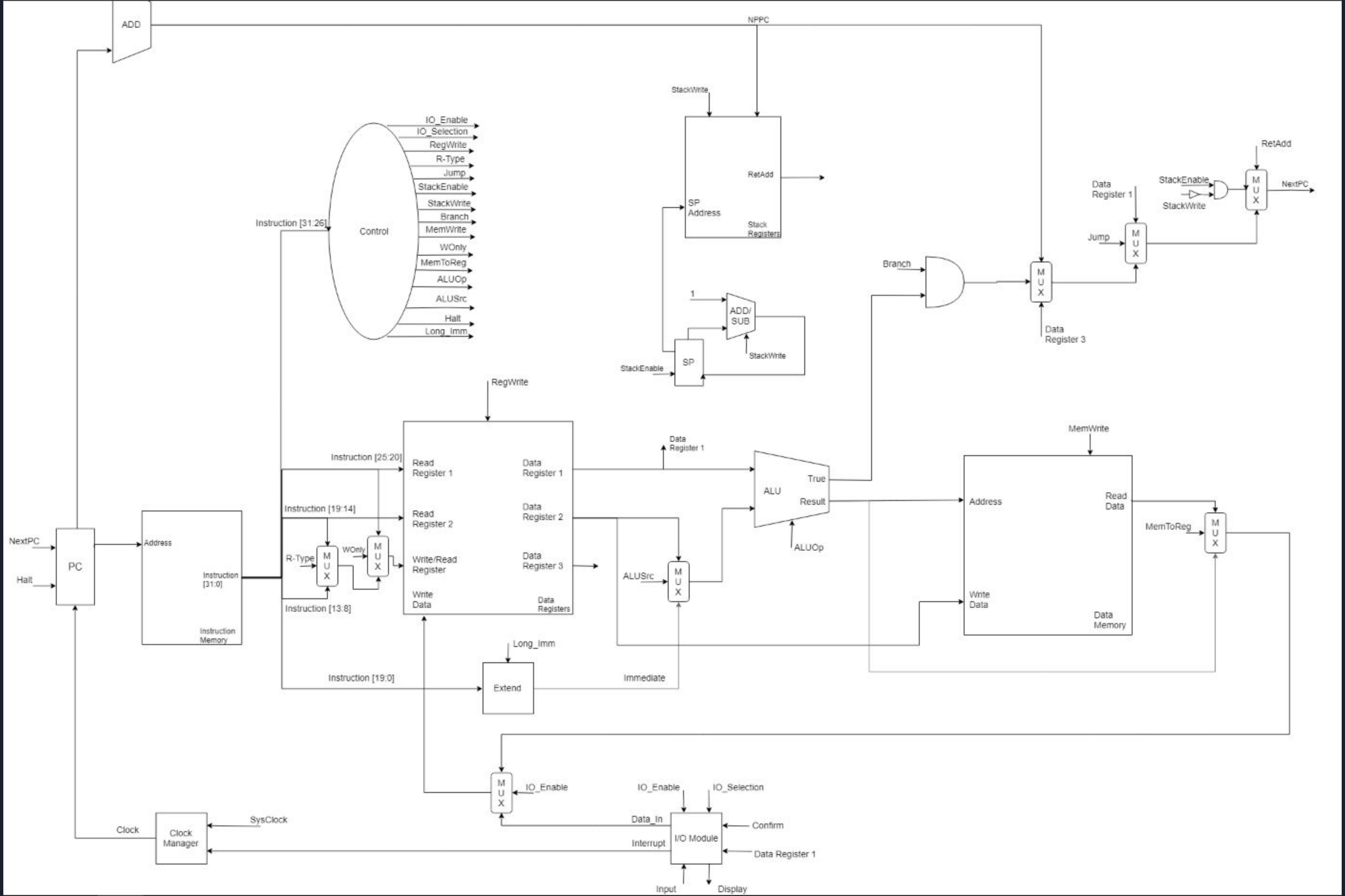
# Implementação da Plataforma “Reaper”

Uma arquitetura baseada em MIPS e RISC-V

## Modificações

# Tipos de Instruções

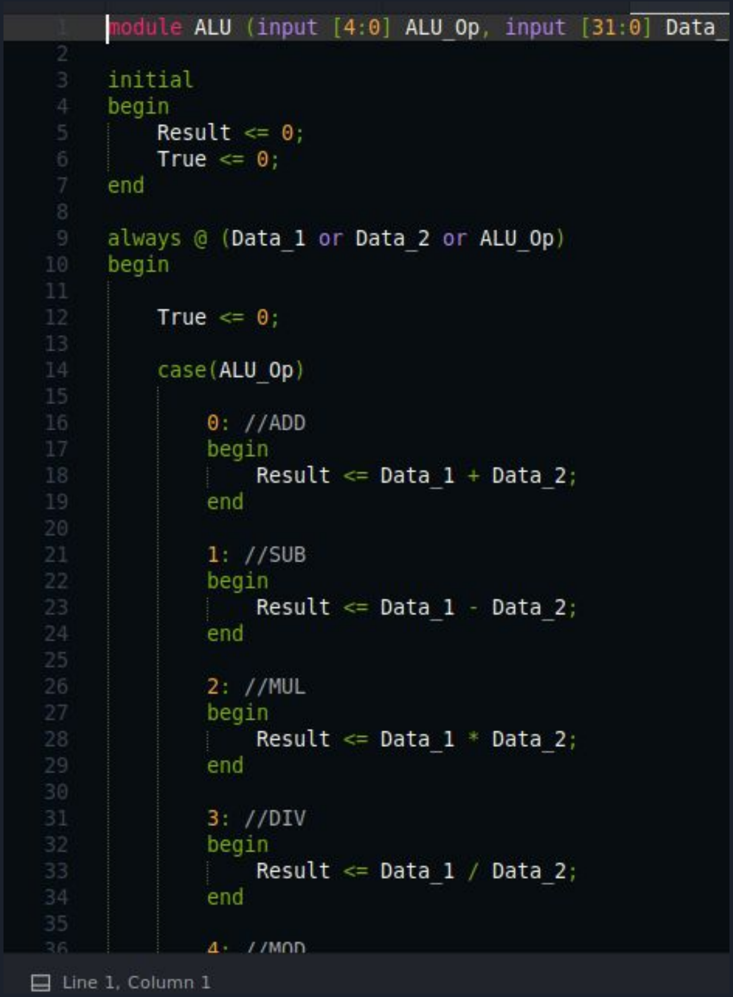
| Instruction Type | | Bit Mapping | | | | |
|---|---|---|---|---|---|---|
| I | Decoder [31:30] | OpCode [29:26] | $s (source) [25:20] | $t [19:14] | $imm [13:0] | |
| J | Decoder [31:30] | OpCode [29:26] | $s (source) [25:20] | $imm / -0- [19:0] | | |
| R | Decoder [31:30] | OpCode [29:26] | $s (source) [25:20] | $t [19:14] | $d [13:8] | -0- [7:0] |


ADD
NPPC
StackWrite
RetAdd
SP Address
Stack Registers
1
ADD/ SUB
StackWrite
StackEnable
SP
IO_Enable
IO_Selection
RegWrite
R-Type
Jump
StackEnable
StackWrite
Branch
MemWrite
WOnly
MemToReg
ALUOp
ALUSrc
Halt
Long_Imm
Instruction [31:26]
Control
RetAdd
Data Register 1
StackEnable
StackWrite
NextPC
Jump
Branch
Data Register 3
RegWrite
Data Register 1
Instruction [25:20]
Instruction [19:14]
Read Register 1
Read Register 2
Write/Read Register
Write Data
Data Register 1
Data Register 2
Data Register 3
Data Registers
R-Type
WOnly
Instruction [13:8]
NextPC
Halt
PC
Address
Instruction [31:0]
Instruction Memory
ALU
True
Result
ALUOp
ALUSrc
MemWrite
Address
Read Data
Write Data
Data Memory
MemToReg
Long_Imm
Instruction [19:0]
Extend
Immediate
IO_Enable
IO_Enable
IO_Selection
Data_In
Interrupt
I/O Module
Confirm
Data Register 1
Input
Display
Clock
Clock Manager
SysClock
M U X

## Desenvolvimento

# ULA

| AluOp | Op |
|---|---|
| 0 | ADD |
| 1 | SUB |
| 2 | MUL |
| 3 | DIV |
| 4 | MOD |
| 5 | AND |
| 6 | OR |
| 7 | XOR |
| 8 | NOT |
| 9 | SHL |
| 10 | SHR |
| 11 | EQ |
| 12 | NEQ |
| 13 | GEQ |
| 14 | GT |
| 15 | LEQ |
| 16 | LT |
| 17 | NOP |
| 18 | IMM |

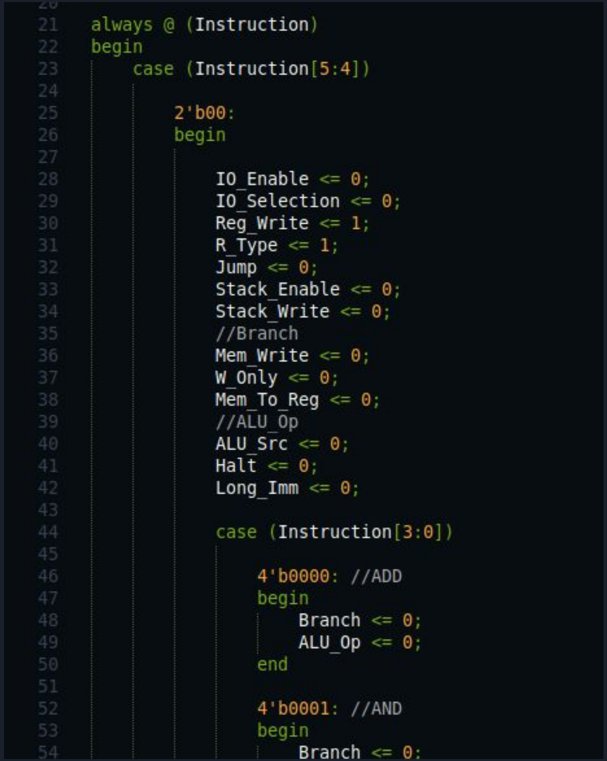


```
module ALU (input [4:0] ALU_Op, input [31:0] Data_

initial
begin
    Result <= 0;
    True <= 0;
end

always @ (Data_1 or Data_2 or ALU_Op)
begin

    True <= 0;

    case(ALU_Op)

        0: //ADD
        begin
            Result <= Data_1 + Data_2;
        end

        1: //SUB
        begin
            Result <= Data_1 - Data_2;
        end

        2: //MUL
        begin
            Result <= Data_1 * Data_2;
        end

        3: //DIV
        begin
            Result <= Data_1 / Data_2;
        end

        4: //MOD
```

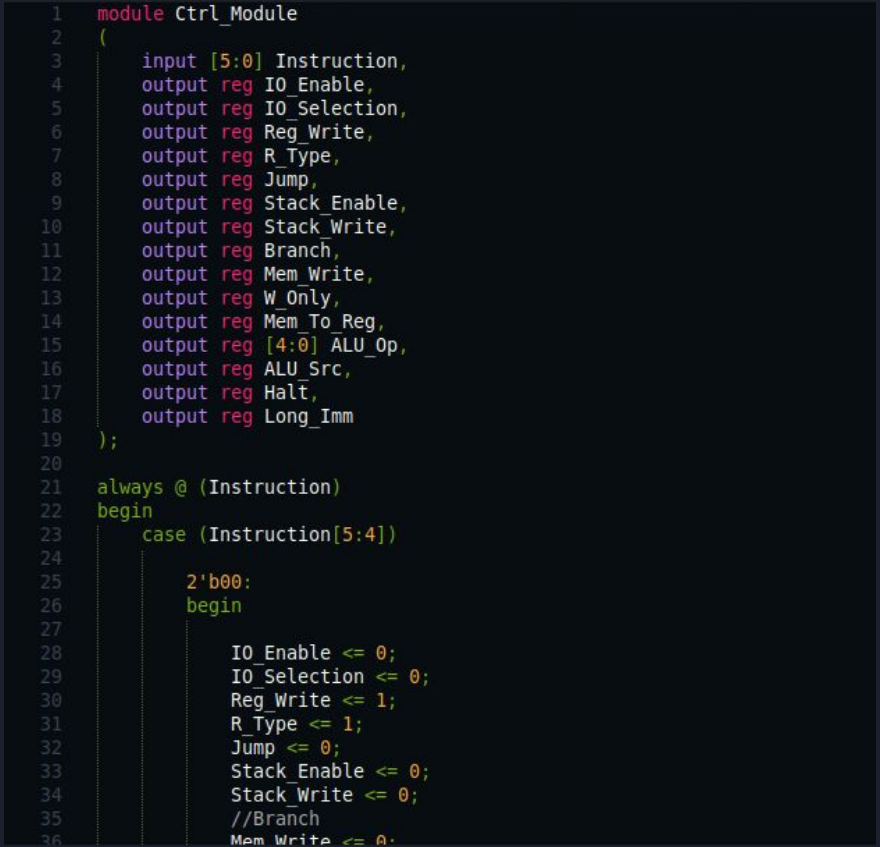
# Controle

| Instruction | | | | IO_ Enable | IO_ Selection | Reg_ Write | R_ Type | Jump | Stack_ Enable | Stack_ Write | Branch | Mem_ Write | W_ Only | Mem_ To_Reg | ALU_ Op | ALU_ Src | Halt | Long_ Imm |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADD $s $t $d | 00 | 0000 | R | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| AND $s $t $d | 00 | 0001 | R | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 0 |
| BEQ $s $t $d | 00 | 0010 | R | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 11 | 0 | 0 | 0 |
| IN $s | 10 | 0000 | J | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 17 | 1 | 0 | 1 |
| JAL $s | 10 | 0001 | J | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 17 | 1 | 0 | 1 |
| JR $s | 10 | 0010 | J | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 17 | 1 | 0 | 1 |
| STGTI $s $t $imm | 11 | 1010 | I | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 14 | 1 | 0 | 0 |
| STLTI $s $t $imm | 11 | 1011 | I | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 16 | 1 | 0 | 0 |
| STNEI $s $t $imm | 11 | 1100 | I | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 12 | 1 | 0 | 0 |

# Controle

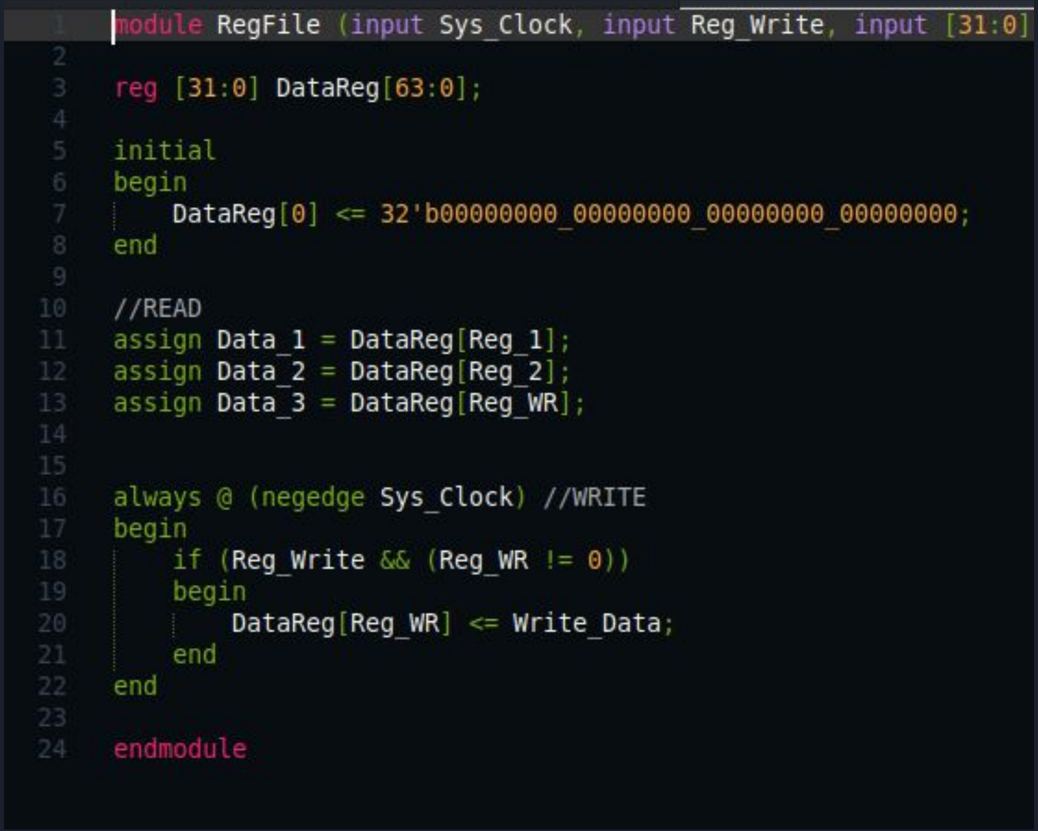


```
always @ (Instruction)
begin
    case (Instruction[5:4])

        2'b00:
        begin

            IO_Enable <= 0;
            IO_Selection <= 0;
            Reg_Write <= 1;
            R_Type <= 1;
            Jump <= 0;
            Stack_Enable <= 0;
            Stack_Write <= 0;
            //Branch
            Mem_Write <= 0;
            W_Only <= 0;
            Mem_To_Reg <= 0;
            //ALU_Op
            ALU_Src <= 0;
            Halt <= 0;
            Long_Imm <= 0;

            case (Instruction[3:0])

                4'b0000: //ADD
                begin
                    Branch <= 0;
                    ALU_Op <= 0;
                end

                4'b0001: //AND
                begin
                    Branch <= 0;
```



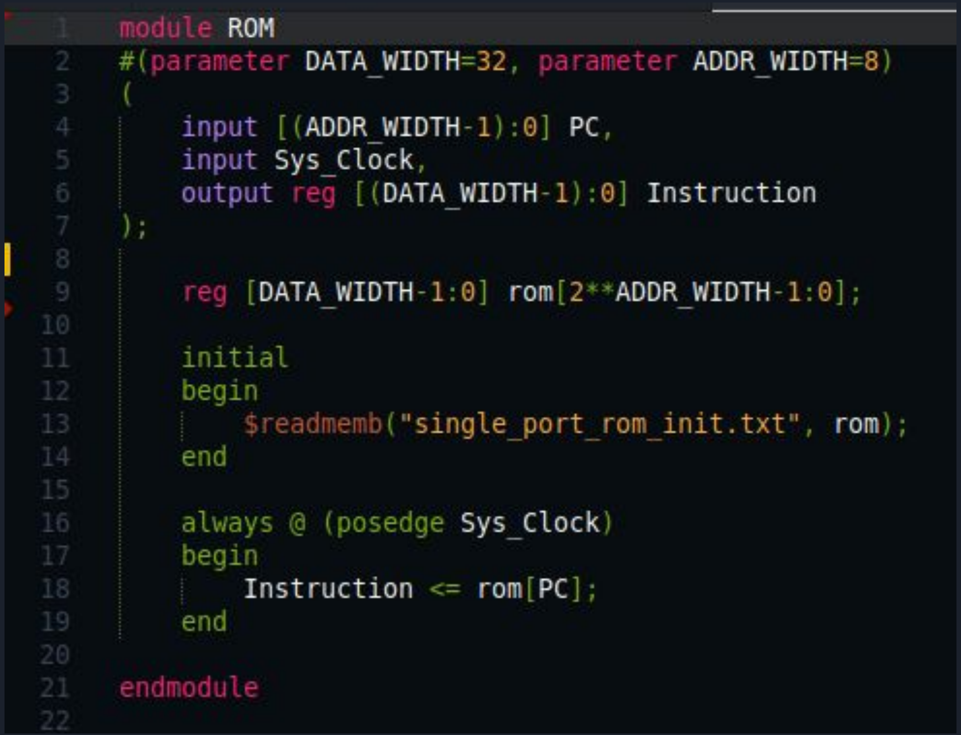
```
module Ctrl_Module
(
    input [5:0] Instruction,
    output reg IO_Enable,
    output reg IO_Selection,
    output reg Reg_Write,
    output reg R_Type,
    output reg Jump,
    output reg Stack_Enable,
    output reg Stack_Write,
    output reg Branch,
    output reg Mem_Write,
    output reg W_Only,
    output reg Mem_To_Reg,
    output reg [4:0] ALU_Op,
    output reg ALU_Src,
    output reg Halt,
    output reg Long_Imm
);

always @ (Instruction)
begin
    case (Instruction[5:4])

        2'b00:
        begin

            IO_Enable <= 0;
            IO_Selection <= 0;
            Reg_Write <= 1;
            R_Type <= 1;
            Jump <= 0;
            Stack_Enable <= 0;
            Stack_Write <= 0;
            //Branch
            Mem_Write <= 0;
```

## Banco de Registradores

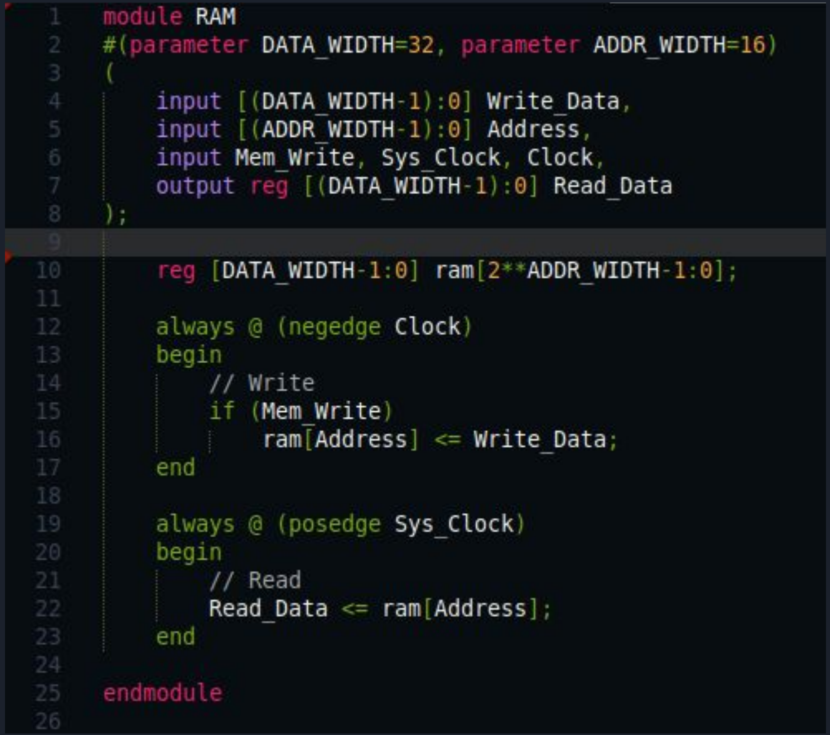


```
module RegFile (input Sys_Clock, input Reg_Write, input [31:0]

reg [31:0] DataReg[63:0];

initial
begin
    DataReg[0] <= 32'b00000000_00000000_00000000_00000000;
end

//READ
assign Data_1 = DataReg[Reg_1];
assign Data_2 = DataReg[Reg_2];
assign Data_3 = DataReg[Reg_WR];


always @ (negedge Sys_Clock) //WRITE
begin
    if (Reg_Write && (Reg_WR != 0))
    begin
        DataReg[Reg_WR] <= Write_Data;
    end
end

endmodule
```

# Memórias ROM e RAM

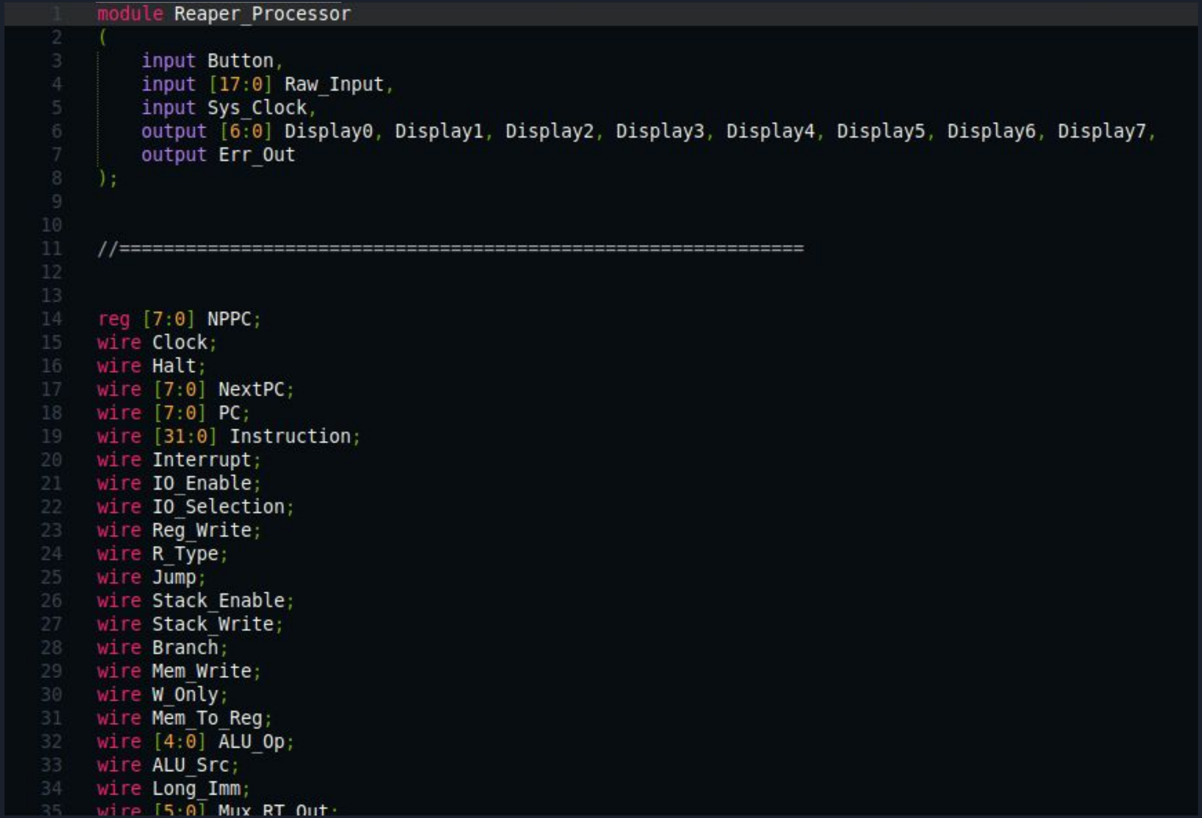


```verilog
module ROM
#(parameter DATA_WIDTH=32, parameter ADDR_WIDTH=8)
(
    input [(ADDR_WIDTH-1):0] PC,
    input Sys_Clock,
    output reg [(DATA_WIDTH-1):0] Instruction
);

    reg [DATA_WIDTH-1:0] rom[2**ADDR_WIDTH-1:0];

    initial
    begin
        $readmemb("single_port_rom_init.txt", rom);
    end

    always @ (posedge Sys_Clock)
    begin
        Instruction <= rom[PC];
    end

endmodule
```



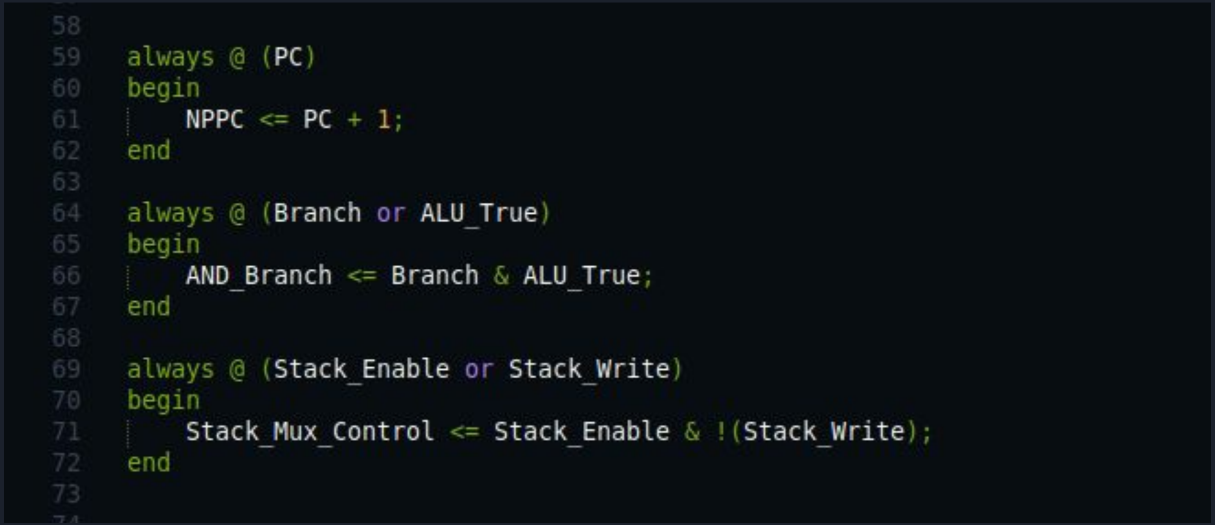
```verilog
module RAM
#(parameter DATA_WIDTH=32, parameter ADDR_WIDTH=16)
(
    input [(DATA_WIDTH-1):0] Write_Data,
    input [(ADDR_WIDTH-1):0] Address,
    input Mem_Write, Sys_Clock, Clock,
    output reg [(DATA_WIDTH-1):0] Read_Data
);

    reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];

    always @ (negedge Clock)
    begin
        // Write
        if (Mem_Write)
            ram[Address] <= Write_Data;
    end

    always @ (posedge Sys_Clock)
    begin
        // Read
        Read_Data <= ram[Address];
    end

endmodule
```

# Interligação



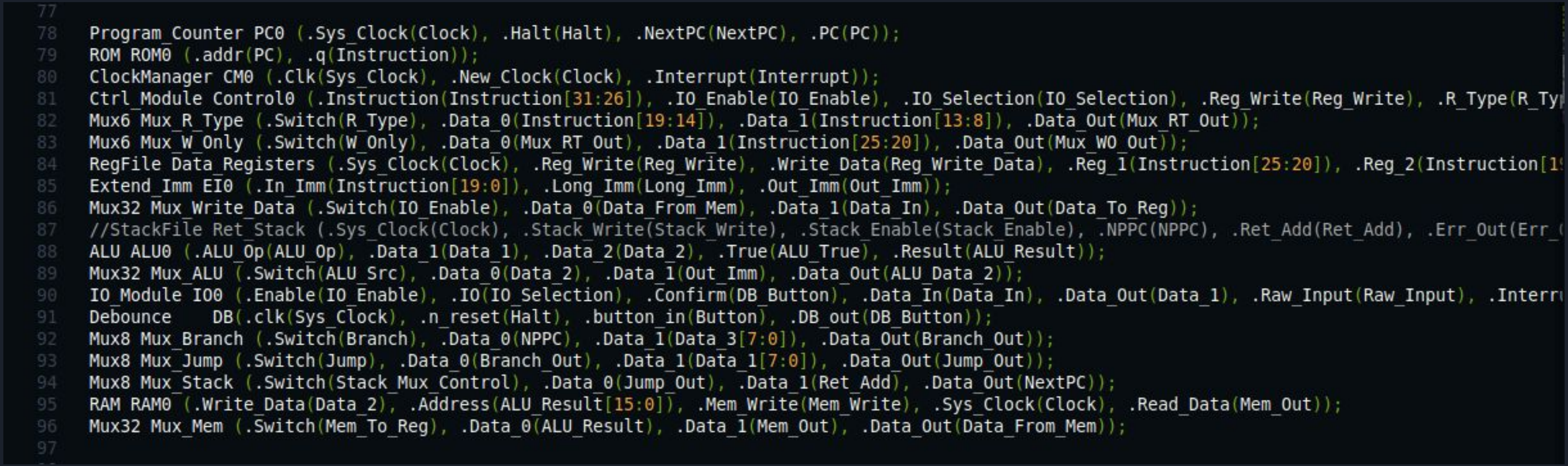
```
module Reaper_Processor
(
    input Button,
    input [17:0] Raw_Input,
    input Sys_Clock,
    output [6:0] Display0, Display1, Display2, Display3, Display4, Display5, Display6, Display7,
    output Err_Out
);


//==============================================================


reg [7:0] NPPC;
wire Clock;
wire Halt;
wire [7:0] NextPC;
wire [7:0] PC;
wire [31:0] Instruction;
wire Interrupt;
wire IO_Enable;
wire IO_Selection;
wire Reg_Write;
wire R_Type;
wire Jump;
wire Stack_Enable;
wire Stack_Write;
wire Branch;
wire Mem_Write;
wire W_Only;
wire Mem_To_Reg;
wire [4:0] ALU_Op;
wire ALU_Src;
wire Long_Imm;
wire [5:0] Mux_RT_Out;
```

## Interligação



```
always @ (PC)
begin
    NPPC <= PC + 1;
end

always @ (Branch or ALU_True)
begin
    AND_Branch <= Branch & ALU_True;
end

always @ (Stack_Enable or Stack_Write)
begin
    Stack_Mux_Control <= Stack_Enable & !(Stack_Write);
end
```

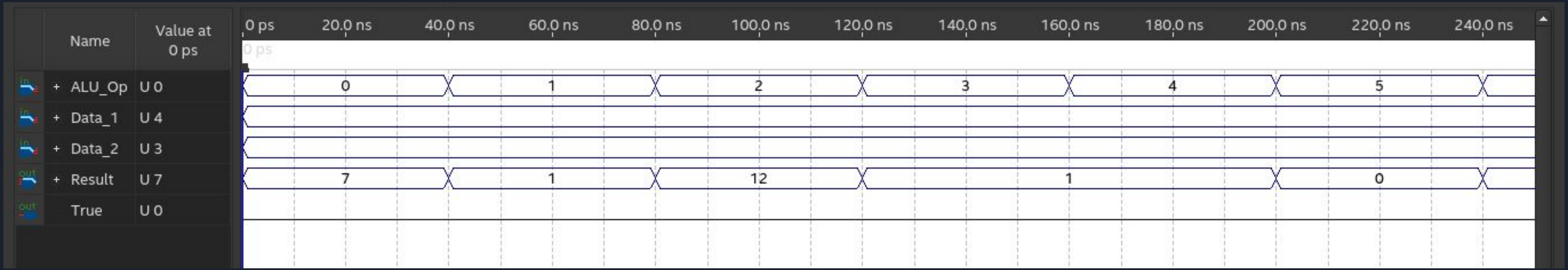
## Interligação



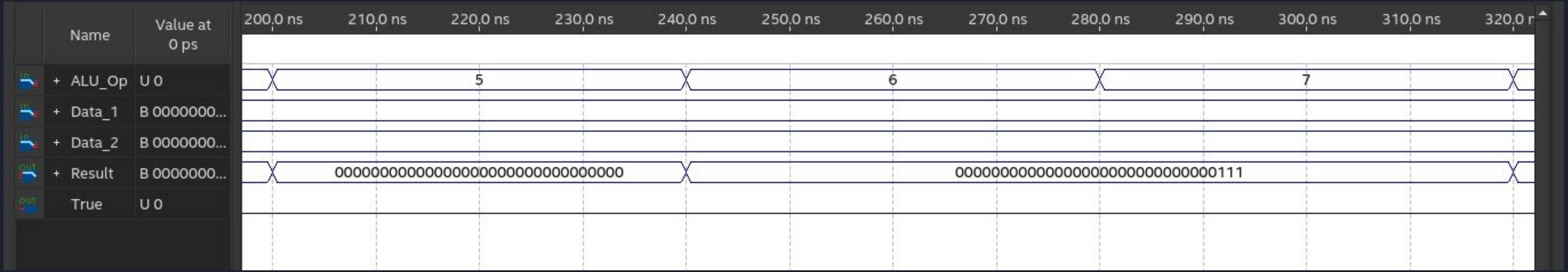
```
Program_Counter PC0 (.Sys_Clock(Clock), .Halt(Halt), .NextPC(NextPC), .PC(PC));
ROM ROM0 (.addr(PC), .q(Instruction));
ClockManager CM0 (.Clk(Sys_Clock), .New_Clock(Clock), .Interrupt(Interrupt));
Ctrl_Module Control0 (.Instruction(Instruction[31:26]), .IO_Enable(IO_Enable), .IO_Selection(IO_Selection), .Reg_Write(Reg_Write), .R_Ty
Mux6 Mux_R_Type (.Switch(R_Type), .Data_0(Instruction[19:14]), .Data_1(Instruction[13:8]), .Data_Out(Mux_RT_Out));
Mux6 Mux_W_Only (.Switch(W_Only), .Data_0(Mux_RT_Out), .Data_1(Instruction[25:20]), .Data_Out(Mux_WO_Out));
RegFile Data_Registers (.Sys_Clock(Clock), .Reg_Write(Reg_Write), .Write_Data(Reg_Write_Data), .Reg_1(Instruction[25:20]), .Reg_2(Instruction[1
Extend_Imm EI0 (.In_Imm(Instruction[19:0]), .Long_Imm(Long_Imm), .Out_Imm(Out_Imm));
Mux32 Mux_Write_Data (.Switch(IO_Enable), .Data_0(Data_From_Mem), .Data_1(Data_In), .Data_Out(Data_To_Reg));
//StackFile Ret_Stack (.Sys_Clock(Clock), .Stack_Write(Stack_Write), .Stack_Enable(Stack_Enable), .NPPC(NPPC), .Ret_Add(Ret_Add), .Err_Out(Err_
ALU ALU0 (.ALU_Op(ALU_Op), .Data_1(Data_1), .Data_2(Data_2), .True(ALU_True), .Result(ALU_Result));
Mux32 Mux_ALU (.Switch(ALU_Src), .Data_0(Data_2), .Data_1(Out_Imm), .Data_Out(ALU_Data_2));
IO_Module IO0 (.Enable(IO_Enable), .IO(IO_Selection), .Confirm(DB_Button), .Data_In(Data_In), .Data_Out(Data_1), .Raw_Input(Raw_Input), .Interr
Debounce    DB(.clk(Sys_Clock), .n_reset(Halt), .button_in(Button), .DB_out(DB_Button));
Mux8 Mux_Branch (.Switch(Branch), .Data_0(NPPC), .Data_1(Data_3[7:0]), .Data_Out(Branch_Out));
Mux8 Mux_Jump (.Switch(Jump), .Data_0(Branch_Out), .Data_1(Data_1[7:0]), .Data_Out(Jump_Out));
Mux8 Mux_Stack (.Switch(Stack_Mux_Control), .Data_0(Jump_Out), .Data_1(Ret_Add), .Data_Out(NextPC));
RAM RAM0 (.Write_Data(Data_2), .Address(ALU_Result[15:0]), .Mem_Write(Mem_Write), .Sys_Clock(Clock), .Read_Data(Mem_Out));
Mux32 Mux_Mem (.Switch(Mem_To_Reg), .Data_0(ALU_Result), .Data_1(Mem_Out), .Data_Out(Data_From_Mem));
```

## Principais Simulações

# ULA - Valores: 4 (0100) e 3(0011) AND, OR, XOR

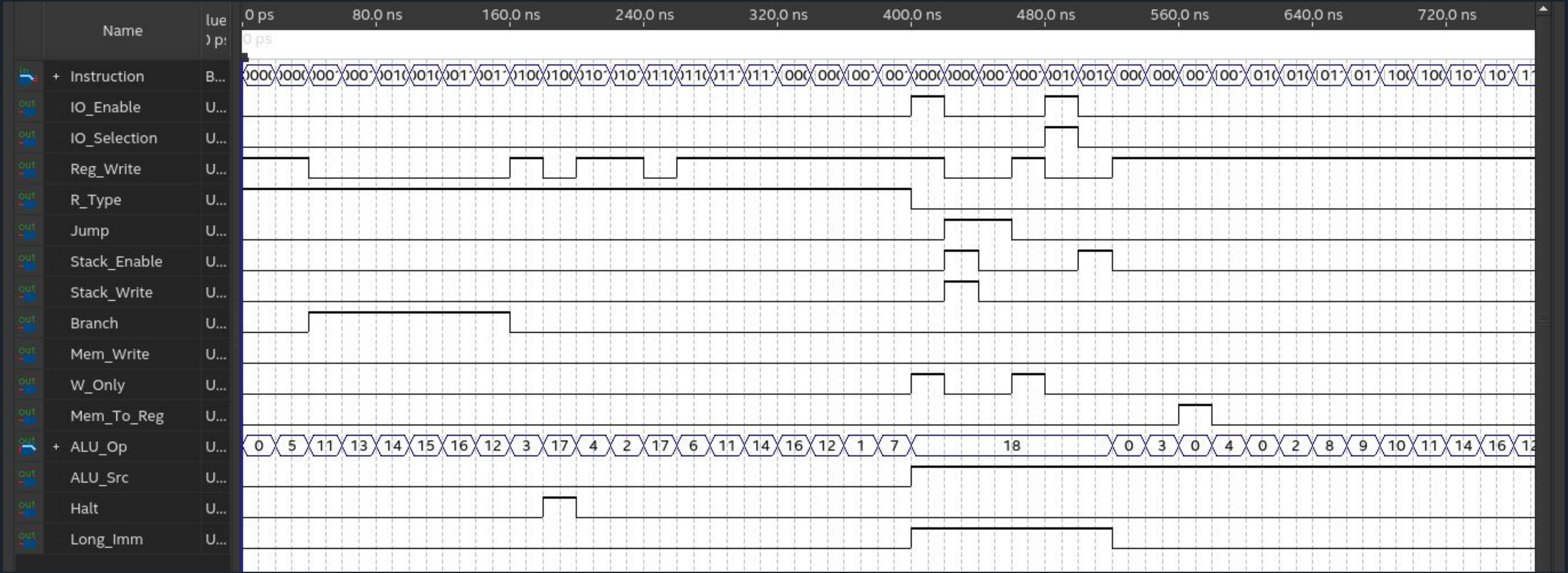


| Name | Value at 0 ps | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| + ALU_Op | U 0 | 0 | 1 | 2 | 3 | 4 | 5 | |
| + Data_1 | U 4 | | | | | | | |
| + Data_2 | U 3 | | | | | | | |
| + Result | U 7 | 7 | 1 | 12 | 1 | | 0 | |
| True | U 0 | | | | | | | |



| Name | Value at 0 ps | | | |
|---|---|---|---|---|
| + ALU_Op | U 0 | 5 | 6 | 7 |
| + Data_1 | B 0000000... | | | |
| + Data_2 | B 0000000... | | | |
| + Result | B 0000000... | 00000000000000000000000000000000 | 00000000000000000000000000000111 | |
| True | U 0 | | | |

# Controle


Name
0 ps
80.0 ns
160.0 ns
240.0 ns
320.0 ns
400.0 ns
480.0 ns
560.0 ns
640.0 ns
720.0 ns
Instruction
IO_Enable
IO_Selection
Reg_Write
R_Type
Jump
Stack_Enable
Stack_Write
Branch
Mem_Write
W_Only
Mem_To_Reg
ALU_Op
ALU_Src
Halt
Long_Imm
0 5 11 13 14 15 16 12 3 17 4 2 17 6 11 14 16 12 1 7 18 0 3 0 4 0 2 8 9 10 11 14 16 12