RISCV32M Instruction Set

Test Case:

Addi x1, x0, 10

Addi x2, x0, 20

Mul x3,x2,x1

div x4, x2, x1

mulh x5, x2, x1

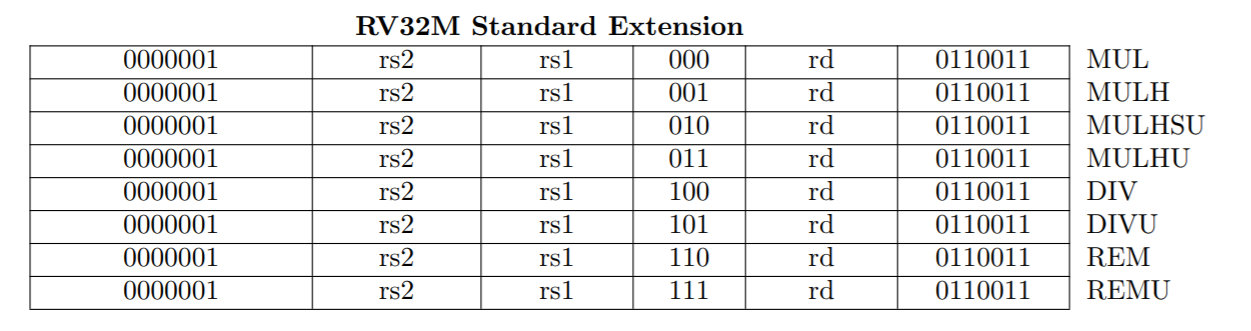
mulhu x6, x2, x1

mulhsu x7, x2, x1

divu x8, x2, x1

Rem x9, x2, x1

Remu x10, x2, x1



The following are pseudo code for what is needed to be changed in the processor.

The changes were only in three modules: ALU, control unit and ALU control unit. The logic of this instruction set was implemented in the ALU.

The wiring would occur in the RISCV Handler with adding bits to the pipeline.

module prv32\_ALU(

input wire [31:0] a, b,

input wire [4:0] shamt,

output reg [31:0] r,

output wire cf, zf, vf, sf,

input wire [3:0] alufn

);

wire [31:0] add, sub, op\_b;

wire cfa, cfs;

assign op\_b = (~b);

assign {cf, add} = alufn[0] ? (a + op\_b + 1'b1) : (a + b);

assign zf = (add == 0);

assign sf = add[31];

assign vf = (a[31] ^ (op\_b[31]) ^ add[31] ^ cf);

wire[31:0] sh;

shifter shifter0(.a(a), .shamt(shamt), .type(alufn[1:0]), .r(sh));

always @ \* begin

r = 0;

(\* parallel\_case \*)

case (alufn)

// arithmetic

4'b00\_00 : r = add;

4'b00\_01 : r = add;

4'b00\_11 : r = b;

// logic

4'b01\_00: r = a | b;

4'b01\_01: r = a & b;

4'b01\_11: r = a ^ b;

// shift

4'b10\_00: r=sh;

4'b10\_01: r=sh;

4'b10\_10: r=sh;

// slt & sltu

4'b11\_01: r = {31'b0,(sf != vf)};

4'b11\_11: r = {31'b0,(~cf)};

endcase

always @(\*) begin

if (a[31] == 1'b1) begin

a = -a;

end

if (b[31] == 1'b1) begin

b = -b;

end else begin

assign a = a;

assign b = b;

end

end

case(a[6:2] == 5’b01100)

4'b00\_00: r = a \* b; //mul

4'b00\_01: r = a & b; //mulh

4'b00\_10: r = a ^ b; //mulhsu

4'b00\_11: r = a \* b; //mulhu

4'b01\_00: r = a / b; //div

4'b01\_01: r = a / b; //divu

4'b01\_10: r = a % b; //rem

4'b01\_11: r = a % b; //remu

endcase

end

endmodule

-///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////-

`timescale 1ns/1ns

module control\_unit(input [4:0] instr, output reg Branch, MemRead, MemtoReg, MemWrite, ALUSrc, RegWrite, output reg [1:0] ALUOp, output reg JALFlag, output reg [1:0] Jal\_selection, output reg [1:0] mux\_spade\_selection, output reg r\_check);

always @(\*) begin

r\_check = 0;

Branch = (instr==5'b11000); //Branch, JALR, JAL

MemRead = (instr==5'b00000);

MemtoReg= (instr==5'b00000);

MemWrite= (instr==5'b01000);

ALUSrc= (instr== 5'b00000) || (instr==5'b01000) || (instr==5'b00100); //This supports the I-format instructions

RegWrite= (instr== 5'b00000)||(instr==5'b01100)||(instr==5'b00100) || (instr==5'b01101) || (instr==5'b00101) || (instr==5'b11011) || (instr==5'b11001);

JALFlag = (instr==5'b11001 || instr==5'b11011);

Jal\_selection = 2'b00;

if(instr[4:0] == 5'b11000) begin

Jal\_selection = 2'b01; //Branch

end

if(instr[4:0] == 5'b11001) begin

Jal\_selection = 2'b10; //JALR

end

if(instr[4:0] == 5'b11011) begin

Jal\_selection = 2'b11; //JAL

end

case(instr)

5'b01101:begin

ALUOp=2'b00; //LUI

mux\_spade\_selection = 2'b11;

end

5'b00101: begin //AUIPC

ALUOp= 2'b00;

mux\_spade\_selection = 2'b00;

end

5'b11011: begin

ALUOp = 2'b00; //JAL

mux\_spade\_selection = 2'b01;

end

5'b11001: begin

ALUOp = 2'b00;

mux\_spade\_selection = 2'b01; //JALR

end

5'b11000: ALUOp = 2'b01; //BEQ, BNE, BLT, BGE, BLTU, BGEU

5'b00000: begin

ALUOp = 2'b00; //LB, LH, LW, LBU, LHU

mux\_spade\_selection = 2'b10;

end

5'b01000: ALUOp = 2'b00; //SB, SH, SW

5'b00100: begin ALUOp = 2'b00; //ADDI, SLTI, SLTIU, XORI, ORI, ANDI, SLLI, SRLI, SRAI,

mux\_spade\_selection = 2'b10;

end

5'b01100: begin

ALUOp = 2'b10; //ADD, SUB, SLL, SLT, SLTU, XOR, SRL, SRA, OR, AND

mux\_spade\_selection = 2'b10;

r\_check = 1;

end

//FENCE

/\*

output reg [1:0] ALUOp, output reg JALFlag, output reg [1:0] Jal\_selection, output reg [1:0] mux\_spade\_selection, output reg r\_check);\*/

5'b00011: begin

Branch = 0; MemRead = 0; MemtoReg = 0; MemWrite = 0; ALUSrc = 0; RegWrite = 0;

ALUOp = 0; JALFlag = 0; Jal\_selection = 0; mux\_spade\_selection = 0; r\_check = 0;

end

//EBREAK

5'b11100: begin

Branch = 0; MemRead = 0; MemtoReg = 0; MemWrite = 0; ALUSrc = 0; RegWrite = 0;

ALUOp = 0; JALFlag = 0; Jal\_selection = 0; mux\_spade\_selection = 0; r\_check = 0;

end

endcase

end

endmodule

-///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////-

`timescale 1ns/1ns

module ALU\_control\_unit(input [1:0] ALUOp, input [2:0] instr14\_12, input instr30, input r\_check, output reg [3:0] ALU\_selection, output reg [1:0] state, output reg sign);

always @(\*) begin

sign = 0;

case (ALUOp)

2'b01: begin

ALU\_selection=4'b0001;

end

2'b00: begin

ALU\_selection=4'b0000; //add

if (instr14\_12 == 3'b010) begin

ALU\_selection = 4'b1101; //SLTI (set if less than immediate)

end

if (instr14\_12 == 3'b011) begin

ALU\_selection = 4'b1111; //SLTIU (set if less than immediate UNSIGNED)

end

if (instr14\_12 == 3'b001) begin

ALU\_selection = 4'b1001; //Shift left logic immediate (SLLI)

end

if (instr14\_12 == 3'b101 && instr30==0) begin

ALU\_selection = 4'b1010; //Shift right logic immediate (SRLI)

end

if (instr14\_12 == 3'b101 && instr30==1) begin

ALU\_selection = 4'b1000; //Shift right arithmetic immediate(SRAI)

end

//state

//Load Word - store word

if (instr14\_12 == 3'b010) begin

state = 2'b10;

end

//Load Half - store half

if (instr14\_12 == 3'b001 || instr14\_12 == 3'b101) begin

state = 2'b01;

end

//Load Byte - store byte

if (instr14\_12 == 3'b000 || instr14\_12 == 3'b100) begin

state = 2'b00;

end

//signed or unsigned

if (instr14\_12[2] == 1'b1) begin

sign = 1'b1;

end

end

2'b10: //This has a lot cases that should be expanded here with if statements

begin

if (instr14\_12==3'b010)begin

ALU\_selection = 4'b1101; //Set if less than (SLT)

end

if (instr14\_12==3'b011)begin

ALU\_selection = 4'b1111; //Set if less than UNSIGNED (SLTU)

end

if (instr14\_12== 3'b000) begin

if (r\_check == 1) begin

if (instr30==0)begin

ALU\_selection=4'b0000; //add

end

else begin

ALU\_selection=4'b0001; //subtract which is technically has the same selection line of add

end

end else if (r\_check == 0) begin

ALU\_selection = 4'b0000; //always add (ADDI) (Cancel SUBI)

end

end

if (instr14\_12== 3'b111) begin

if (instr30==0)begin

ALU\_selection=4'b0101; //and

end

end

if (instr14\_12== 3'b110) begin

if (instr30==0)begin

ALU\_selection=4'b0100; //or

end

end

if (instr14\_12== 3'b100) begin

if (instr30==0)begin

ALU\_selection=4'b0111; //xor

end

end

//Shiftat

if (instr14\_12== 3'b001) begin

ALU\_selection=4'b1001; //sll

end

if (instr14\_12== 3'b101 && instr30 == 0) begin

ALU\_selection=4'b1010; //srl

end

if (instr14\_12== 3'b101 && instr30 == 1) begin

ALU\_selection=4'b1000; //sra

end

end

endcase

end

endmodule