me how to use the concept of structural modeling.

References:

No external references were used for this lab other than the lab material provided to us at the beginning of the lab.

Appendix:

The source codes for all 10 experiments as well as the auxiliary programs are included below.

cpu.v:

```
module yMux1(z, a, b, c);

output z;

input a, b, c;

wire notC, upper, lower;

not my_not(notC, c);

and upperAnd(upper, a, notC);

and lowerAnd(lower, c, b);

or my_or(z, upper, lower);

endmodule

module yMux(z, a, b, c);

parameter SIZE = 2;

output [SIZE-1:0] z;

input [SIZE-1:0] a, b;

input c;


// usage:

// yMux #(64) my_mux(z,a,b,c);

// or

// yMux #(.SIZE(64) my_mux(z,a,b,c);

// or

// defparam my_mux.SIZE = 64;

// yMux my_mux(z,a,b,c);


yMux1 mine[SIZE-1:0](z, a, b, c);
```

```
endmodule

module yMux4to1(z,a0,a1,a2,a3,c);

parameter SIZE = 2;

output [SIZE-1:0] z;

input [SIZE-1:0] a0, a1, a2, a3;

input [1:0] c;

wire [SIZE-1:0] zLo, zHi;


yMux #(SIZE) lo(zLo, a0, a1, c[0]);

yMux #(SIZE) hi(zHi, a2, a3, c[0]);

yMux #(SIZE) final(z, zLo, zHi, c[1]);


endmodule

module yAdder1(z, cout, a, b, cin);

output z, cout;

input a, b, cin;


xor left_xor (tmp, a, b);

xor right_xor (z, cin, tmp);

and left_and(outL, a, b);

and right_and(outR, tmp, cin);

or my_or(cout, outR, outL);


endmodule

module yAdder(z, cout, a, b, cin);

output [31:0] z;
output cout;

input [31:0] a, b;

input cin;

wire [31:0] in, out;


yAdder1 mine[31:0](z, out, a, b, in);


assign in[0] = cin;

assign in[31:1] = out[30:0];

assign cout = out[31];


endmodule

module yArith(z, cout, a, b, ctrl);

output [31:0] z;

output cout;

input [31:0] a, b;

input ctrl;

wire [31:0] notB, tmp;

wire cin;


yAdder mine(z, out, a, notB, cin);


assign cin = ctrl ? 1 : 0;

assign notB = ctrl ? ~b : b;


endmodule

module yAlu(z, ex, a, b, op);
```

```
input [31:0] a, b;

input [2:0] op;

output [31:0] z;

output ex;

wire [31:0] arith_z, and_z, or_z, slt_z;

wire [15:0] or16;

wire [7:0] or8;

wire [3:0] or4;

wire [1:0] or2;

wire cout;


//assign slt_z[31:1] = 0;

assign slt_z = (a[31] ^ b[31]) ? a[31] : arith_z[31];

and my_and[31:0] (and_z, a, b);

or my_or[31:0] (or_z, a, b);

//not my_nor[31:0] (nor_z, or_z, 32'b11111111111111111111111111111111);


or or_16[15:0] (or16, z[31:16], z[15:0]);

or or_8[7:0] (or8, or16[15:8], or16[7:0]);

or or_4[3:0] (or4, or8[7:4], or8[3:0]);

or or_2[1:0] (or2, or4[3:2], or4[1:0]);

nor nor_1  (ex, or2[1],  or2[0]);


//assign ex = ~(|z);


yArith my_arith(arith_z, cout, a, b, op[2]);

yMux4to1 #(32) my_mux(z, and_z, or_z, arith_z, slt_z, op[1:0]);
```

```
endmodule

module yMux2(z, a, b, c);

output [1:0] z;

input [1:0] a, b;

input c;


yMux1 upper(z[0], a[0], b[0], c);

yMux1 lower(z[1], a[1], b[1], c);


endmodule

module yIF(ins, PCp4, PCin, clk);

output [31:0] ins, PCp4;

input [31:0] PCin;

input clk;

reg [31:0] memIn;

wire [31:0] mem_addr;


yAlu my_alu(PCp4, ex, PCin, 32'd4, 3'b010);

register #(32) my_reg(mem_addr, PCin, clk, 1'b1);

mem my_mem(ins, mem_addr, memIn, clk, 1'b1, 1'b0);


endmodule

module yID(rd1, rd2, imm, jTarget, ins, wd, RegDst, RegWrite, clk);

output [31:0] rd1, rd2, imm;

output [25:0] jTarget;

input [31:0] ins, wd;
input RegDst, RegWrite, clk;

wire [4:0] wn, rn1, rn2;


assign rn1 = ins[25:21];

assign rn2 = ins[20:16];

assign imm[15:0] = ins[15:0];

yMux #(16) se(imm[31:16], 16'b0000000000000000, 16'b1111111111111111, ins[15]);

assign jTarget = ins[25:0];


yMux #(5) my_mux(wn, rn2, ins[15:11], RegDst);


rf myRF(rd1, rd2, rn1, rn2, wn, wd, clk, RegWrite);


endmodule

module yEX(z, zero, rd1, rd2, imm, op, ALUSrc);

output [31:0] z;

output zero;

input [31:0] rd1, rd2, imm;

input [2:0] op;

input ALUSrc;

wire [31:0] b;


yMux #(32) my_mux(b, rd2, imm, ALUSrc);

yAlu my_alu(z, zero, rd1, b, op);


endmodule

// data memory
```

```
module yDM(memOut, exeOut, rd2, clk, MemRead, MemWrite);

output [31:0] memOut;

input [31:0] exeOut, rd2;

input clk, MemRead, MemWrite;


mem my_mem(memOut, exeOut, rd2, clk, MemRead, MemWrite);


endmodule

// write back

module yWB(wb, exeOut, memOut, Mem2Reg);

output [31:0] wb;

input [31:0] exeOut, memOut;

input Mem2Reg;

yMux #(32) my_mux (wb, exeOut, memOut, Mem2Reg);

endmodule
```

LabM1:

```
module labM;

reg [31:0] d;

reg clk, enable, flag;

wire [31:0] z;

register #(32) mine(z, d, clk, enable);

initial

begin

 flag = $value$plusargs ("enable=%b", enable);

 d = 15; clk = 0; #1

 $display ("clk=%b d=%d z=%d", clk, d, z);

 d = 20; clk = 1; #1
 $display ("clk=%b d=%d z=%d", clk, d, z);
 d = 25; clk = 0; #1
 $display ("clk=%b d=%d z=%d", clk, d, z);
 d = 30; clk = 1; #1
 $display ("clk=%b d=%d z=%d", clk, d, z);
 $finish;
end
endmodule
```

LabM2:

```
module labM;
reg [31:0] d, e;
reg clk, enable, flag;
wire [31:0] z;
register #(32) mine(z, d, clk, enable);
initial
begin
 clk = 0;
 flag = $value$plusargs ("enable=%b", enable);
 repeat(20)
 begin
  #2 d = $random;
 end
 $finish;
end
always
begin
 #5 clk = ~clk;
 if (clk === 1 && enable === 1)

  e = d;

end

initial

 $monitor ("%5d: clk=%b d=%d z=%d expect=%d", $time, clk, d, z, e);

endmodule
```

<u>LabM3:</u>

```
module labM;

reg [31:0]wd;

reg clk, w, flag;

reg [4:0] rn1, rn2, wn;

wire [31:0] rd1, rd2;

integer i;

rf myRF(rd1, rd2, rn1, rn2, wn, wd, clk, w);

initial

begin

 clk = 0;

 flag = $value$plusargs ("w=%b", w);

 for (i = 0; i < 32; i = i + 1)

 begin

  clk = 0;

  wd = i * i;

  wn = i;

  clk = 1;

  #1;

 end

 repeat (10)
 begin
  rn1 = $random % 32;
  rn2 = $random % 32;
  #1;
  $display ("%5d: rn1=%d rn2=%d rd1=%d rd2=%d", $time, rn1, rn2, rd1, rd2);
 end
 $finish;
end
endmodule
```

LabM4:

```
module labM;
reg [31:0] address, memIn;
reg clk, read, write, flag;
wire [31:0] memOut;
integer i;
mem data(memOut, address, memIn, clk, read, write);
initial
begin
 clk = 0;
 write = 1;
 memIn = 32'h12345678;
 address = 16;
 clk = 1;
 #1;
 clk = 0;
 write = 1;
 memIn = 32'h89abcdef;
 address = 24;

 clk = 1;

 #1;

 write = 0;

 read = 1;

 address = 16;

 repeat (3)

 begin

  #1 $display ("Address %d contains %h", address, memOut);

  address = address + 4;

 end

 $finish;

end

endmodule
```

LabM5:

```
module labM;

reg clk, read, write;

reg [31:0] address, memIn;

wire [31:0] memOut;

mem data(memOut, address, memIn, clk, read, write);

initial

begin

 address = 32'h80; write = 0; read = 1;

 repeat (11)

 begin

  #1 $display ("Address %d contains %h", address, memOut);

  address = address + 4;
 end
 $finish;
end
endmodule
```

LabM6:

```
module labM;
reg clk, read, write;
reg [31:0] address, memIn;
wire [31:0] memOut;
mem data(memOut, address, memIn, clk, read, write);
initial
begin
 address = 32'h80; write = 0; read = 1;
 repeat (11)
 begin
  #1;
  if (memOut[31:26] == 0)
   $display ("R: op=%d rs=%d rt=%d rd=%d shamt=%d funct=%d", memOut[31:26], memOut[25:21],
memOut[20:16], memOut[15:11], memOut[10:6], memOut[5:0]);
  else if (memOut[31:26] == 2)
   $display ("J: op=%d addr=%d", memOut[31:26], memOut[25:0]);
  else
   $display ("I: op=%d rs=%d rt=%d i=%d", memOut[31:26], memOut[25:21], memOut[20:16], memOut[15:0]);
  address = address + 4;
 end
 $finish;
end
```

```
endmodule
```

LabM7:

```
module labM;
reg clk;
reg [31:0] PCin;
wire [31:0] ins, PCp4;
yIF myIF(ins, PCp4, PCin, clk);
initial
begin
 // entry point
 PCin = 128;
 // run program
 repeat (11)
 begin
  // fetch an ins
  clk = 1; #1
  // execute the ins
  clk = 0; #1
  // view results
  $display ("instruction = %h", ins);
  PCin = PCp4;
 end
 $finish;
end
endmodule
```

LabM8:

```
module labM;
```

```
reg [31:0] PCin;

reg clk, RegDst, RegWrite, ALUSrc;

reg [2:0] op;

wire [31:0] wd, rd1, rd2, imm, ins, PCp4, z;

wire [25:0] jTarget;

wire zero;

yIF myIF(ins, PCp4, PCin, clk);

yID myID(rd1, rd2, imm, jTarget, ins, wd, RegDst, RegWrite, clk);

yEX myEX(z, zero, rd1, rd2, imm, op, ALUSrc);

assign wd = z;

initial

begin

 // entry point

 PCin = 128;

 // run program

 repeat (11)

 begin

  // fetch an ins

  clk = 1; #1

  // set control signals

  RegDst = 0; RegWrite = 0; ALUSrc = 1; op = 3'b010;

  if (ins[31:26] == 0)

  begin

   // R

   RegDst = 1; RegWrite = 1; ALUSrc = 0;

  end

  else if (ins[31:26] == 2)
  begin
   // J
   RegDst = 0; RegWrite = 0; ALUSrc = 1;
  end
  else
  begin
   // I
   if (ins[31:26] === 6'h23) // load word
   begin
    RegDst = 0; RegWrite = 1; ALUSrc = 1;
   end
   else if (ins[31:26] === 6'h2b) // store word
   begin
    RegDst = 0; RegWrite = 0; ALUSrc = 1;
   end
   else if (ins[31:26] === 6'h4) // beq
   begin
    RegDst = 0; RegWrite = 0; ALUSrc = 0;
   end
  end
  // execute the ins
  clk = 0; #1
  // view results
  $display ("%h: rd1=%2d rd2=%2d imm=%h jTarget=%h z=%2d zero=%b", ins, rd1, rd2, imm, jTarget, z, zero);
  PCin = PCp4;
 end
 $finish;
```

```
end

endmodule
```

LabM9:

```
module labM;

reg [31:0] PCin;

reg clk, RegDst, RegWrite, ALUSrc, Mem2Reg, MemRead, MemWrite;

reg [2:0] op;

wire [31:0] wd, rd1, rd2, imm, ins, PCp4, z, memOut, wb;

wire [25:0] jTarget;

wire zero;

yIF myIF(ins, PCp4, PCin, clk);

yID myID(rd1, rd2, imm, jTarget, ins, wd, RegDst, RegWrite, clk);

yEX myEX(z, zero, rd1, rd2, imm, op, ALUSrc);

yDM myDM(memOut, z, rd2, clk, MemRead, MemWrite);

yWB myWB(wb, z, memOut, Mem2Reg);

assign wd = wb;

initial

begin

 // entry point

 PCin = 128;

 // run program

 repeat (11)

 begin

  // fetch an ins

  clk = 1; #1

  // set control signals

  RegDst = 0; RegWrite = 0; ALUSrc = 1; MemRead = 0; MemWrite = 0; Mem2Reg = 0; op = 3'b010;
```

```
if (ins[31:26] == 0)
begin
 // R
 RegDst = 1; RegWrite = 1; ALUSrc = 0;
 MemRead = 0; MemWrite = 0; Mem2Reg = 0;
end
else if (ins[31:26] == 2)
begin
 // J
 RegDst = 0; RegWrite = 0; ALUSrc = 1;
 MemRead = 0; MemWrite = 0; Mem2Reg = 0;
end
else
begin
 // I
 if (ins[31:26] === 6'h23) // load word
 begin
  RegDst = 0; RegWrite = 1; ALUSrc = 1;
  MemRead = 1; MemWrite = 0; Mem2Reg = 1;
 end
 else if (ins[31:26] === 6'h2b) // store word
 begin
  RegDst = 0; RegWrite = 0; ALUSrc = 1;
  MemRead = 0; MemWrite = 1; Mem2Reg = 0;
 end
 else if (ins[31:26] === 6'h4) // beq
 begin
```

```
   RegDst = 0; RegWrite = 0; ALUSrc = 0;

   MemRead = 0; MemWrite = 0; Mem2Reg = 0;

  end

 end

 // execute the ins

 clk = 0; #1

 // view results

 $display("%h: rd1=%2d rd2=%2d z=%3d zero=%b wb=%2d", ins, rd1, rd2, z, zero, wb);

 PCin = PCp4;

 end

 $finish;

end

endmodule
```

LabM10:

```
module labM;

reg [31:0] PCin;

reg clk, RegDst, RegWrite, ALUSrc, Mem2Reg, MemRead, MemWrite;

reg beq, j;

reg [1*8:0] ins_type;

reg [2:0] op;

wire [31:0] wd, rd1, rd2, imm, ins, PCp4, z, memOut, wb;

wire [25:0] jTarget;

wire zero;

yIF myIF(ins, PCp4, PCin, clk);

yID myID(rd1, rd2, imm, jTarget, ins, wd, RegDst, RegWrite, clk);

yEX myEX(z, zero, rd1, rd2, imm, op, ALUSrc);

yDM myDM(memOut, z, rd2, clk, MemRead, MemWrite);
```

```
yWB myWB(wb, z, memOut, Mem2Reg);

assign wd = wb;

initial

begin

 // entry point

 PCin = 'h80;

 // run program

 repeat (43)

 begin

  // fetch an ins

  clk = 1; #1

  // set control signals

  RegDst = 0; RegWrite = 0; ALUSrc = 1; MemRead = 0; MemWrite = 0; Mem2Reg = 0; op = 3'b010;

  beq = 0; j = 0;

  ins_type = "_";

  if (ins[31:26] == 6'h0)

  begin

   // R

   RegDst = 1; RegWrite = 1; ALUSrc = 0;

   MemRead = 0; MemWrite = 0; Mem2Reg = 0;

   ins_type = "R";

   if (ins[5:0] == 6'h25)

    op = 3'b001;

  end

  else if (ins[31:26] == 6'h2)

  begin

   // J
   RegDst = 0; RegWrite = 0; ALUSrc = 1;
   MemRead = 0; MemWrite = 0; Mem2Reg = 0;
   j = 1;
   ins_type = "J";
  end
  else
  begin
   // I
   ins_type = "I";
   if (ins[31:26] === 6'h23) // load word
   begin
    RegDst = 0; RegWrite = 1; ALUSrc = 1;
    MemRead = 1; MemWrite = 0; Mem2Reg = 1;
   end
   else if (ins[31:26] === 6'h2b)
   begin
    RegDst = 0; RegWrite = 0; ALUSrc = 1;
    MemRead = 0; MemWrite = 1; Mem2Reg = 0;
   end
   else if (ins[31:26] === 6'h4) // beq
   begin
    RegDst = 0; RegWrite = 0; ALUSrc = 0;
    MemRead = 0; MemWrite = 0; Mem2Reg = 0;
    beq = 1;
   end
   else if (ins[31:26] === 6'h8) // addi
   begin
```

```
   RegDst = 0; RegWrite = 1; ALUSrc = 1;
   MemRead = 0; MemWrite = 0; Mem2Reg = 0;
  end
 end
   clk = 0; #1
 $display("%h: rd1=%2d rd2=%2d z=%3d zero=%b wb=%2d\t%s\t%h", ins, rd1, rd2, z, zero, wb, ins_type, PCin);
 if (beq && zero === 1)
  PCin = PCp4 + (imm << 2);
 else if (j)
  PCin = jTarget << 2;
 else
  PCin = PCp4;
 end
 $finish;
end
endmodule
```

prog.s:

```
        .text
main:   add $t5, $0, $0 # index
        add $s0, $0, $0 # sum
        add $a0, $0, $0 # or reduction
        loop: lw $t0, 0x50($t5) # loop: t0 = array[t
        beq $t0, $0, done # if (t0 == 0) done
        add $s0, $s0, $t0
        or $a0, $a0, $t0
        addi $t5, $t5, 4 # t5++
        j loop
```

```
done:   sw $s0, 0x20($0) # done: save s0

        sw $a0, 0x24($0) # save a0
```

ram.dat:

```
        @20 00000000 // the sum

        @24 00000000 // the or reduction

        @50 00000001 // array [0]

        @54 00000003 // array [1]

        @58 00000005 // array [2]

        @5C 00000007 // array [3]

        @60 00000009 // array [4]

        @64 0000000B // array [5]

        @68 00000000 // null terminator

        @80 00006820

        @84 00008020

        @88 00002020

        @8C 8da80050

        @90 11000004

        @94 02088020

        @98 00882025

        @9C 21ad0004

        @A0 08000023

        @A4 ac100020

        @A8 ac040024
```

yDM.v:

```
module yDM(memOut, exeOut, rd2, clk, MemRead, MemWrite);

output [31:0] memOut;

input [31:0] exeOut, rd2;
input clk, MemRead, MemWrite;

mem my_mem(memOut, exeOut, rd2, clk, MemRead, MemWrite);

endmodule
```

yEX.v:

```
module yEX(z, zero, rd1, rd2, imm, op, ALUSrc);

output [31:0] z;

output zero;

input [31:0] rd1, rd2, imm;

input [2:0] op;

input ALUSrc;

wire [31:0] b;

yMux #(32) my_mux(b, rd2, imm, ALUSrc);

yAlu my_alu(z, zero, rd1, b, op);

endmodule
```

yID.v:

```
module yID(rd1, rd2, imm, jTarget, ins, wd, RegDst, RegWrite, clk);

output [31:0] rd1, rd2, imm;

output [25:0] jTarget;

input [31:0] ins, wd;

input RegDst, RegWrite, clk;

wire [4:0] wn, rn1, rn2;

assign rn1 = ins[25:21];

assign rn2 = ins[20:16];

assign imm[15:0] = ins[15:0];

yMux #(16) se(imm[31:16], 16'b0000000000000000, 16'b1111111111111111, ins[15]);

assign jTarget = ins[25:0];

yMux #(5) my_mux(wn, rn2, ins[15:11], RegDst);
```

```
rf myRF(rd1, rd2, rn1, rn2, wn, wd, clk, RegWrite);

endmodule
```

<u>yIF.v:</u>

```
module yIF(ins, PCp4, PCin, clk);

output [31:0] ins, PCp4;

input [31:0] PCin;

input clk;

reg [31:0] memIn;

wire [31:0] mem_addr;

yAlu my_alu(PCp4, ex, mem_addr, 32'd4, 3'b010);

register #(32) my_reg(mem_addr, PCin, clk, 1'b1);

mem my_mem(ins, mem_addr, memIn, clk, 1'b1, 1'b0);

endmodule
```

<u>yWB.v:</u>

```
module yWB(wb, exeOut, memOut, Mem2Reg);

output [31:0] wb;

input [31:0] exeOut, memOut;

input Mem2Reg;

yMux #(32) my_mux (wb, exeOut, memOut, Mem2Reg);

endmodule
```