Organization of Digital Computers Lab
EECS 112L

Lab 2

Kyle Zyler Cayanan
80576955

02/01/2022

1 - Objective

For this experiment we are to implement a set of instructions for a MIPS processor shown in the figure below. We only need to implement a few modules in hardware in order to support the jump and branch instructions. All other instructions can be implemented by adding support for the control signals with the control module, ALU control module, and ALU. Two muxes, an and gate, a shifter, and an adder is all that's needed to implement the new instructions hardware wise.

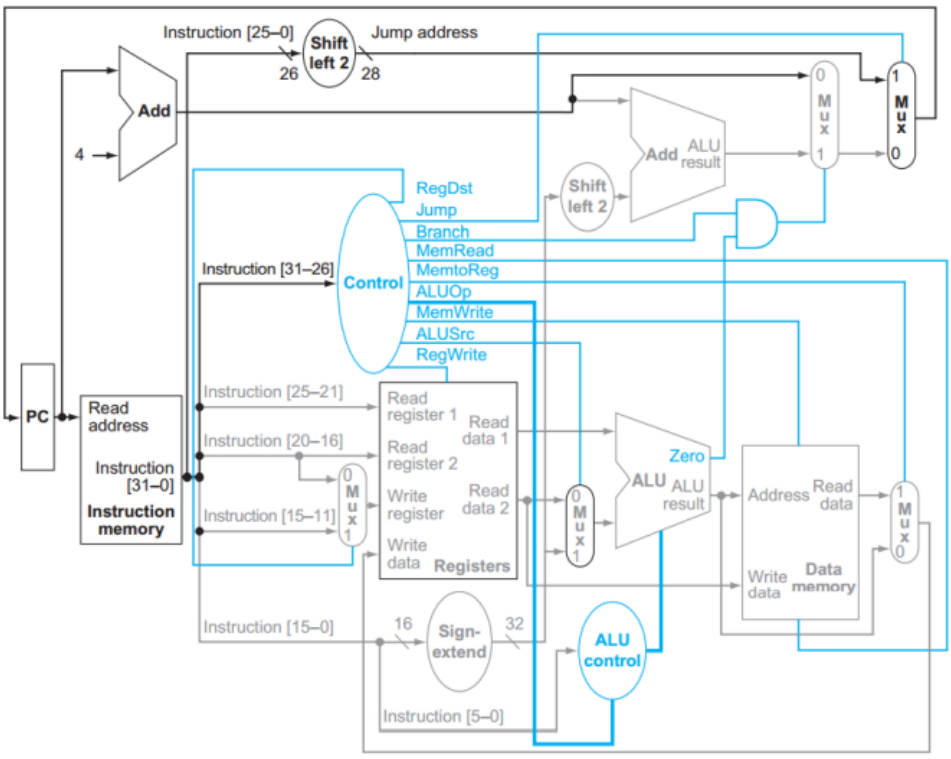


Figure 2: modified mips processor.

2 - Procedure

We start support for the instructions by modifying the following files: control.v, ALUControl.v, and ALU.v. We use the tables below to define the control signals for each corresponding instruction. Instruction [31:26] corresponds to the control signal input, and table 2 helps define the output control signals like RegDest, Jump, Branch, ALUOp, etc. Instruction [5:0] and ALUop gets concatenated to determine the ALU operation needed for the instruction. In addition to defining the signals we also need to implement new operations to the ALU, operations for instructions like multiplication, division, and shifting instructions.

Name	format	Instruction[31:26]	Instruction[5:0]	ALUop	alu_control
Add	R	000000	100000	10	0010
Addi	I	001000	-	00	0010
and	R	000000	100100	10	0000
andi	I	001100	-	11	0000
Beq	I	000100	-	01	0110
Lw	I	100011	-	00	0010
Nor	R	000000	100111	10	1100
Or	R	000000	100101	10	0001
Slt	R	000000	101010	10	0111
Sll	R	110000	000000	10	1000
Srl	R	110000	000010	10	1001
sra	R	110000	000011	10	1010
Sw	I	101011	-	00	0010
Sub	R	000000	100010	10	0110
xor	R	000000	100110	10	0100
Mult	R	000000	011000	10	0101
div	R	000000	011010	10	1011
jump	J	000010	-	-	-

Table 2: Instruction set.

Name	Format	op	rs	rt	rd	shamt	funct	Comments
Add	R	0	reg	reg	reg	0	32	add $s1, $s2, $s3
Addi	I	8	reg	reg	n.a.	n.a.	n.a.	addi $s1, $s2, 20
and	R	0	reg	reg	reg	0	36	and $s1, $s2, $s3
andi	I	12	reg	reg	n.a.	n.a.	n.a.	andi $s1, $s2, 20
Beq	I	4	reg	reg	n.a.	n.a.	n.a.	Beq $s1, $s0, L1
Lw	I	35	reg	reg	n.a.	n.a.	n.a.	lw $s1, 20($s2)
Nor	R	0	reg	reg	reg	0	39	nor $s1, $s2, $s3
Or	R	0	reg	reg	reg	0	37	or $s1, $s2, $s3
Slt	R	0	reg	reg	reg	n.a.	42	slt $s1, $s2, $s3
Sll	R	48	reg	reg	reg	Shift amount	0	sll $s1, $s2, 10
Srl	R	48	reg	reg	reg	Shift amount	2	srl $s1, $s2, 10
sra	R	48	reg	reg	reg	Shift amount	3	sra $s1, $s2, 10
Sw	I	43	reg	reg	n.a.	n.a.	n.a.	sw $s1, 20($s2)
Sub	R	0	reg	reg	reg	0	34	sub $s1, $s2, $s3
xor	R	0	reg	reg	reg	0	38	xor $s1, $s2, $s3
Mult	R	0	reg	reg	reg	0	24	mult $s1, $s2, $s3
div	R	0	reg	reg	reg	0	26	div $s1, $s2, $s3
jump	J	2	n.a.	n.a.	n.a.	n.a.	n.a.	J 2500

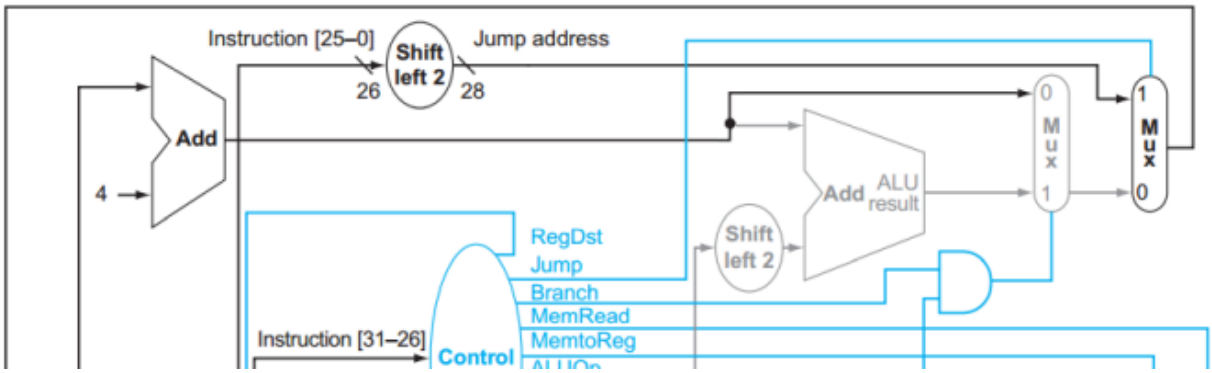
To implement the branch and jump instructions, we must modify the datapath.v file to be able to calculate the address of the beq instruction if it is taken and the address for jump. To do this we begin by adding two muxes that are wired together, one mux is for branch and the other is for jump. The branch mux will take in the address of the program counter plus 4 if it is not taken, otherwise it will take the branch target address if the control signal for branch and the alu results in zero. The jump mux is a 2:1 mux that has the output of the branch mux and the calculated target address. This mux chooses the input based on the jump control signal, if it is 1 it chooses the target address, and if it is 0 it chooses the output of the branch mux.

The jump address is calculated by the following:

It is just the lower 26 bits of the instruction shifted to the left by 2.

The branch target address is calculated by the following:

(PC + 4) + (SignExt[imm16 << 2])



3 - Simulation Results

The following are just a few instructions that are loaded into the instruction memory to be tested. The test bench writes to the console and indicates whether or not the instruction succeeded or failed.

```
34        ////////////////////////////////////////////////////// grading
35        // load to registers 1 to 10
36                                                  // instruction           alu result in hex      register content       mem content
37        rom[0] = 32'b10001100000000010000000000000000; // r1  = mem[0]          4 (add 0)            r1 = 00000005          -
38        rom[1] = 32'b10001100000000100000000000000100; // r2  = mem[1]          8 (add 1)            r2 = 0fdf6e91          -
39        rom[2] = 32'b10001100000000110000000000001000; // r3  = mem[2]          c (add 2)            r3 = 6a31439b          -
40        rom[3] = 32'b10001100000001000000000000001100; // r4  = mem[3]         10(add 3)             r4 = 56343ffd          -
41        rom[4] = 32'b10001100000001010000000000010000; // r5  = mem[4]         14(add 4)             r5 = 429eeddb          -
42        rom[5] = 32'b10001100000001100000000000010100; // r6  = mem[5]         18(add 5)             r6 = 90000000          -
43        rom[6] = 32'b10001100000001110000000000011000; // r7  = mem[6]         1c(add 6)             r7 = 9134fd75          -
44        rom[7] = 32'b10001100000010000000000000011100; // r8  = mem[7]         20(add 7)             r8 = bcd11247          -
45        rom[8] = 32'b10001100000010010000000000100000; // r9  = mem[8]         24(add 8)             r9 = b55bd831          -
46        rom[9] = 32'b10001100000010100000000000100100; // r10 = mem[9]         28(add 9)             r10= d18fa600          -
47
48        // two positive operands
49        rom[10] = 32'b00110000011010111111111101100011; // andi r11,r3,#ff63        6a314303          r11= 6a314303          -
50        rom[11] = 32'b00000000001000100110000000100111; // nor  r12,r1,r2           f020916a          r12= f020916a          -
51        rom[12] = 32'b00000000001000100110100000101010; // slt  r13,r1,r2           00000001          r13= 00000001          -
52        rom[13] = 32'b11000000010000000111000011000000; // sll  r14,r2,#3           7efb7488          r14= 7efb7488          -
53        rom[14] = 32'b11000000001000000111100101000010; // srl  r15,r1,#5           00000000          r15= 00000000          -
54        rom[15] = 32'b11000000110000001000001110000011; // sra  r16,r6,#6           fe400000          r16= fe400000          -
55        rom[16] = 32'b00000000010000111000100000100110; // xor  r17,r2,r3           65ee2d0a          r17= 65ee2d0a          -
56        rom[17] = 32'b00000000001000101001000000011000; // mult r17,r1,r2           4f5d28d5          r18= 4f5d28d5          -
57        rom[18] = 32'b00000000010000011001100000011010; // div  r19,r2,r1           032cafb6          r19= 032cafb6          -
58        // store the result in memory
59        rom[19] = 32'b10101100000010110000000000101100; // sw mem[r0+11] <= r11     2c(add 11)             -             mem[11]= 6a314303
60        rom[20] = 32'b10101100000011000000000000110000; // sw mem[r0+12] <= r12     30(add 12)             -             mem[12]= f020916a
61        rom[21] = 32'b10101100000011010000000000110100; // sw mem[r0+13] <= r13     34(add 13)             -             mem[13]= 00000001
62        rom[22] = 32'b10101100000011100000000000111000; // sw mem[r0+14] <= r14     38(add 14)             -             mem[14]= 7efb7488
63        rom[23] = 32'b10101100000011110000000000111100; // sw mem[r0+15] <= r15     3c(add 15)             -             mem[15]= 00000000
64        rom[24] = 32'b10101100000010000000000000001000; // sw mem[r0+16] <= r16     40(add 16)             -             mem[16]= fe400000
65        rom[25] = 32'b10101100000010001000000000001000; // sw mem[r0+17] <= r17     44(add 17)             -             mem[17]= 65ee2d0a
66        rom[26] = 32'b10101100000010010000000000001000; // sw mem[r0+18] <= r18     48(add 18)             -             mem[18]= 4f5d28d5
67        rom[27] = 32'b10101100000010011000000000001100; // sw mem[r0+19] <= r19     4c(add 19)             -             mem[19]= 032cafb6
68
69        // one positive and one negative operand
70        rom[28] = 32'b00110000111010110000111101100011; // andi r11,r7,#f63         00000d61          r11= 00000d61          -
71        rom[29] = 32'b00000000010001110110000000100111; // nor  r12,r2,r7           6000000a          r12= 6000000a          -
72        rom[30] = 32'b00000000010001110110100000101010; // slt  r13,r2,r7           00000001          r13= 00000001          -
73        rom[31] = 32'b11000000111000000111001101000000; // sll  r14,r2,#13          9faea000          r14= 9faea000          -
74        rom[32] = 32'b11000001000000000111100111000010; // srl  r15,r8,#7           0179a224          r15= 0179a224          -
75        rom[33] = 32'b11000001001000001000000010000011; // sra  r16,r9,#2           ed56f60c          r16= ed56f60c          -
76        rom[34] = 32'b00000000010001111000100000100110; // xor  r17,r2,r7           9eeb93e4          r17= 9eeb93e4          -
77        rom[35] = 32'b00000000010001111001000000011000; // mult r17,r2,r7           a7d6d545          r18= a7d6d545          -
78        rom[36] = 32'b00000000111000101001100000011010; // div  r19,r7,r2           00000009          r19= 00000009          -
```

```
// store the result in memory
rom[19] = 32'b10101100000010110000000000101100; // sw mem[r0+11] <= r11       2c(add 11)              -              mem[11]= 6a314303
rom[20] = 32'b10101100000011000000000000110000; // sw mem[r0+12] <= r12       30(add 12)              -              mem[12]= f020916a
rom[21] = 32'b10101100000011010000000000110100; // sw mem[r0+13] <= r13       34(add 13)              -              mem[13]= 00000001
rom[22] = 32'b10101100000011100000000000111000; // sw mem[r0+14] <= r14       38(add 14)              -              mem[14]= 7efb7488
rom[23] = 32'b10101100000011110000000000111100; // sw mem[r0+15] <= r15       3c(add 15)              -              mem[15]= 00000000
rom[24] = 32'b10101100000010000000000000001000; // sw mem[r0+16] <= r16       40(add 16)              -              mem[16]= fe400000
rom[25] = 32'b10101100000010001000000000001000; // sw mem[r0+17] <= r17       44(add 17)              -              mem[17]= 65ee2d0a
rom[26] = 32'b10101100000010010000000000001000; // sw mem[r0+18] <= r18       48(add 18)              -              mem[18]= 4f5d28d5
rom[27] = 32'b10101100000010011000000000001000; // sw mem[r0+19] <= r19       4c(add 19)              -              mem[19]= 032cafb6

// one positive and one negative operand
rom[28] = 32'b00110000111010110000111101100011; // andi r11,r7,#f63          00000d61          r11= 00000d61          -
rom[29] = 32'b00000000010001110110000000100111; // nor  r12,r2,r7            6000000a          r12= 6000000a          -
rom[30] = 32'b00000000010001110110100000101010; // slt  r13,r2,r7            00000001          r13= 00000001          -
rom[31] = 32'b11000000111000000111001101000000; // sll  r14,r2,#13           9faea000          r14= 9faea000          -
rom[32] = 32'b11000001000000000111100111000010; // srl  r15,r8,#7            0179a224          r15= 0179a224          -
rom[33] = 32'b11000001001000001000000010000011; // sra  r16,r9,#2            ed56f60c          r16= ed56f60c          -
rom[34] = 32'b00000000010001111000100000100110; // xor  r17,r2,r7            9eeb93e4          r17= 9eeb93e4          -
rom[35] = 32'b00000000010001111001000000011000; // mult r17,r2,r7            a7d6d545          r18= a7d6d545          -
rom[36] = 32'b00000000111000101001100000011010; // div  r19,r7,r2            00000009          r19= 00000009          -
// store the result in memory
rom[37] = 32'b10101100000010110000000001010000; // sw mem[r0+20] <= r11       50(add 20)              -              mem[20]= 00000d61
rom[38] = 32'b10101100000011000000000001010100; // sw mem[r0+21] <= r12       54(add 21)              -              mem[21]= 6000000a
rom[39] = 32'b10101100000011010000000001011000; // sw mem[r0+22] <= r13       58(add 22)              -              mem[22]= 00000001
rom[40] = 32'b10101100000011100000000001011100; // sw mem[r0+23] <= r14       5c(add 23)              -              mem[23]= 9faea000
rom[41] = 32'b10101100000011110000000001100000; // sw mem[r0+24] <= r15       60(add 24)              -              mem[24]= 0179a224
rom[42] = 32'b10101100000010000000000001100100; // sw mem[r0+25] <= r16       64(add 25)              -              mem[25]= ed56f60c
rom[43] = 32'b10101100000010001000000001101000; // sw mem[r0+26] <= r17       68(add 26)              -              mem[26]= 9eeb93e4
rom[44] = 32'b10101100000010010000000001101100; // sw mem[r0+27] <= r18       6c(add 27)              -              mem[27]= a7d6d545
rom[45] = 32'b10101100000010011000000001110000; // sw mem[r0+28] <= r19       70(add 28)              -              mem[28]= 00000009

// one positive and one negative operand
rom[46] = 32'b00110001010010111111000010010111; // andi r11,r10,#e127        d18fa000          r11= d18fa000          -
rom[47] = 32'b00000000011010000110000000100111; // nor  r12,r3,r8            010eac20          r12= 010eac20          -
rom[48] = 32'b00000001000000110110100000101010; // slt  r13,r8,r3            00000000          r13= 00000000          -
rom[49] = 32'b11000000011000000111010001000000; // sll  r14,r3,#17           87360000          r14= 87360000          -
rom[50] = 32'b11000001000000000111110100000010; // srl  r15,r8,#20           00000bcd          r15= 00000bcd          -
rom[51] = 32'b11000001000000001000000001000011; // sra  r16,r8,#1            de688923          r16= de688923          -
rom[52] = 32'b00000000011010001000100000100110; // xor  r17,r3,r8            d6e051dc          r17= d6e051dc          -
rom[53] = 32'b00000000011010001001000000011000; // mult r17,r3,r8            eff5a5fd          r18= eff5a5fd          -
rom[54] = 32'b00000001000000111001100000011010; // div  r19,r8,r3            00000001          r19= 00000001          -
```

After running the simulation the console yields the following results:

ANDI 1 success!

NOR 1 success!

SLT 1 success!

SLL 1 success!

SRL 1 success!

SRA 1 success!

XOR 1 success!

MULT 1 success!

DIV 1 success!

ANDI 2 success!

NOR 2 success!

SLT 2 success!

SLL 2 success!

SRL 2 success!

SRA 2 success!

XOR 2 success!

MULT 2 success!

DIV 2 success!

ANDI 3 success!

NOR 3 success!

SLT 3 success!

SLL 3 success!

SRL 3 success!

SRA 3 success!

XOR 3 success!

MULT 3 success!

DIV 3 success!

ANDI 4 success!

NOR 4 success!

SLT 4 success!

SLL 4 success!

SRL 4 success!

SRA 4 success!

XOR 4 success!

MULT 4 success!

DIV 4 success!

ANDI 5 success!

NOR 5 success!

SLT 5 success!

SLL 5 success!

SRL 5 success!

SRA 5 success!

XOR 5 success!

MULT 5 success!

DIV 5 success!

BEQ 1 success!

BEQ 2 success!

BEQ 3 success!

BEQ 4 success!

BEQ 5 success!

j 1 success!

j 2 success!

j 3 success!

j 4 success!

j 5 success!

points : 70

All instructions work like normal according to the procedure.