# CA Lab: Lab 11

student: Dimitri Tabatadze

June 17, 2023

## Task Description

Implement an instruction decoder.

## Solution

The module:

```
module InstructionDecoder (
    input [31:0] Instruction,
    // ALU
    output [3:0] Af,
    output I,
    output ALU_MUX_SEL,
    // GPR
    output [4:0] Cad,
    output GP_WE,
    output [1:0] GP_MUX_SEL, // ALU, MEMORY, SHIFTER, PC
    // BCE
    output [3:0] Bf,
    // MEMORY
    output DM_WE,
    // SHIFTER
    output reg [2:0] Shift_type,
    // PC
    output [1:0] PC_MUX_Select
);

    wire [5:0] opc, fun;
    wire [4:0] rs, rt, rd, sa;
    wire [25:0] iindex;
    wire [25:0] imm;
    wire j, alu;

    assign opc = Instruction[31:26];
    assign fun = Instruction[5:0];
    assign rs = Instruction[25:21];
    assign rt = Instruction[20:16];
    assign rd = Instruction[15:11];
    assign sa = Instruction[10:6];
    assign imm = Instruction[15:0];
```

```
    assign iindex = Instruction[25:0];

    assign I = opc != 0;
    assign j = opc == 6'b000010;
    assign ALU_MUX_SEL = ~I;
    assign Cad = rd;

    assign alu = (!I && fun[5:4] == 2'b10) || (I && opc[5:3] == 3'b001)
    ;

    assign Af = alu ? ((!I) ? fun[3:0] : {Instruction[27], opc[5:2]}) :
     4'b0000;
    assign GP_WE = alu || opc[5:3] == 3'b100 || opc == 6'b000011 || opc
    [5:3] == 3'b100 || ((!I) && fun == 6'b000010) || ((!I) && fun == 6'
    b001100);
    assign DM_WE = opc == 6'b101011;
    assign Bf = {opc[2:0], Instruction[16]};
    assign Cad = (opc == 6'b000011) ? 5'b11111 : ((!I) ? rd : rt);
    assign GP_MUX_SEL = {!alu && opc == 6'b100011, !alu && (!I) && fun
    == 6'b000010};

    assign PC_MUX_Select = {!(!I && (fun == 6'b001000 || fun == 6'
    b001001)) || !(I && !j && opc[5:3] == 3'b000), !(I && !j && opc
    [5:3] == 3'b000) && !j};

    always @ (Instruction) begin
        if(!I) begin
            Shift_type <= 3'b000;
            case (fun)
                6'b000: Shift_type <= 1;
                6'b010: Shift_type <= 2;
                6'b011: Shift_type <= 3;
                6'b100: Shift_type <= 4;
                6'b110: Shift_type <= 5;
                6'b111: Shift_type <= 6;
            endcase
        end
    end

endmodule
```

The testbench:

```
module id_tb();
    reg [31:0] instruction;
    wire [3:0] Af;
    wire I;
    wire ALU_MUX_SEL;
    wire [4:0] Cad;
    wire GP_WE;
    wire [1:0] GP_MUX_SEL;
    wire [3:0] Bf;
    wire DM_WE;
    wire [2:0] Shift_type;
    wire [1:0] PC_MUX_Select;


    InstructionDecoder UUT (
```

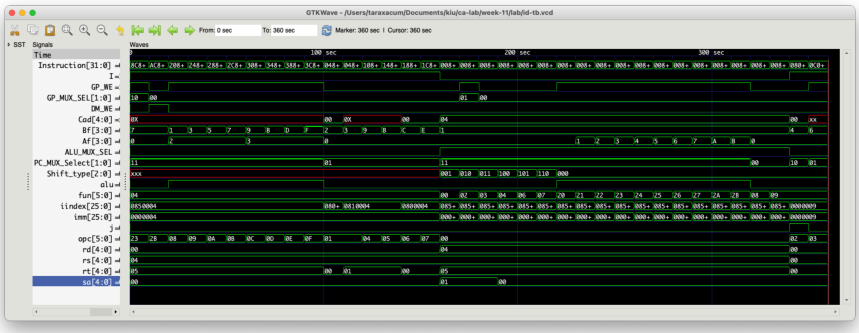
```
        .Instruction(instruction),
        .Af(Af),
        .I(I),
        .ALU_MUX_SEL(ALU_MUX_SEL),
        .Cad(Cad),
        .GP_WE(GP_WE),
        .GP_MUX_SEL(GP_MUX_SEL),
        .Bf(Bf),
        .DM_WE(DM_WE),
        .Shift_type(Shift_type),
        .PC_MUX_Select(PC_MUX_Select)
    );

    initial begin
    $dumpfile("id-tb.vcd");
    $dumpvars(0, id_tb);
        ////////////////////// TEST ITYPE INSTRUCTIONS
        instruction = 32'b10001100100001010000000000000100; // LW
        #10;
        instruction = 32'b10101100100001010000000000000100; // SW
        #10;
        instruction = 32'b00100000100001010000000000000100; // ADDI
        #10;
        instruction = 32'b00100100100001010000000000000100; // ADDIU
        #10;
        instruction = 32'b00101000100001010000000000000100; // SUBI
        #10;
        instruction = 32'b00101100100001010000000000000100; // SUBIU
        #10;
        instruction = 32'b00110000100001010000000000000100; // ANDI
        #10;
        instruction = 32'b00110100100001010000000000000100; // ORI
        #10;
        instruction = 32'b00111000100001010000000000000100; // XORI
        #10;
        instruction = 32'b00111100100001010000000000000100; // LUI
        #10;
        //////BRANCH
        instruction = 32'b00000100100000000000000000000100; // bltz
        #10;
        instruction = 32'b00000100100000001000000000000100; // bGEz
        #10;
        instruction = 32'b00010000100000001000000000000100; // beq
        #10;
        instruction = 32'b00010100100000001000000000000100; // bne
        #10;
        instruction = 32'b00011000100000000000000000000100; // blez
        #10;
        instruction = 32'b00011100100000000000000000000100; // bgtz
        #10;
        ////////////////// TEST RTYPE
        instruction = 32'b00000000100001010010000001000000; // SLL
        #10;
        instruction = 32'b00000000100001010010000001000010; // SRL
        #10;
        instruction = 32'b00000000100001010010000001000011; // SRA
        #10;
```

```
        instruction = 32'b00000000100001010010000000000100; // SLLV
        #10;
        instruction = 32'b00000000100001010010000000000110; // SRLV
        #10;
        instruction = 32'b00000000100001010010000000000111; // SRAV
        #10;
        instruction = 32'b00000000100001010010000000100000; // ADD
        #10;
        instruction = 32'b00000000100001010010000000100001; // ADDU
        #10;
        instruction = 32'b00000000100001010010000000100010; // SUB
        #10;
        instruction = 32'b00000000100001010010000000100011; // SUBU
        #10;
        instruction = 32'b00000000100001010010000000100100; // AND
        #10;
        instruction = 32'b00000000100001010010000000100101; // OR
        #10;
        instruction = 32'b00000000100001010010000000100110; // XOR
        #10;
        instruction= 32'b00000000100001010010000000100111; // NOR
        #10;
        instruction = 32'b00000000100001010010000000101010; // SLT
        #10;
        instruction = 32'b00000000100001010010000000101011; // SLTU
        #10;
        instruction = 32'b00000000100001010010000000001000; // JR
        #10;
        instruction = 32'b00000000100001010010000000001001; // JALR
        #10;
        /////////////// JTYPE
        instruction = 32'b00001000000000000000000000001001; // J
        #10;
        instruction = 32'b00001100000000000000000000001001; // JAL
        #10;
    $finish;
    end
endmodule
```

## Simulation and Testing

The timing diagrams:




# Reference

- me