```
1   module alu (enable, Rs1, Rs2, Rd, instr, mulresult, exec2, stackout, mul1, mul2, Rout,
    jump, memaddr);
2
3   input enable; // active LOW, disables the ALU during load/store operations so that
    undefined behaviour does not occur
4   input signed [15:0] Rs1; // input source register 1
5   input signed [15:0] Rs2; // input source register 2
6   input signed [15:0] Rd; // input destination register
7   input [15:0] instr; // current instruction being executed
8   input signed [31:0] mulresult; // 32-bit result from multiplier
9   input exec2; // Input from state machine to indicate when to take in result from
    multiplication
10  input [15:0] stackout; // input from stack to be fed back to registers
11
12  output reg signed [15:0] mul1; // first number to be multiplied
13  output reg signed [15:0] mul2; // second number to be multiplied
14  output signed [15:0] Rout; // value to be saved to destination register
15  output jump; // tells decoder whether Jump condition is true
16  reg carry; // Internal carry register that is updated during appropriate opcodes
17  output reg [10:0] memaddr; // address to load data from / store data to RAMd
18
19  wire [5:0]opcode = instr[14:9]; // opcode of current instruction
20  reg signed [16:0] alusum; // extra bit to hold carry from operations other than Multiply
21  assign Rout = alusum [15:0];
22  assign jump = (alusum[16] && ((opcode[5:2] == 4'b0000) | (opcode[5:2] == 4'b0001) | (
    opcode[5:2] == 4'b0010)));
23  reg [15:0] mulextra;
24
25  //Jump Conditionals:
26  wire JC1, JC2, JC3, JC4, JC5, JC6, JC7, JC8;
27  assign JC1 = (Rs1 < Rs2);
28  assign JC2 = (Rs1 > Rs2);
29  assign JC3 = (Rs1 == Rs2);
30  assign JC4 = (Rs1 == 0);
31  assign JC5 = (Rs1 >= Rs2);
32  assign JC6 = (Rs1 <= Rs2);
33  assign JC7 = (Rs1 != Rs2);
34  assign JC8 = (Rs1 < 0);
35
36  always @(opcode, mulresult)
37     begin
38        if(!enable) begin
39           case (opcode)
40              6'b000000: alusum = {1'b1, Rd}; // JMP Unconditional Jump, first bit high to
    indicate jump and passes through Rd
41              6'b000001: alusum = {8'b10000000, instr[8:0]};  //JMA unconditional jump to
    address, first bit high to indicate jump, rest set to destination
42
43              6'b000100: alusum = {JC1, Rd}; // JC1 Conditional Jump A < B
44              6'b000101: alusum = {JC2, Rd}; // JC2 Conditional Jump A > B
45              6'b000110: alusum = {JC3, Rd}; // JC3 Conditional Jump A = B
46              6'b000111: alusum = {JC4, Rd}; // JC4 Conditional Jump A = 0
47
48              6'b001000: alusum = {JC5, Rd}; // JC5 Conditional Jump A >= B / A !< B
49              6'b001001: alusum = {JC6, Rd}; // JC6 Conditional Jump A <= B / A !> B
50              6'b001010: alusum = {JC7, Rd}; // JC7 Conditional Jump A != B
51              6'b001011: alusum = {JC8, Rd}; // JC8 Conditional Jump A < 0
52
53              6'b001100: alusum = {1'b0, Rs1 & Rs2}; // AND Bitwise AND
54              6'b001101: alusum = {1'b0, Rs1 | Rs2}; // OR Bitwise OR
55              6'b001110: alusum = {1'b0, Rs1 ^ Rs2}; // XOR Bitwise XOR
56              6'b001111: alusum = {1'b0, ~Rs1}; // NOT Bitwise NOT
57
58              6'b010000: alusum = {1'b0, ~Rs1 | ~Rs2}; // NND Bitwise NAND
59              6'b010001: alusum = {1'b0, ~Rs1 & ~Rs2}; // NOR Bitwise NOR
60              6'b010010: alusum = {1'b0, Rs1 ~^ Rs2}; // XNR Bitwise XNOR
61              6'b010011: alusum = {1'b0, Rs1}; // MOV Move (Rd = Rs1)
62
63              6'b010100: begin
64                    alusum = {1'b0, Rs1} + {1'b0, Rs2}; // ADD Add (Rd = Rs1 + Rs2)
65                    carry = alusum[16];
66                 end
67              6'b010101: begin
68                    alusum = {1'b0, Rs1} + {1'b0, Rs2} + carry; // ADC Add w/ Carry (Rd =
    Rs1 + Rs2 + C)
69                    carry = alusum[16];
70                 end
71              6'b010110: begin
72                    alusum = {1'b0, Rs1} + {17'b00000000000000001}; // ADO Add 1 (Rd = Rd
    + 1)
73                    carry = alusum[16];
```

```
74                  end
75              6'b010111: ;
76
77              6'b011000: begin
78                      alusum = {1'b0, Rs1} - {1'b0, Rs2}; // SUB Subtract (Rd = Rs1 - Rs2)
79                      carry = alusum[16];
80                  end
81              6'b011001: begin
82                      alusum = {1'b0, Rs1} - {1'b0, Rs2} + carry - {17'b00000000000000001};
    // SBC Subtract w/ Carry (Rd = Rs1 - Rs2 + C - 1)
83                      carry = alusum[16];
84                  end
85              6'b011010: begin
86                      alusum = {1'b0, Rs1} - {17'b00000000000000001}; // SBO Subtract 1 (Rd
    = Rd - 1)
87                      carry = alusum[16];
88                  end
89              6'b011011: ;
90
91              6'b011100: begin // MUL Multiply (Rd = Rs1 * Rs2)
92                      if(!exec2) begin
93                          if(Rs1[15]) begin
94                              mul1 = ~Rs1 + {16'h0001};
95                          end
96                          else begin
97                              mul1 = Rs1;
98                          end
99                          if(Rs2[15]) begin
100                             mul2 = ~Rs2 + {16'h0001};
101                         end
102                         else begin
103                             mul2 = Rs2;
104                         end
105                         alusum = 17'b00000000000000000;
106                         carry = (Rs1[15]^Rs2[15]) ? 1'b1 : 1'b0;
107                     end
108                     else begin
109                         {mulextra, alusum[15:0]} = (carry) ? ~mulresult + 32'h00000001 :
    mulresult;
110                     end
111                 end
112             6'b011101: begin // MLA Multiply and Add  (Rd = Rs2 + (Rd * Rs1))
113                     if(!exec2) begin
114                         if(Rd[15]) begin
115                             mul1 = ~Rd + {16'h0001};
116                         end
117                         else begin
118                             mul1 = Rd;
119                         end
120                         if(Rs1[15]) begin
121                             mul2 = ~Rs1 + {16'h0001};
122                         end
123                         else begin
124                             mul2 = Rs1;
125                         end
126                         alusum = 17'b00000000000000000;
127                         carry = (Rs1[15]^Rs2[15]) ? 1'b1 : 1'b0;
128                     end
129                     else begin
130                         {mulextra, alusum[15:0]} = (carry) ? ~mulresult + 32'h00000001 +
    {16'h0000, Rs2} : mulresult + {16'h0000, Rs2};
131                     end
132                 end
133             6'b011110: begin // MLS Multiply and Subtract (Rd = Rs2 - (Rd * Rs1)[15:0])
134                     if(!exec2) begin
135                         if(Rd[15]) begin
136                             mul1 = ~Rd + {16'h0001};
137                         end
138                         else begin
139                             mul1 = Rd;
140                         end
141                         if(Rs1[15]) begin
142                             mul2 = ~Rs1 + {16'h0001};
143                         end
144                         else begin
145                             mul2 = Rs1;
146                         end
147                         alusum = 17'b00000000000000000;
148                         carry = (Rs1[15]^Rs2[15]) ? 1'b1 : 1'b0;
149                     end
150                     else begin
```

```
151                         alusum = (carry) ? {1'b0, Rs2 - (~mulresult[15:0] + 16'h0001)} :
    {1'b0, Rs2 - mulresult[15:0]};
152                     end
153                 end
154             6'b011111: alusum = mulextra; // MRT Retrieve Multiply MSBs (Rd = MSBs)
155
156             6'b100000: alusum = {1'b0, Rs1 << Rs2}; // LSL Logical Shift Left (Rd = Rs1
    shifted left by value of Rs2)
157             6'b100001: alusum = {1'b0, Rs1 >> Rs2}; // LSR Logical Shift Right (Rd = Rs1
    shifted right by value of Rs2)
158             6'b100010: alusum = {Rs1[15], Rs1 >>> Rs2}; // ASR Arithmetic Shift Right
    (Rd = Rs1 shifted right by value of Rs2, maintaining sign bit)
159             6'b100011: ;
160
161             6'b100100: alusum = {1'b0, (Rs1 >> Rs2[3:0]) | (Rs1 << (16 - Rs2[3:0]))}; //
    ROR Shift Right Loop (Rd = Rs1 shifted right by Rs2, but Rs1[0] -> Rs1[15])
162 //          6'b100101: alusum = ({Rs1, carry} >> (Rs2 % 17)) | ({Rs1, carry} << (17 -
    (Rs2 % 17)));// RRC Shift Right Loop w/ Carry (Rd = Rs1 shifted right by Rs2, but Rs1[0]
    -> Carry & Carry -> Rs1[15])
163             6'b100110: alusum = {1'b1, Rd};  //CLL function call
164             6'b100111: begin  //RTN return to prev call
165                   if(exec2) begin
166                         alusum = {1'b0, stackout};
167                   end
168                end
169
170             6'b101000: alusum = {1'b0, Rs1}; // PSH Push value to stack (Stack = Rs1)
171             6'b101001: alusum = {1'b0, stackout}; // POP Pop value from stack (Rd = Stack)
172             6'b101010: begin // LDR Indirect Load (Rd = Mem[Rs1])
173                   if(!exec2) begin
174                         memaddr = Rs1[10:0];
175                   end
176                end
177             6'b101011: begin // STR Indirect Store (Mem[Rd] = Rs1)
178                   memaddr = Rd[10:0];
179                end
180
181             6'b111110: ; // NOP No Operation (Do Nothing for a cycle)
182             6'b111111: alusum = {1'b0, 16'h0000}; // STP Stop (Program Ends)
183
184             default: ; // During Load & Store as well as undefined opcodes
185          endcase;
186          end
187       else begin
188             alusum = {1'b0, 16'h0000}; // Bring output low during Load/Store so it does
    not interfere
189          end
190    end
191
192 endmodule
```