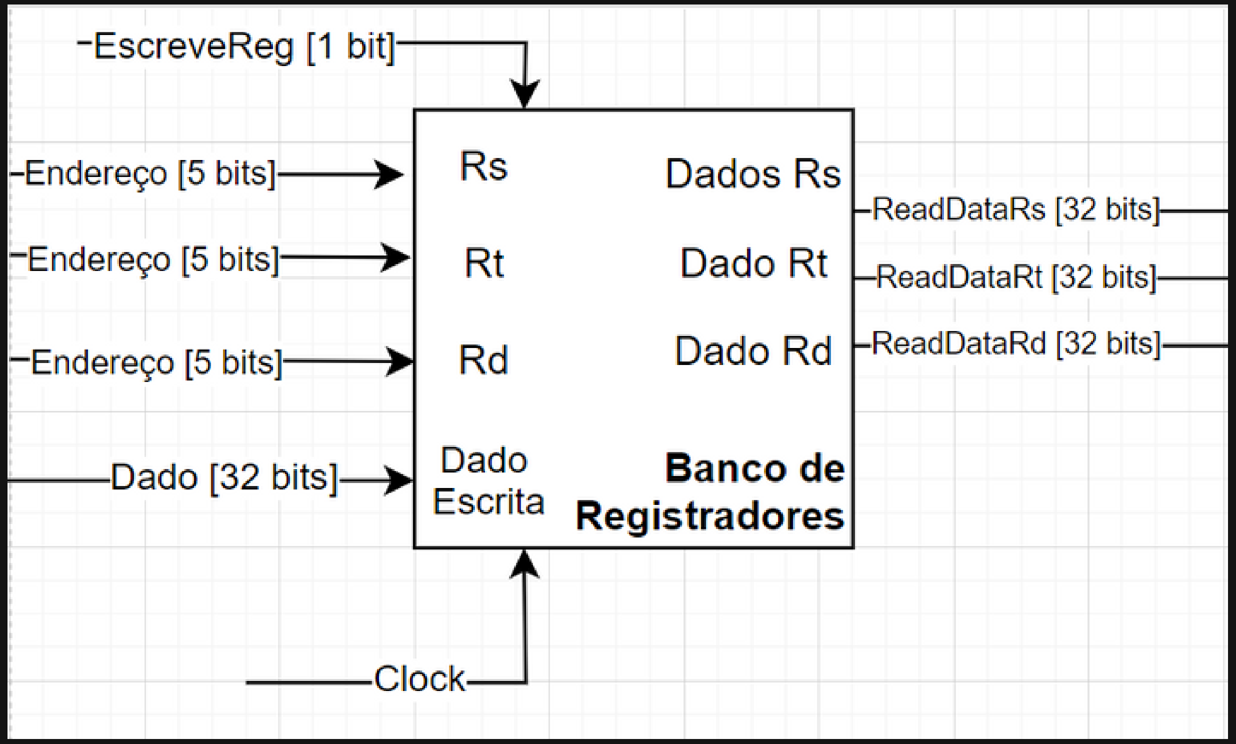
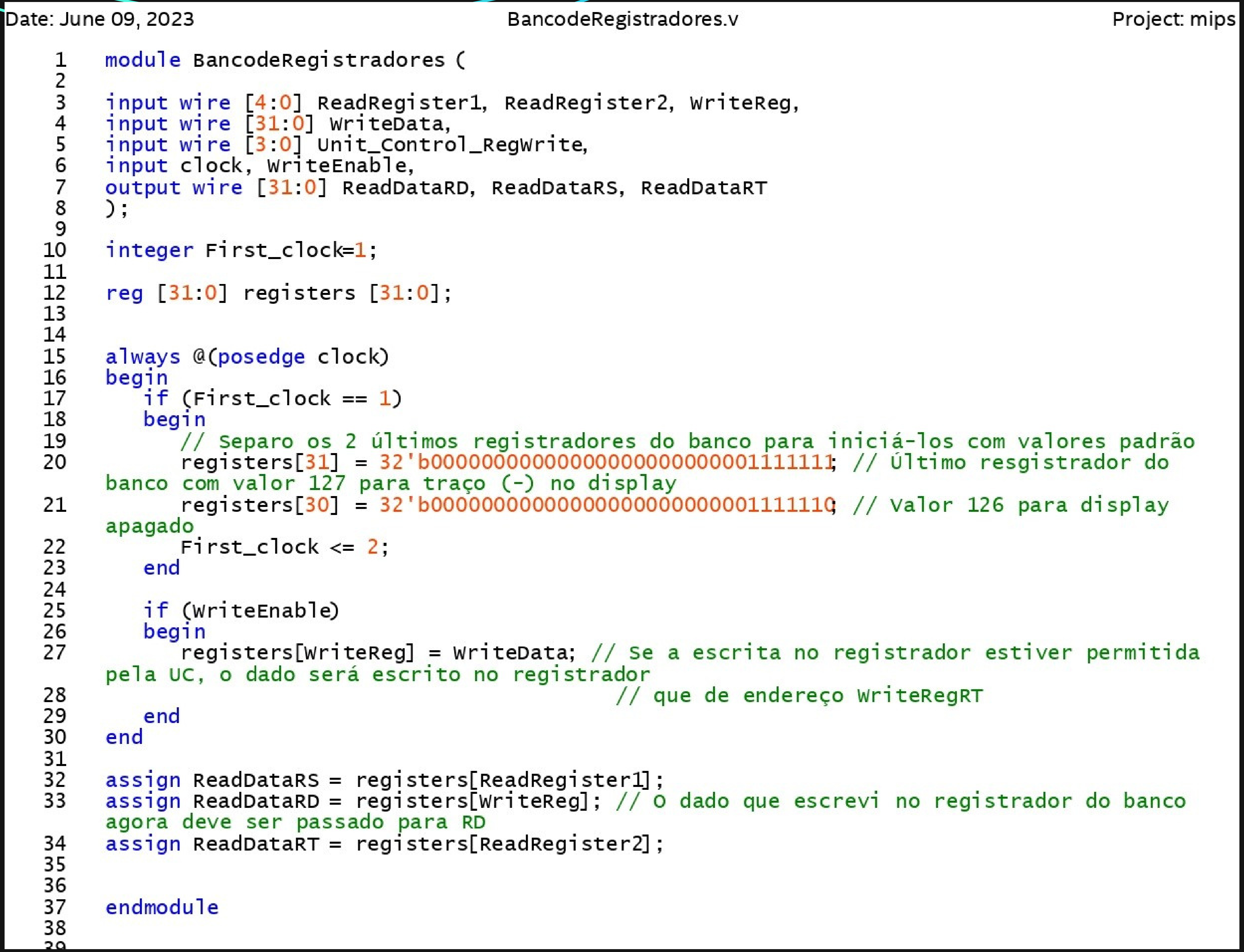
Desenvolvimento de um Processador MIPS - PC3

MATEUS VESPASIANO DE CASTRO 159505

BANCO DE REGISTRADORES

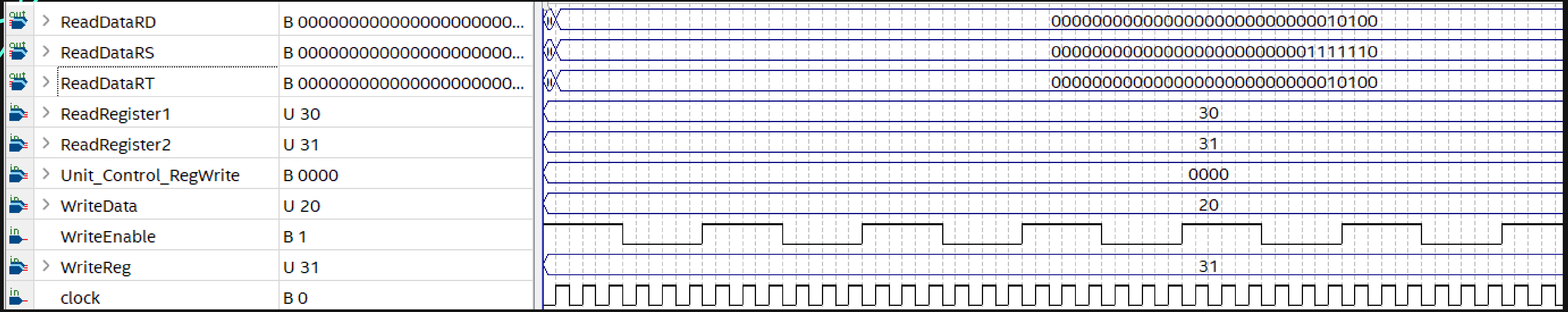




Date: June 09, 2023 BancodeRegistradores.v Project: mips

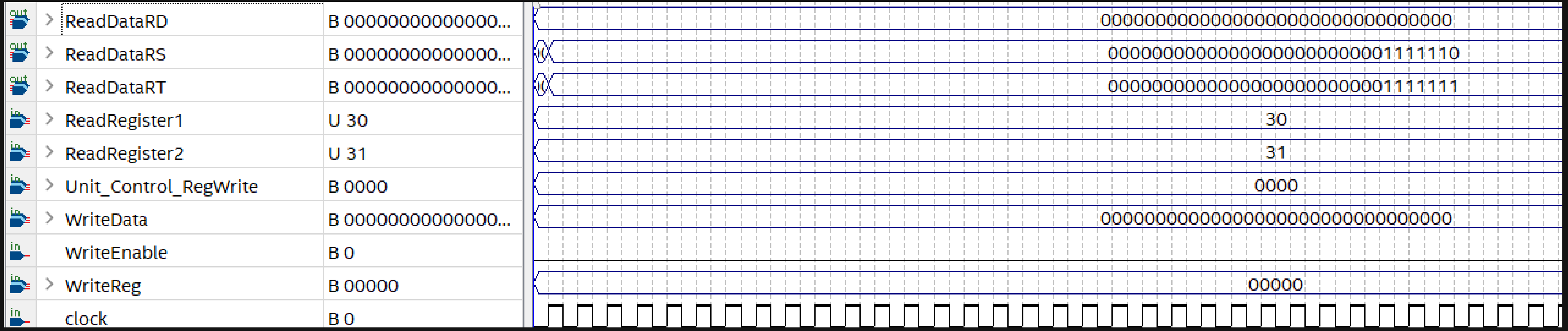
```
module BancodeRegistradores (

input wire [4:0] ReadRegister1, ReadRegister2, WriteReg,
input wire [31:0] WriteData,
input wire [3:0] Unit_Control_RegWrite,
input clock, WriteEnable,
output wire [31:0] ReadDataRD, ReadDataRS, ReadDataRT
);

integer First_clock=1;

reg [31:0] registers [31:0];


always @(posedge clock)
begin
   if (First_clock == 1)
   begin
      // Separo os 2 últimos registradores do banco para iniciá-los com valores padrão
      registers[31] = 32'b00000000000000000000000001111111; // Último resgistrador do banco com valor 127 para traço (-) no display
      registers[30] = 32'b00000000000000000000000001111110; // Valor 126 para display apagado
      First_clock <= 2;
   end

   if (WriteEnable)
   begin
      registers[WriteReg] = WriteData; // Se a escrita no registrador estiver permitida pela UC, o dado será escrito no registrador
                                                               // que de endereço WriteRegRT
   end
end

assign ReadDataRS = registers[ReadRegister1];
assign ReadDataRD = registers[WriteReg]; // O dado que escrevi no registrador do banco agora deve ser passado para RD
assign ReadDataRT = registers[ReadRegister2];


endmodule

```

ESCRITA BANCO



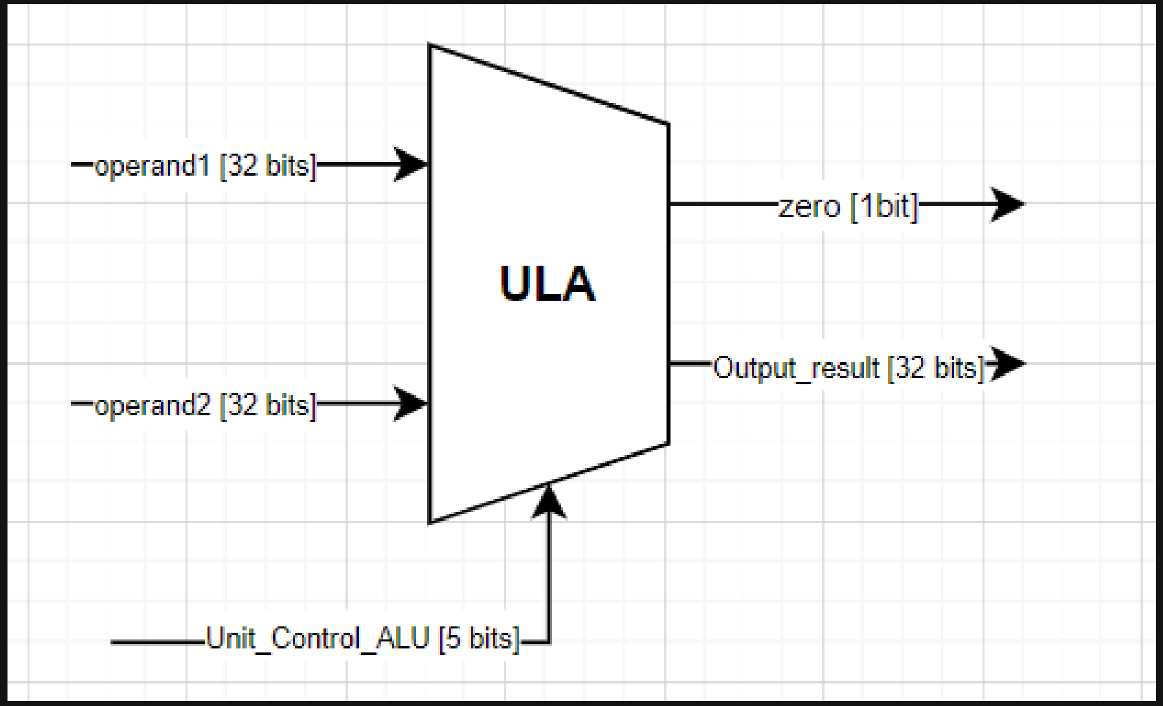
```
ReadDataRD            B 00000000000000000000...   00000000000000000000000000010100
ReadDataRS            B 00000000000000000000...   00000000000000000000000001111110
ReadDataRT            B 00000000000000000000...   00000000000000000000000000010100
ReadRegister1         U 30                        30
ReadRegister2         U 31                        31
Unit_Control_RegWrite B 0000                      0000
WriteData             U 20                        20
WriteEnable           B 1
WriteReg              U 31                        31
clock                 B 0
```

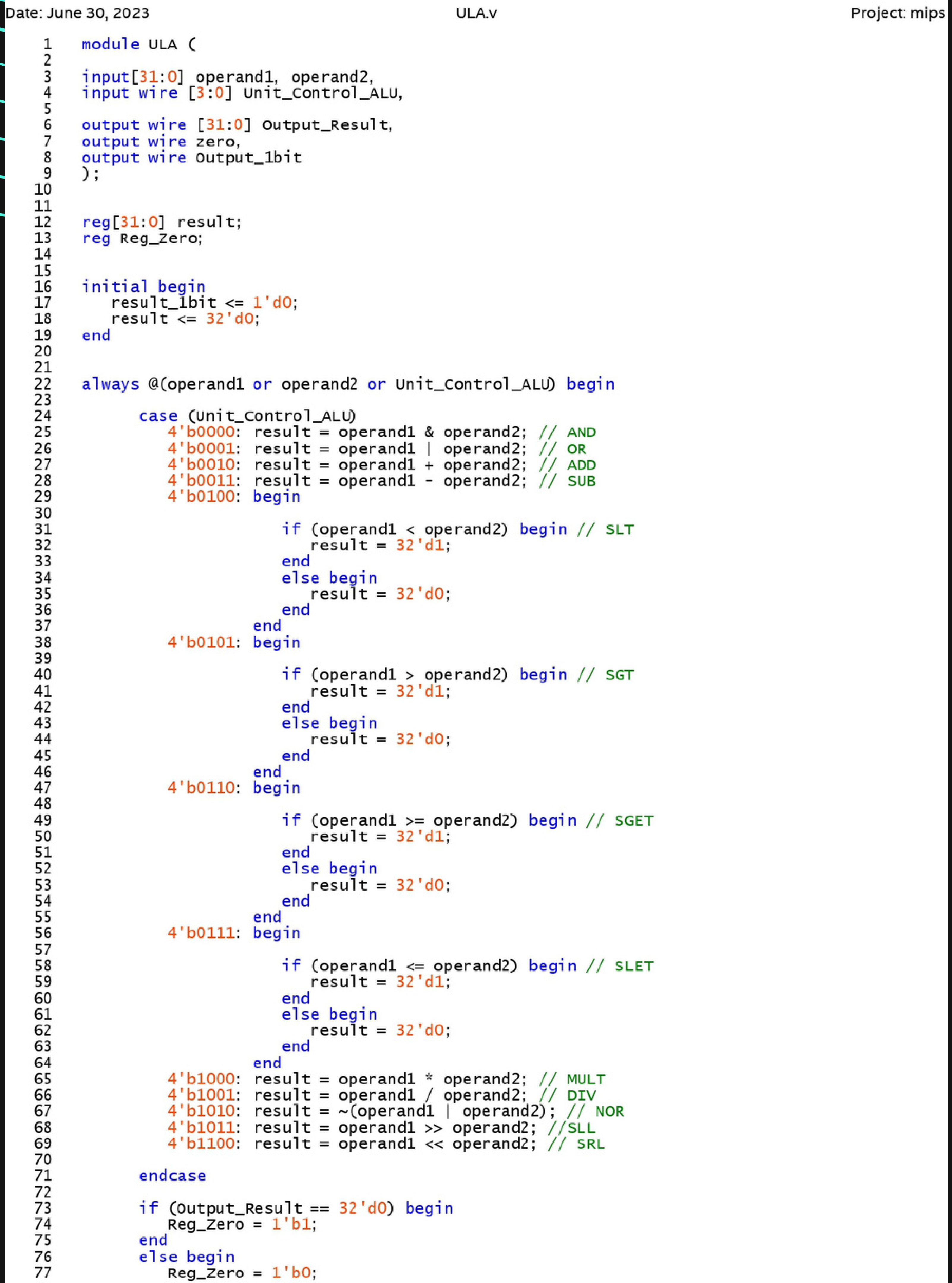
LEITURA BANCO



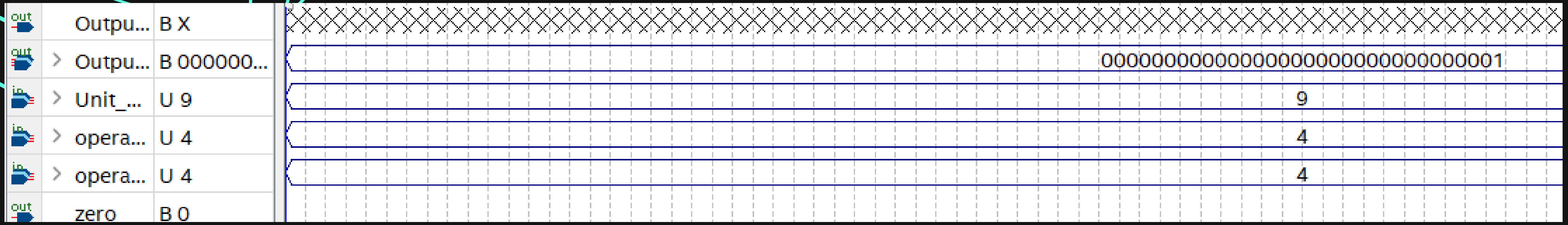
```
ReadDataRD            B 00000000000000...   00000000000000000000000000000000
ReadDataRS            B 00000000000000...   00000000000000000000000001111110
ReadDataRT            B 00000000000000...   00000000000000000000000001111111
ReadRegister1         U 30                  30
ReadRegister2         U 31                  31
Unit_Control_RegWrite B 0000                0000
WriteData             B 00000000000000...   00000000000000000000000000000000
WriteEnable           B 0
WriteReg              B 00000               00000
clock                 B 0
```

UNIDADE LÓGICA E ARITMÉTICA



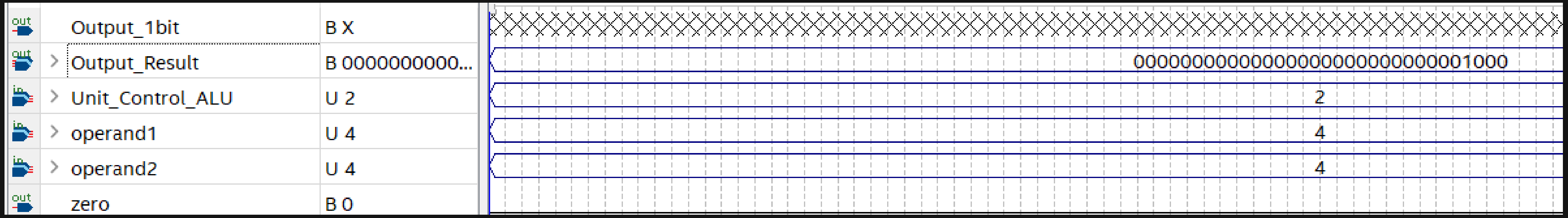

Date: June 30, 2023 ULA.v Project: mips

```
module ULA (

input[31:0] operand1, operand2,
input wire [3:0] Unit_Control_ALU,

output wire [31:0] Output_Result,
output wire zero,
output wire Output_1bit
);


reg[31:0] result;
reg Reg_Zero;


initial begin
   result_1bit <= 1'd0;
   result <= 32'd0;
end


always @(operand1 or operand2 or Unit_Control_ALU) begin

      case (Unit_Control_ALU)
         4'b0000: result = operand1 & operand2; // AND
         4'b0001: result = operand1 | operand2; // OR
         4'b0010: result = operand1 + operand2; // ADD
         4'b0011: result = operand1 - operand2; // SUB
         4'b0100: begin

                     if (operand1 < operand2) begin // SLT
                        result = 32'd1;
                     end
                     else begin
                        result = 32'd0;
                     end
                  end
         4'b0101: begin

                     if (operand1 > operand2) begin // SGT
                        result = 32'd1;
                     end
                     else begin
                        result = 32'd0;
                     end
                  end
         4'b0110: begin

                     if (operand1 >= operand2) begin // SGET
                        result = 32'd1;
                     end
                     else begin
                        result = 32'd0;
                     end
                  end
         4'b0111: begin

                     if (operand1 <= operand2) begin // SLET
                        result = 32'd1;
                     end
                     else begin
                        result = 32'd0;
                     end
                  end
         4'b1000: result = operand1 * operand2; // MULT
         4'b1001: result = operand1 / operand2; // DIV
         4'b1010: result = ~(operand1 | operand2); // NOR
         4'b1011: result = operand1 >> operand2; //SLL
         4'b1100: result = operand1 << operand2; // SRL

      endcase

      if (Output_Result == 32'd0) begin
         Reg_Zero = 1'b1;
      end
      else begin
         Reg_Zero = 1'b0;
```



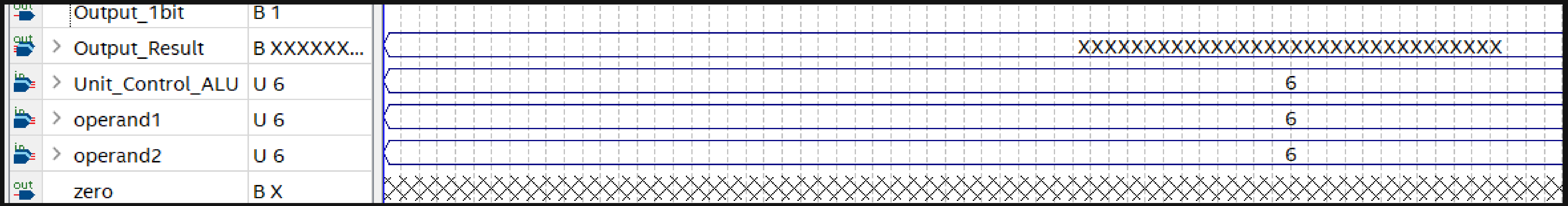
Outpu...	B X	
Outpu...	B 000000...	00000000000000000000000000000001
Unit_...	U 9	9
opera...	U 4	4
opera...	U 4	4
zero	B 0	

DIV



Output_1bit	B X	
Output_Result	B 0000000000...	00000000000000000000000000001000
Unit_Control_ALU	U 2	2
operand1	U 4	4
operand2	U 4	4
zero	B 0	

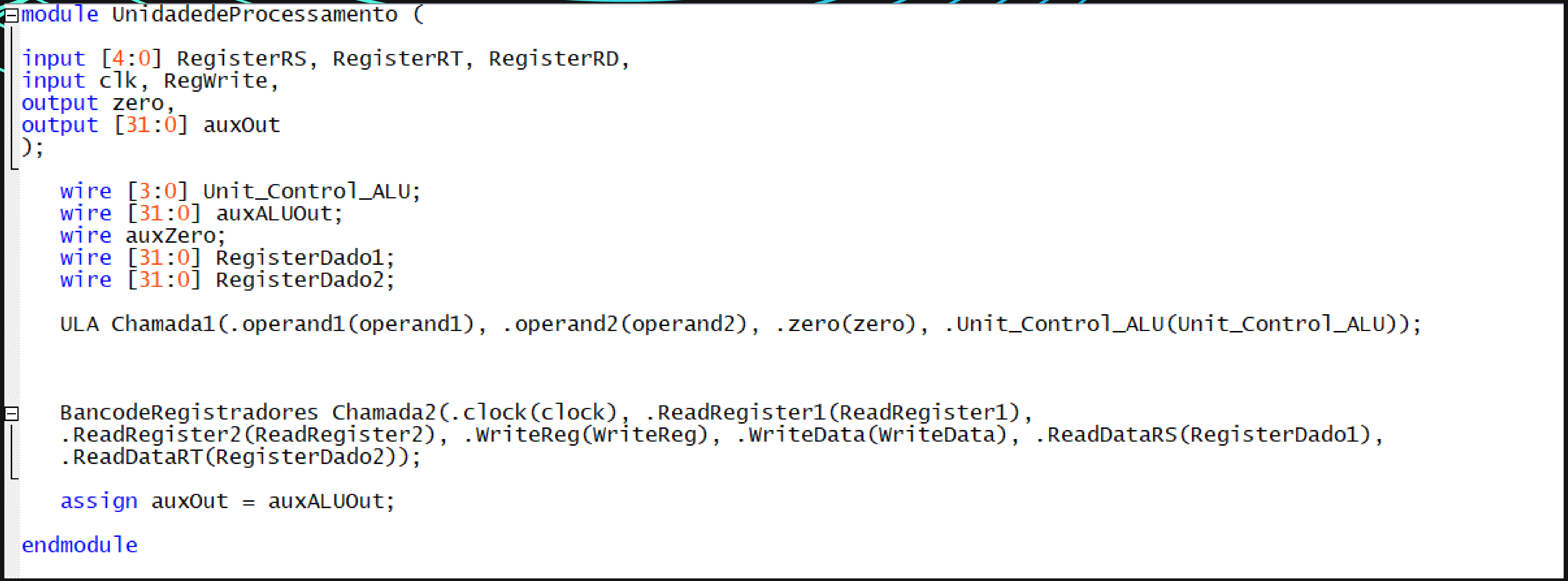
ADD



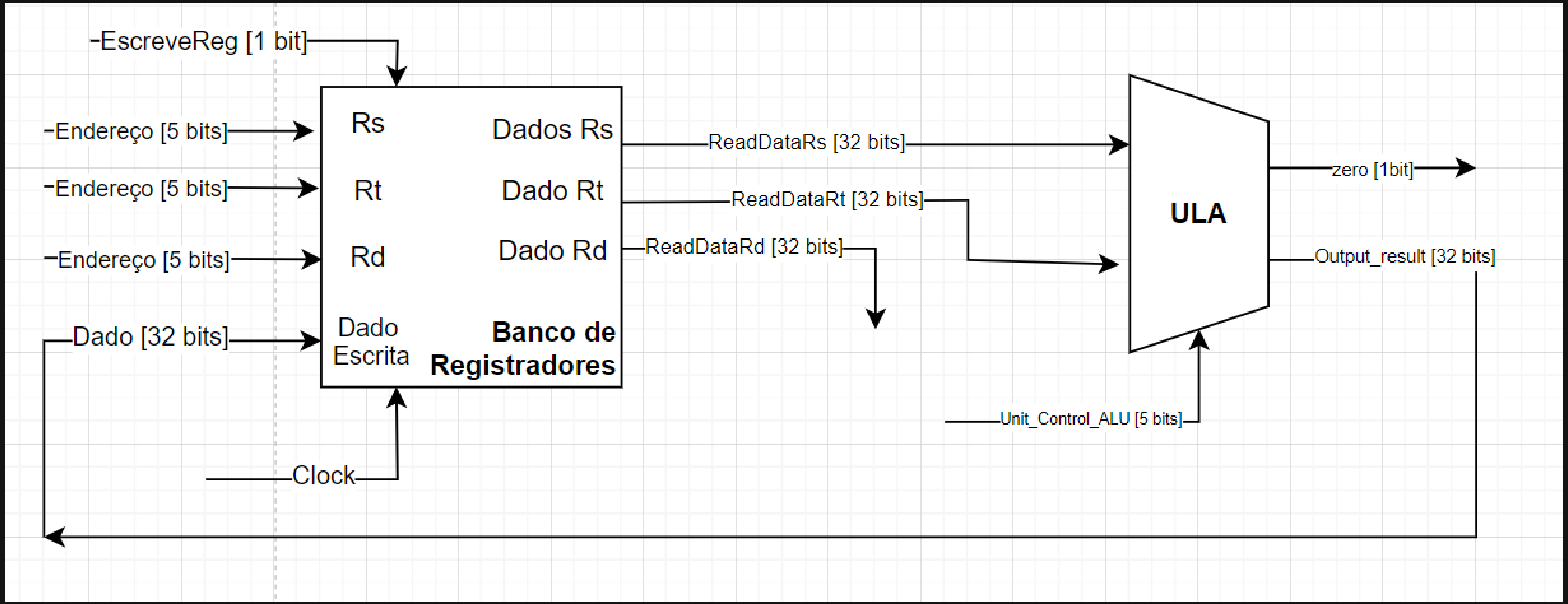
Output_1bit	B 1	
Output_Result	B XXXXXX...	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Unit_Control_ALU	U 6	6
operand1	U 6	6
operand2	U 6	6
zero	B X	

SGET

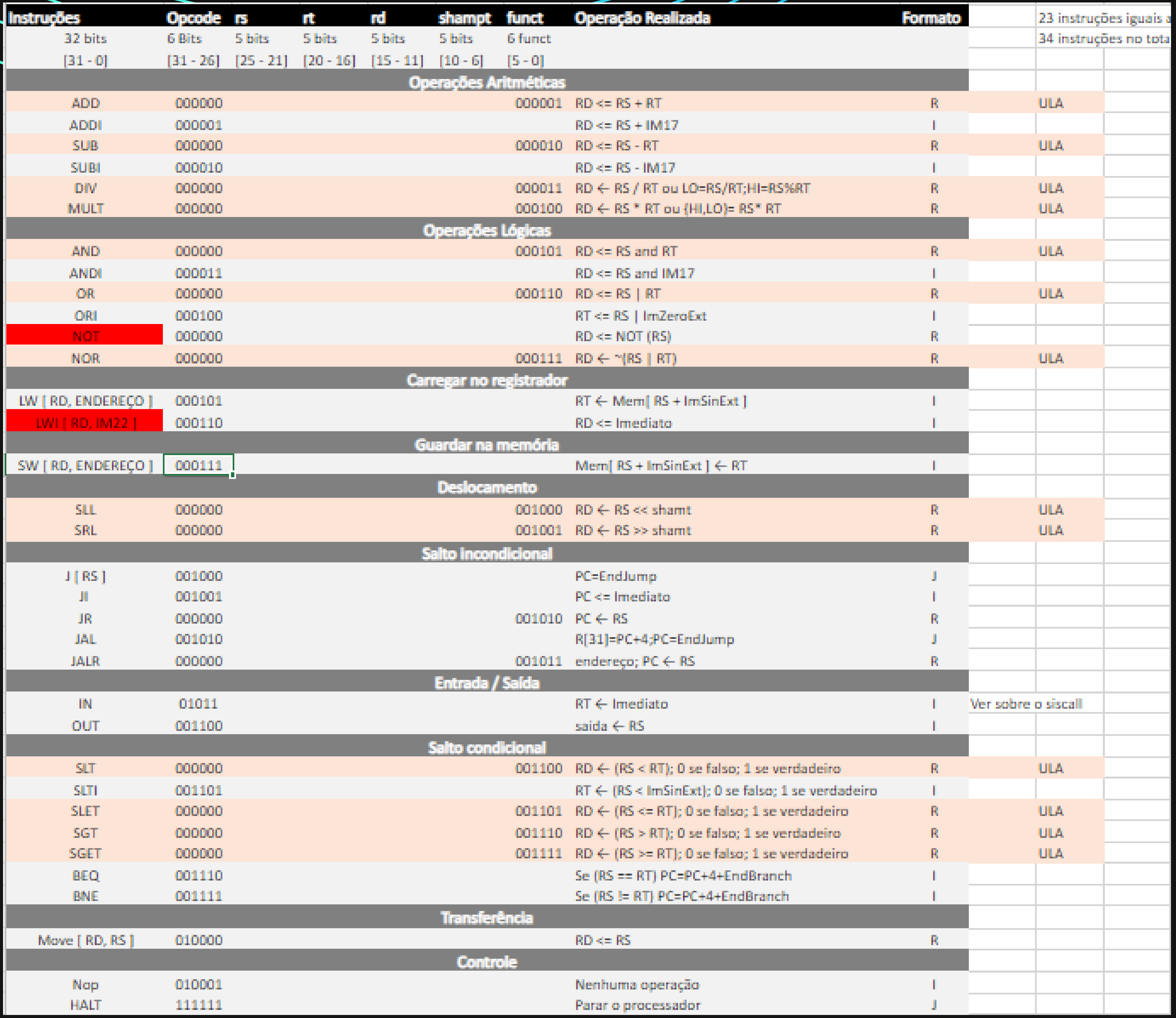
UNIDADE DE PROCESSAMENTO



```
module UnidadedeProcessamento (

input [4:0] RegisterRS, RegisterRT, RegisterRD,
input clk, RegWrite,
output zero,
output [31:0] auxOut
);

   wire [3:0] Unit_Control_ALU;
   wire [31:0] auxALUOut;
   wire auxZero;
   wire [31:0] RegisterDado1;
   wire [31:0] RegisterDado2;

   ULA Chamada1(.operand1(operand1), .operand2(operand2), .zero(zero), .Unit_Control_ALU(Unit_Control_ALU));


   BancodeRegistradores Chamada2(.clock(clock), .ReadRegister1(ReadRegister1),
   .ReadRegister2(ReadRegister2), .WriteReg(WriteReg), .WriteData(WriteData), .ReadDataRS(RegisterDado1),
   .ReadDataRT(RegisterDado2));

   assign auxOut = auxALUOut;

endmodule
```

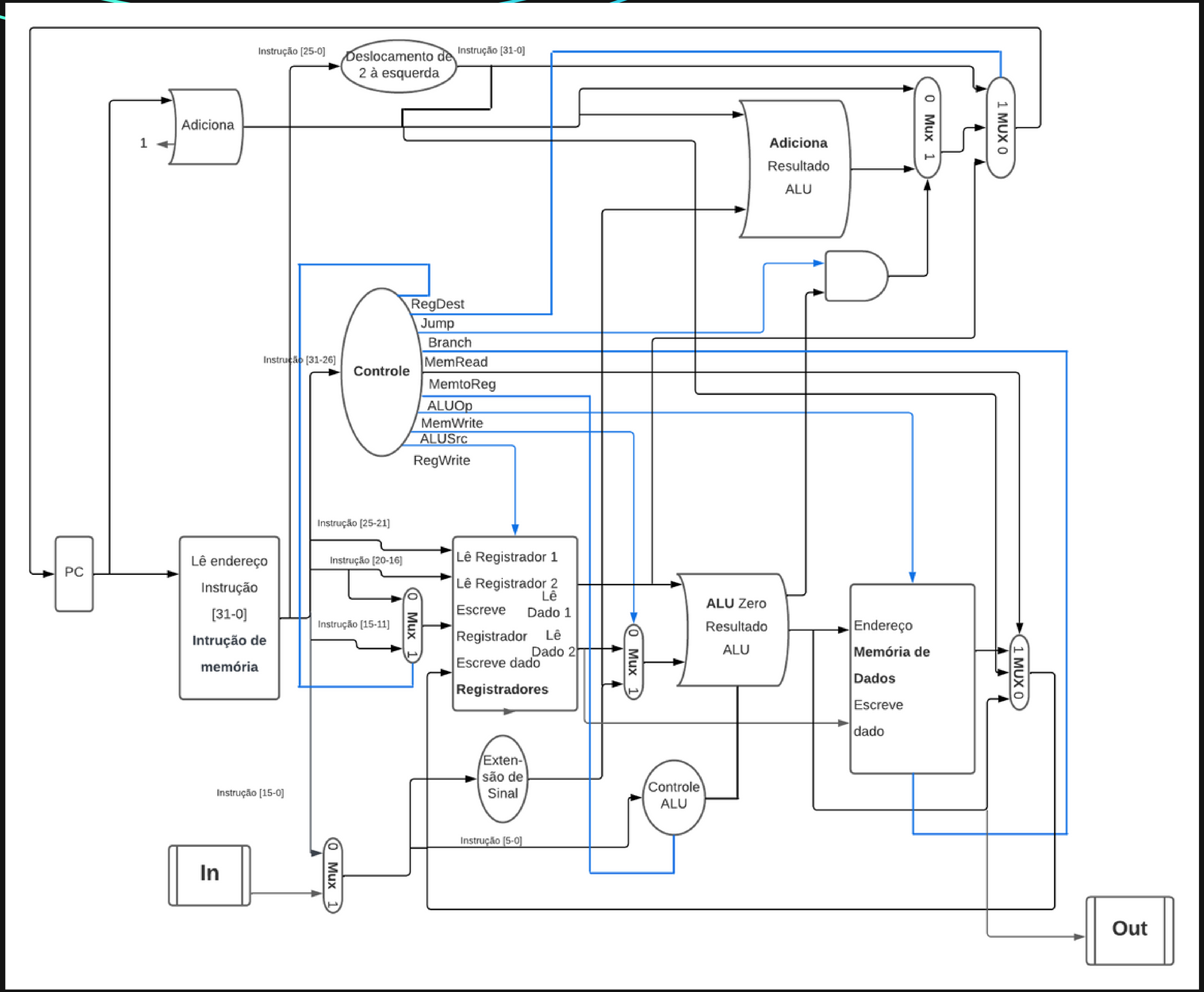


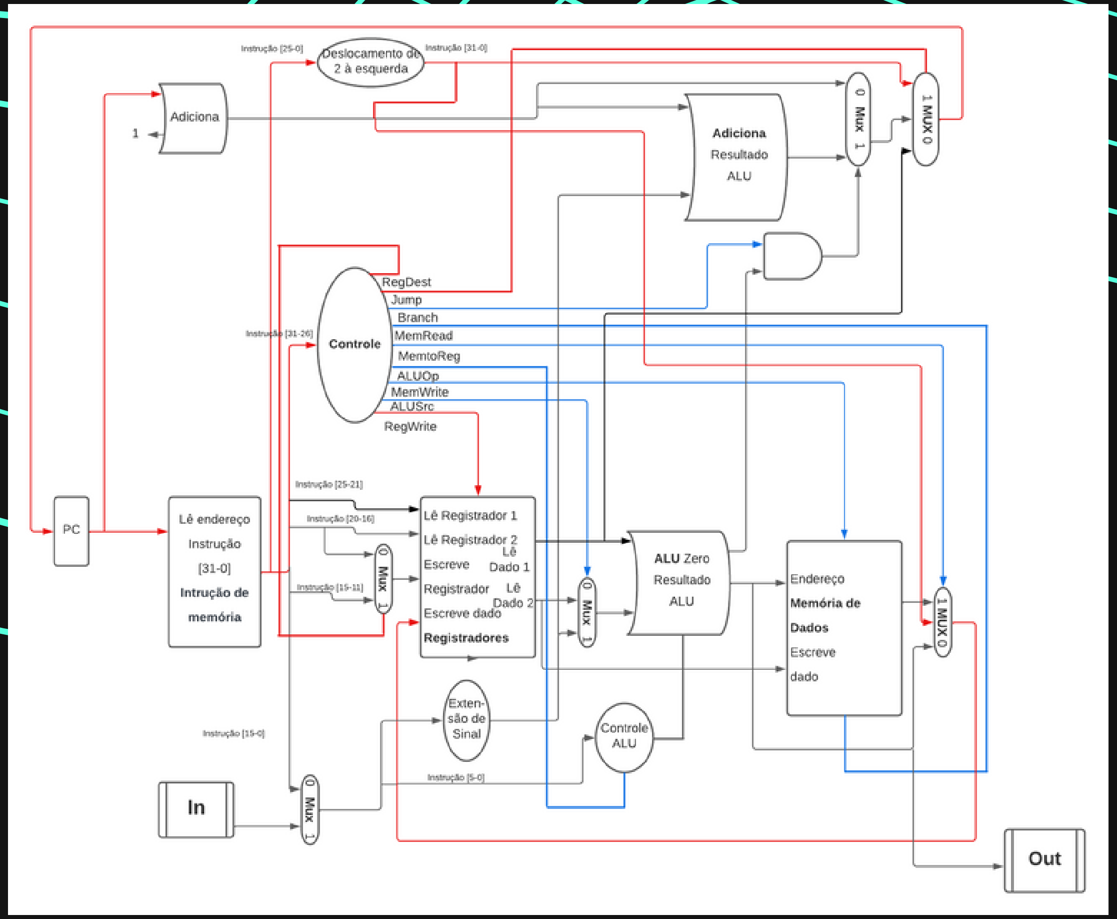
UNIDADE DE PROCESSAMENTO - PLANEJAMENTO



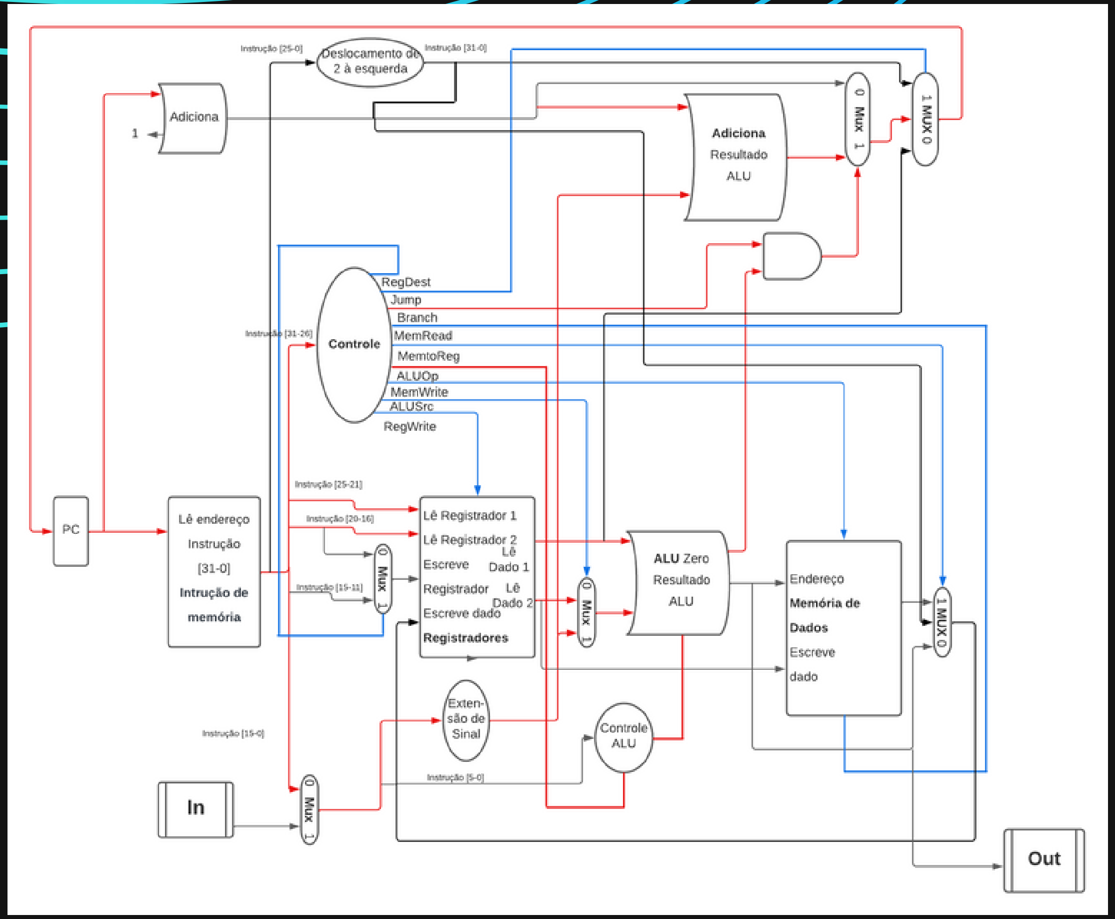
Instruções	Opcode	rs	rt	rd	shampt	funct	Operação Realizada	Formato		
32 bits	6 Bits	5 bits	5 bits	5 bits	5 bits	6 funct				23 instruções iguais a
[31 - 0]	[31 - 26]	[25 - 21]	[20 - 16]	[15 - 11]	[10 - 6]	[5 - 0]				34 instruções no tota
Operações Aritméticas										
ADD	000000					000001	RD <= RS + RT	R		ULA
ADDI	000001						RD <= RS + IM17	I		
SUB	000000					000010	RD <= RS - RT	R		ULA
SUBI	000010						RD <= RS - IM17	I		
DIV	000000					000011	RD ← RS / RT ou LO=RS/RT;HI=RS%RT	R		ULA
MULT	000000					000100	RD ← RS * RT ou {HI,LO}= RS* RT	R		ULA
Operações Lógicas										
AND	000000					000101	RD <= RS and RT	R		ULA
ANDI	000011						RD <= RS and IM17	I		
OR	000000					000110	RD <= RS | RT	R		ULA
ORI	000100						RT <= RS | ImZeroExt	I		
NOT	000000						RD <= NOT (RS)	R		
NOR	000000					000111	RD ← ~(RS | RT)	R		ULA
Carregar no registrador										
LW [RD, ENDEREÇO]	000101						RT ← Mem[RS + ImSinExt]	I		
LWI [RD, IM22]	000110						RD <= Imediato	I		
Guardar na memória										
SW [RD, ENDEREÇO]	000111						Mem[RS + ImSinExt] ← RT	I		
Deslocamento										
SLL	000000					001000	RD ← RS << shamt	R		ULA
SRL	000000					001001	RD ← RS >> shamt	R		ULA
Salto Incondicional										
J [RS]	001000						PC=EndJump	J		
JI	001001						PC <= Imediato	I		
JR	000000					001010	PC ← RS	R		
JAL	001010						R[31]=PC+4;PC=EndJump	J		
JALR	000000					001011	endereço; PC ← RS	R		
Entrada / Saída										
IN	01011						RT ← Imediato	I	Ver sobre o siscall	
OUT	001100						saida ← RS	I		
Salto condicional										
SLT	000000					001100	RD ← (RS < RT); 0 se falso; 1 se verdadeiro	R		ULA
SLTI	001101						RT ← (RS < ImSinExt); 0 se falso; 1 se verdadeiro	I		
SLET	000000					001101	RD ← (RS <= RT); 0 se falso; 1 se verdadeiro	R		ULA
SGT	000000					001110	RD ← (RS > RT); 0 se falso; 1 se verdadeiro	R		ULA
SGET	000000					001111	RD ← (RS >= RT); 0 se falso; 1 se verdadeiro	R		ULA
BEQ	001110						Se (RS == RT) PC=PC+4+EndBranch	I		
BNE	001111						Se (RS != RT) PC=PC+4+EndBranch	I		
Transferência										
Move [RD, RS]	010000						RD <= RS	R		
Controle										
Nop	010001						Nenhuma operação	I		
HALT	111111						Parar o processador	J		

UNIDADE DE PROCESSAMENTO - PLANEJAMENTO





JAL

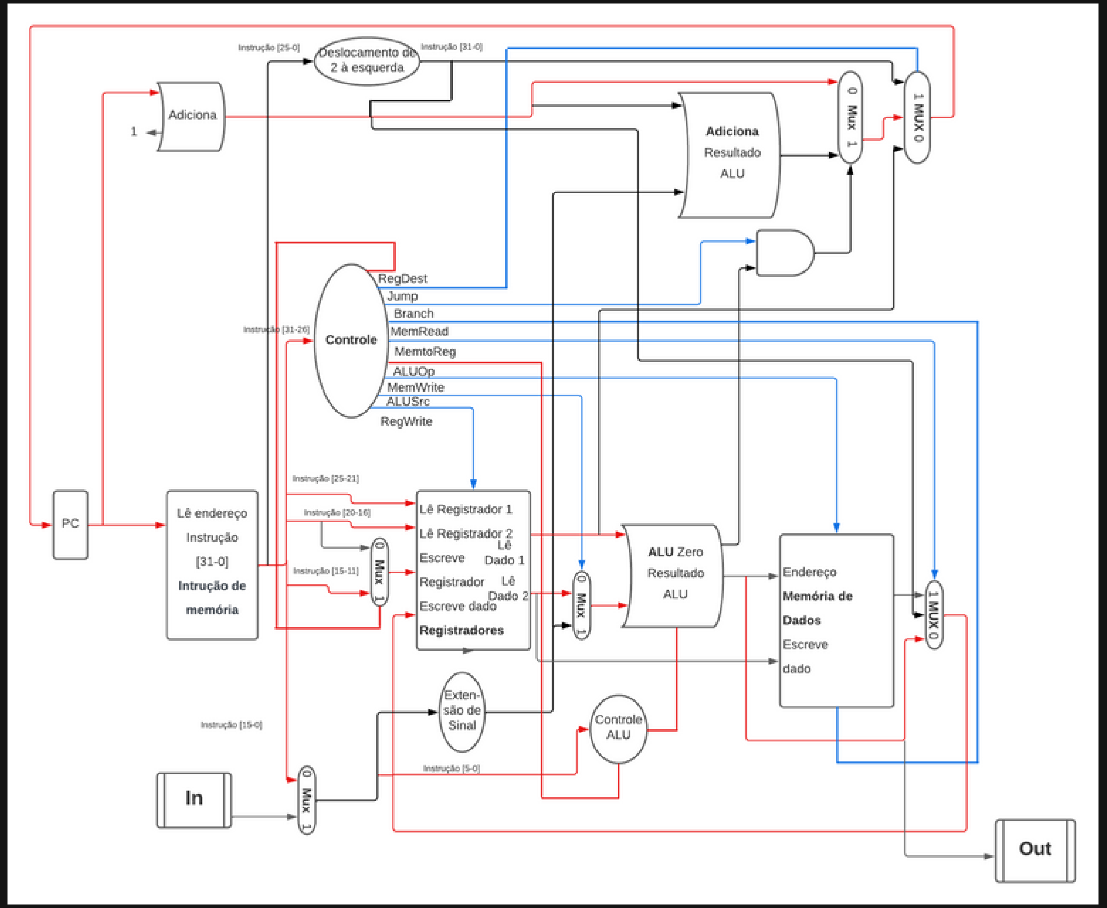


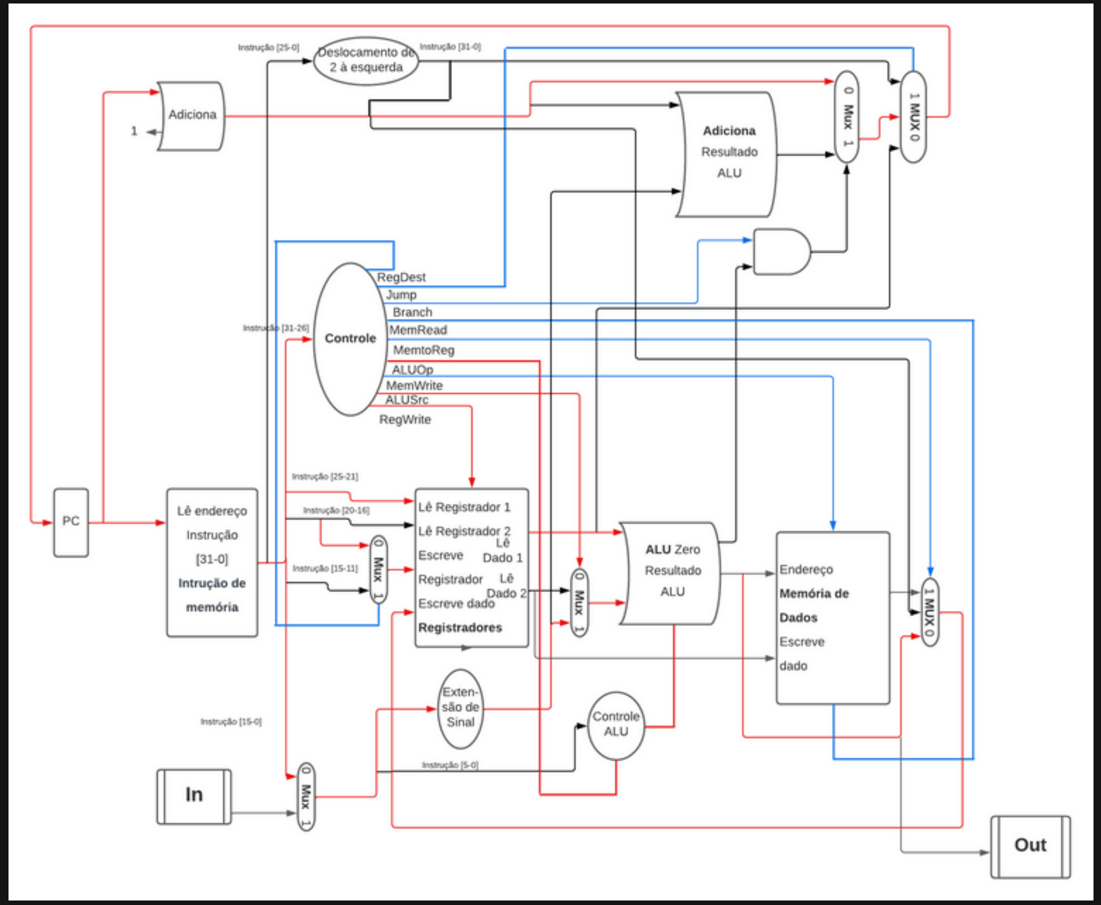
BRANCH

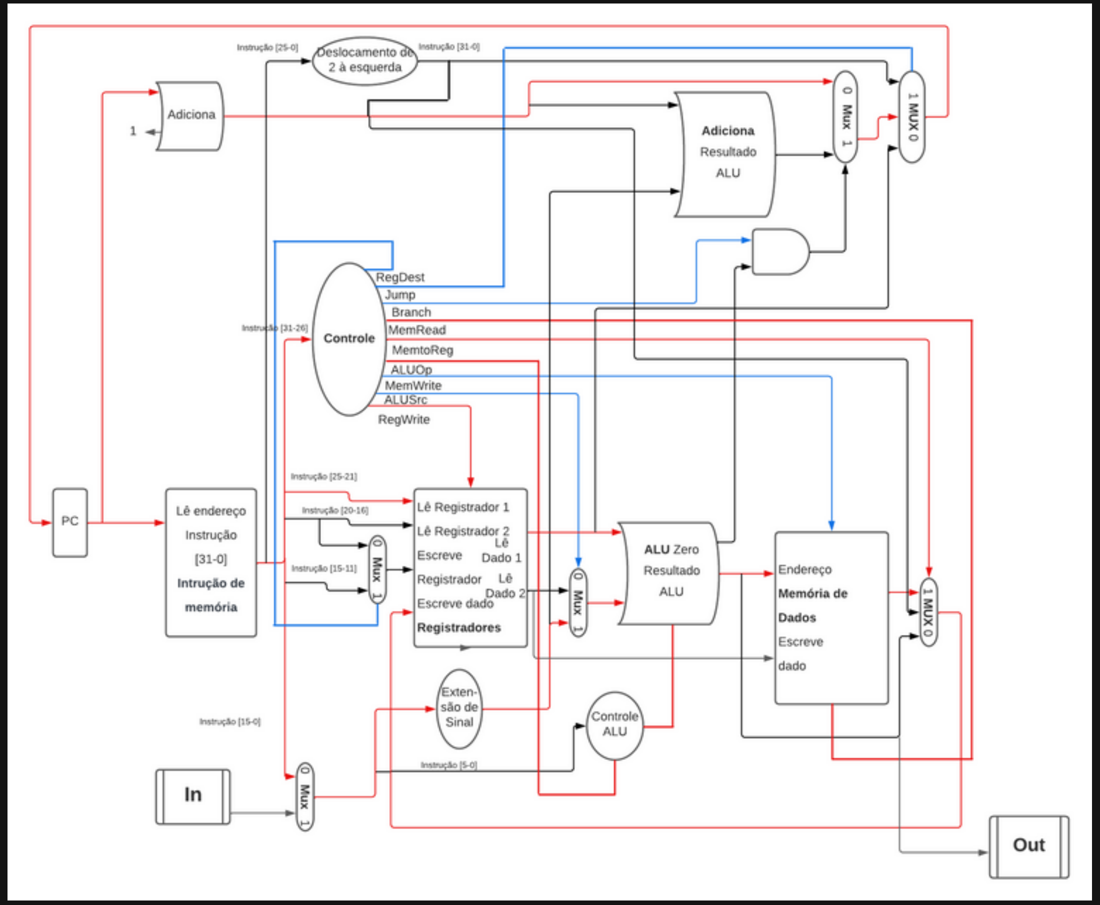
TIPO R

TIPO I

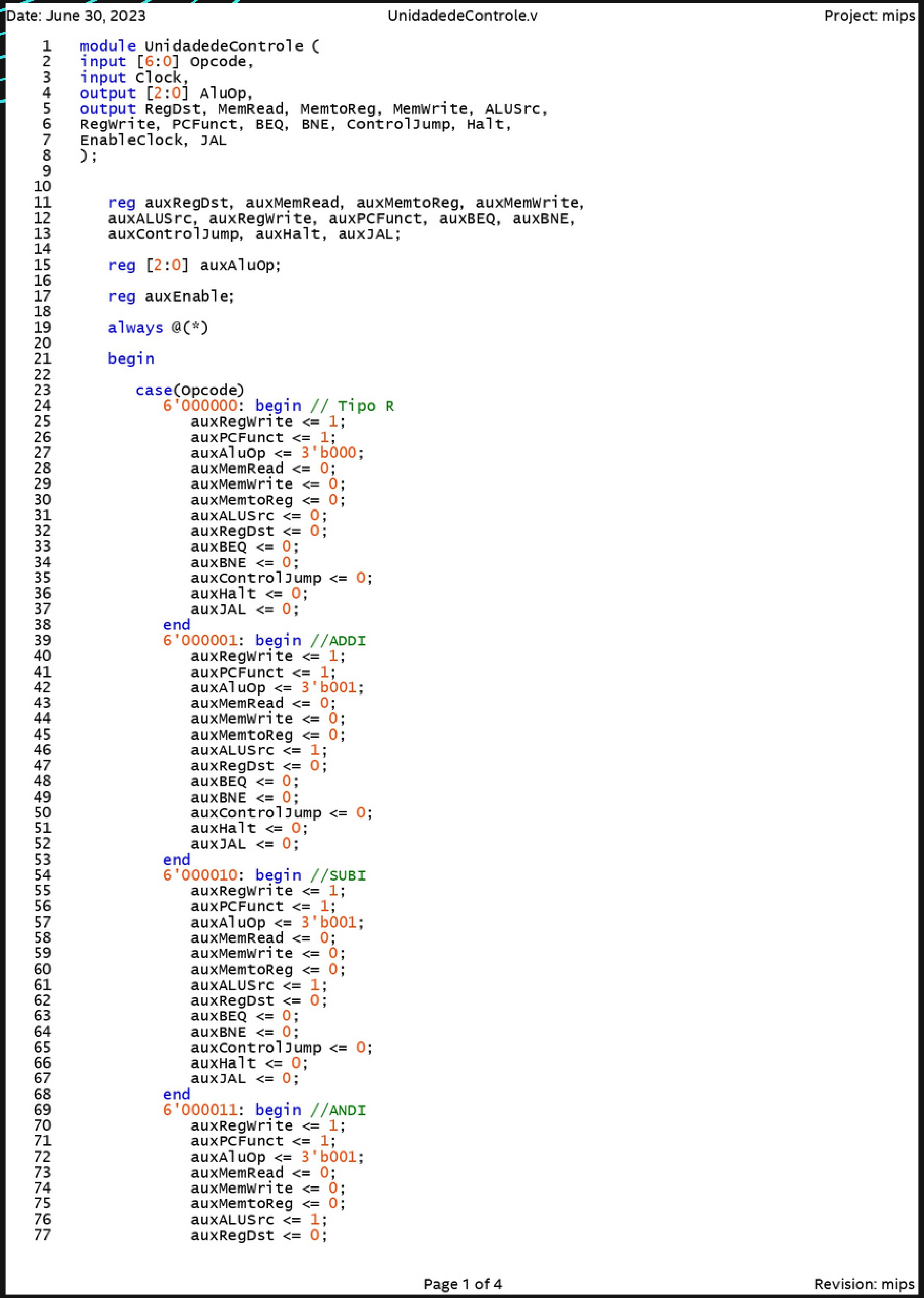
LW





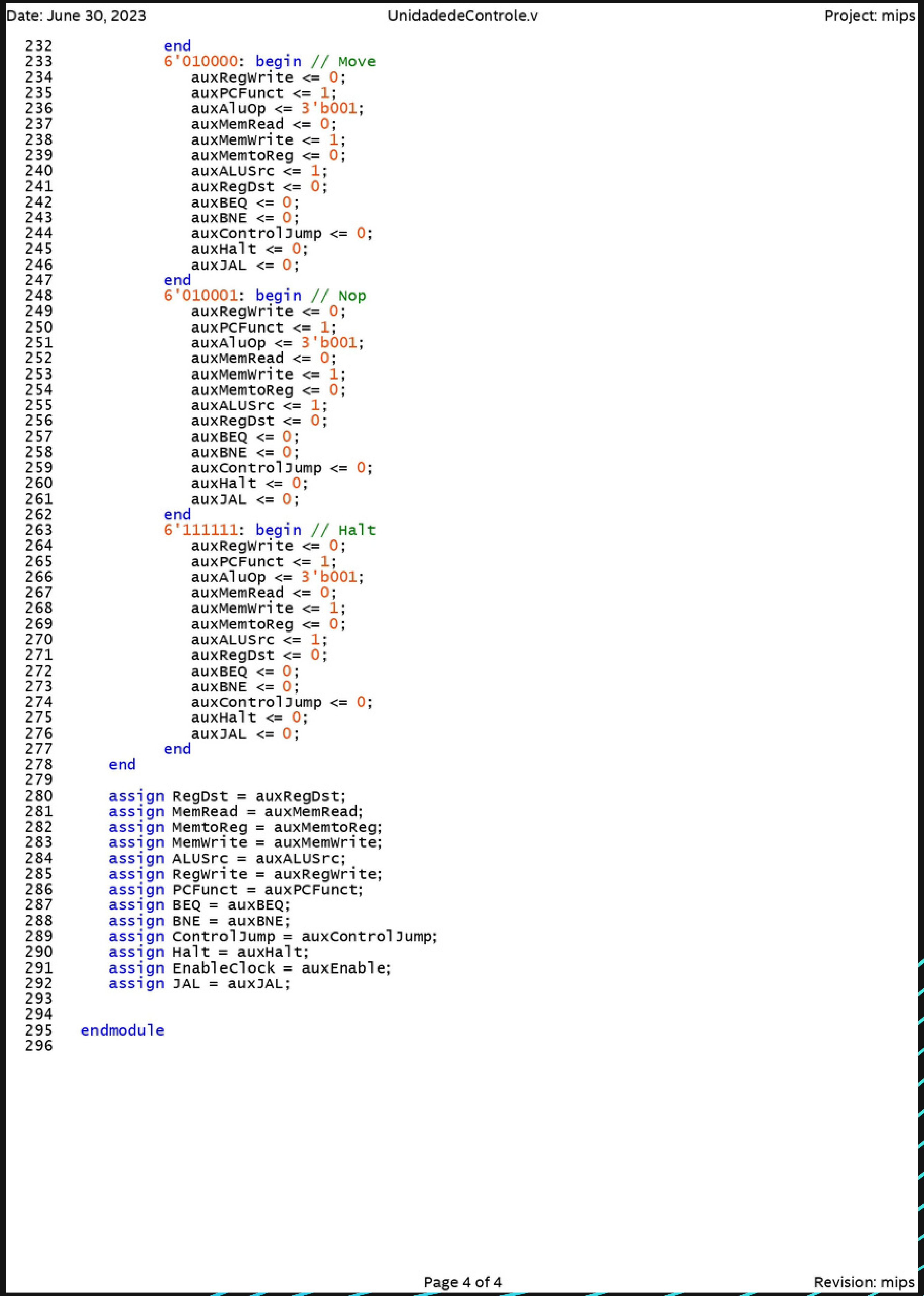


UNIDADE DE CONTROLE


Date: June 30, 2023 UnidadedeControle.v Project: mips

```verilog
module UnidadedeControle (
input [6:0] Opcode,
input Clock,
output [2:0] AluOp,
output RegDst, MemRead, MemtoReg, MemWrite, ALUSrc,
RegWrite, PCFunct, BEQ, BNE, ControlJump, Halt,
EnableClock, JAL
);


   reg auxRegDst, auxMemRead, auxMemtoReg, auxMemWrite,
   auxALUSrc, auxRegWrite, auxPCFunct, auxBEQ, auxBNE,
   auxControlJump, auxHalt, auxJAL;

   reg [2:0] auxAluOp;

   reg auxEnable;

   always @(*)

   begin

      case(Opcode)
         6'000000: begin // Tipo R
            auxRegWrite <= 1;
            auxPCFunct <= 1;
            auxAluOp <= 3'b000;
            auxMemRead <= 0;
            auxMemWrite <= 0;
            auxMemtoReg <= 0;
            auxALUSrc <= 0;
            auxRegDst <= 0;
            auxBEQ <= 0;
            auxBNE <= 0;
            auxControlJump <= 0;
            auxHalt <= 0;
            auxJAL <= 0;
         end
         6'000001: begin //ADDI
            auxRegWrite <= 1;
            auxPCFunct <= 1;
            auxAluOp <= 3'b001;
            auxMemRead <= 0;
            auxMemWrite <= 0;
            auxMemtoReg <= 0;
            auxALUSrc <= 1;
            auxRegDst <= 0;
            auxBEQ <= 0;
            auxBNE <= 0;
            auxControlJump <= 0;
            auxHalt <= 0;
            auxJAL <= 0;
         end
         6'000010: begin //SUBI
            auxRegWrite <= 1;
            auxPCFunct <= 1;
            auxAluOp <= 3'b001;
            auxMemRead <= 0;
            auxMemWrite <= 0;
            auxMemtoReg <= 0;
            auxALUSrc <= 1;
            auxRegDst <= 0;
            auxBEQ <= 0;
            auxBNE <= 0;
            auxControlJump <= 0;
            auxHalt <= 0;
            auxJAL <= 0;
         end
         6'000011: begin //ANDI
            auxRegWrite <= 1;
            auxPCFunct <= 1;
            auxAluOp <= 3'b001;
            auxMemRead <= 0;
            auxMemWrite <= 0;
            auxMemtoReg <= 0;
            auxALUSrc <= 1;
            auxRegDst <= 0;
```

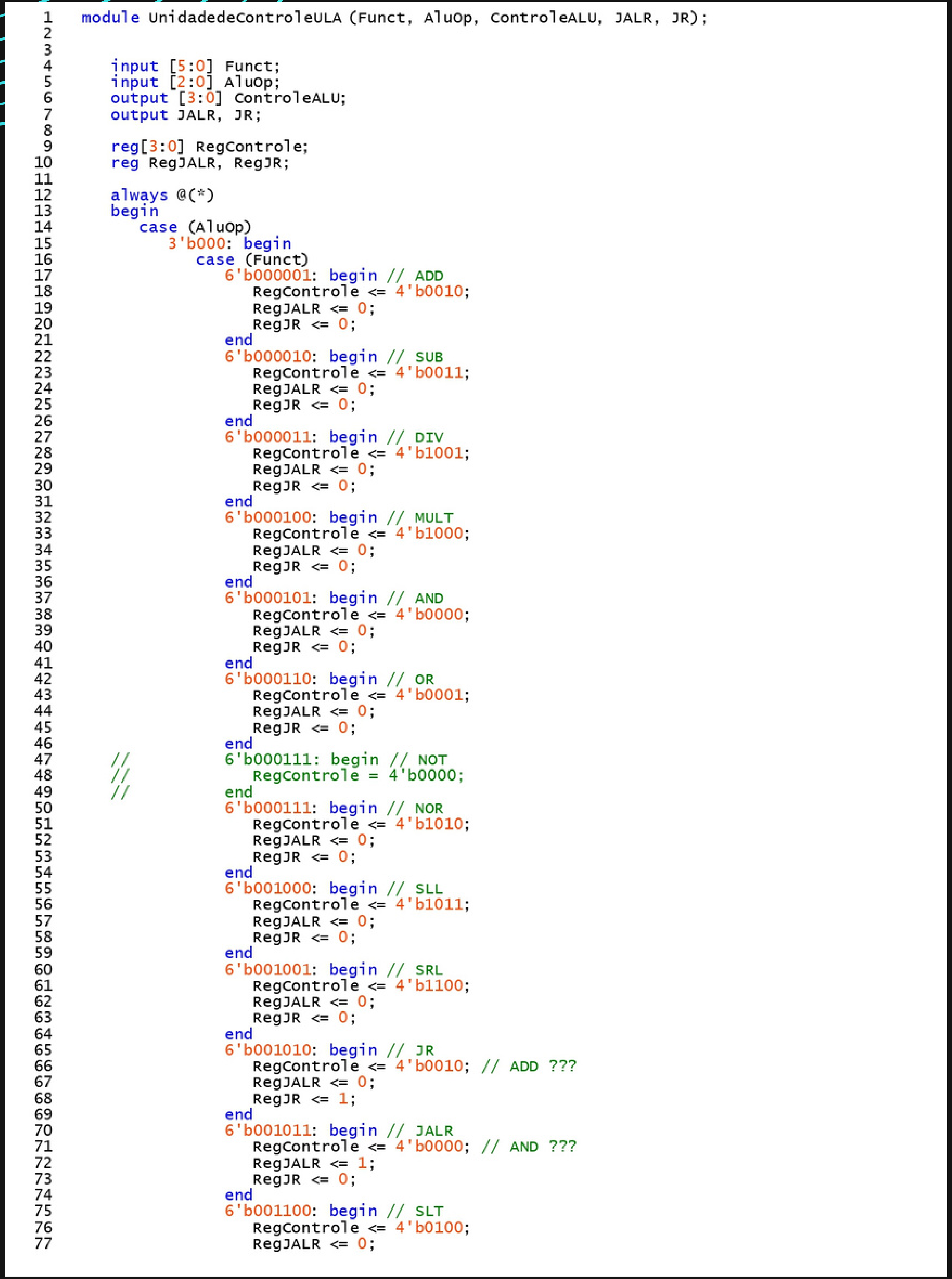
Page 1 of 4 Revision: mips


Date: June 30, 2023 UnidadedeControle.v Project: mips

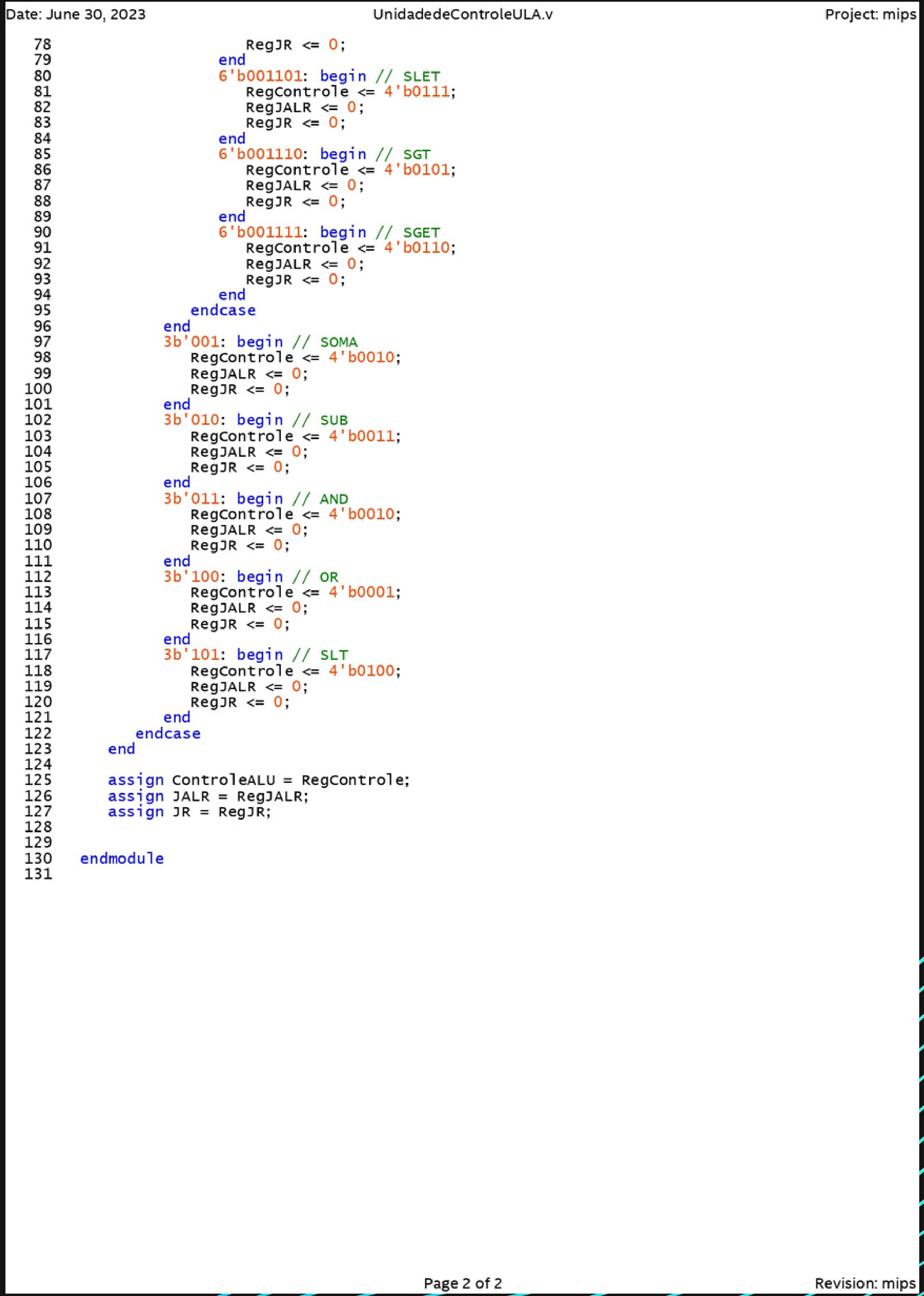
```verilog
         end
         6'010000: begin // Move
            auxRegWrite <= 0;
            auxPCFunct <= 1;
            auxAluOp <= 3'b001;
            auxMemRead <= 0;
            auxMemWrite <= 1;
            auxMemtoReg <= 0;
            auxALUSrc <= 1;
            auxRegDst <= 0;
            auxBEQ <= 0;
            auxBNE <= 0;
            auxControlJump <= 0;
            auxHalt <= 0;
            auxJAL <= 0;
         end
         6'010001: begin // Nop
            auxRegWrite <= 0;
            auxPCFunct <= 1;
            auxAluOp <= 3'b001;
            auxMemRead <= 0;
            auxMemWrite <= 1;
            auxMemtoReg <= 0;
            auxALUSrc <= 1;
            auxRegDst <= 0;
            auxBEQ <= 0;
            auxBNE <= 0;
            auxControlJump <= 0;
            auxHalt <= 0;
            auxJAL <= 0;
         end
         6'111111: begin // Halt
            auxRegWrite <= 0;
            auxPCFunct <= 1;
            auxAluOp <= 3'b001;
            auxMemRead <= 0;
            auxMemWrite <= 1;
            auxMemtoReg <= 0;
            auxALUSrc <= 1;
            auxRegDst <= 0;
            auxBEQ <= 0;
            auxBNE <= 0;
            auxControlJump <= 0;
            auxHalt <= 0;
            auxJAL <= 0;
         end
   end

   assign RegDst = auxRegDst;
   assign MemRead = auxMemRead;
   assign MemtoReg = auxMemtoReg;
   assign MemWrite = auxMemWrite;
   assign ALUSrc = auxALUSrc;
   assign RegWrite = auxRegWrite;
   assign PCFunct = auxPCFunct;
   assign BEQ = auxBEQ;
   assign BNE = auxBNE;
   assign ControlJump = auxControlJump;
   assign Halt = auxHalt;
   assign EnableClock = auxEnable;
   assign JAL = auxJAL;


endmodule
```

Page 4 of 4 Revision: mips

UNIDADE DE CONTROLE ULA



```verilog
module UnidadedeControleULA (Funct, AluOp, ControleALU, JALR, JR);


   input [5:0] Funct;
   input [2:0] AluOp;
   output [3:0] ControleALU;
   output JALR, JR;

   reg[3:0] RegControle;
   reg RegJALR, RegJR;

   always @(*)
   begin
      case (AluOp)
         3'b000: begin
            case (Funct)
               6'b000001: begin // ADD
                  RegControle <= 4'b0010;
                  RegJALR <= 0;
                  RegJR <= 0;
               end
               6'b000010: begin // SUB
                  RegControle <= 4'b0011;
                  RegJALR <= 0;
                  RegJR <= 0;
               end
               6'b000011: begin // DIV
                  RegControle <= 4'b1001;
                  RegJALR <= 0;
                  RegJR <= 0;
               end
               6'b000100: begin // MULT
                  RegControle <= 4'b1000;
                  RegJALR <= 0;
                  RegJR <= 0;
               end
               6'b000101: begin // AND
                  RegControle <= 4'b0000;
                  RegJALR <= 0;
                  RegJR <= 0;
               end
               6'b000110: begin // OR
                  RegControle <= 4'b0001;
                  RegJALR <= 0;
                  RegJR <= 0;
               end
   //          6'b000111: begin // NOT
   //             RegControle = 4'b0000;
   //          end
               6'b000111: begin // NOR
                  RegControle <= 4'b1010;
                  RegJALR <= 0;
                  RegJR <= 0;
               end
               6'b001000: begin // SLL
                  RegControle <= 4'b1011;
                  RegJALR <= 0;
                  RegJR <= 0;
               end
               6'b001001: begin // SRL
                  RegControle <= 4'b1100;
                  RegJALR <= 0;
                  RegJR <= 0;
               end
               6'b001010: begin // JR
                  RegControle <= 4'b0010; // ADD ???
                  RegJALR <= 0;
                  RegJR <= 1;
               end
               6'b001011: begin // JALR
                  RegControle <= 4'b0000; // AND ???
                  RegJALR <= 1;
                  RegJR <= 0;
               end
               6'b001100: begin // SLT
                  RegControle <= 4'b0100;
                  RegJALR <= 0;
```

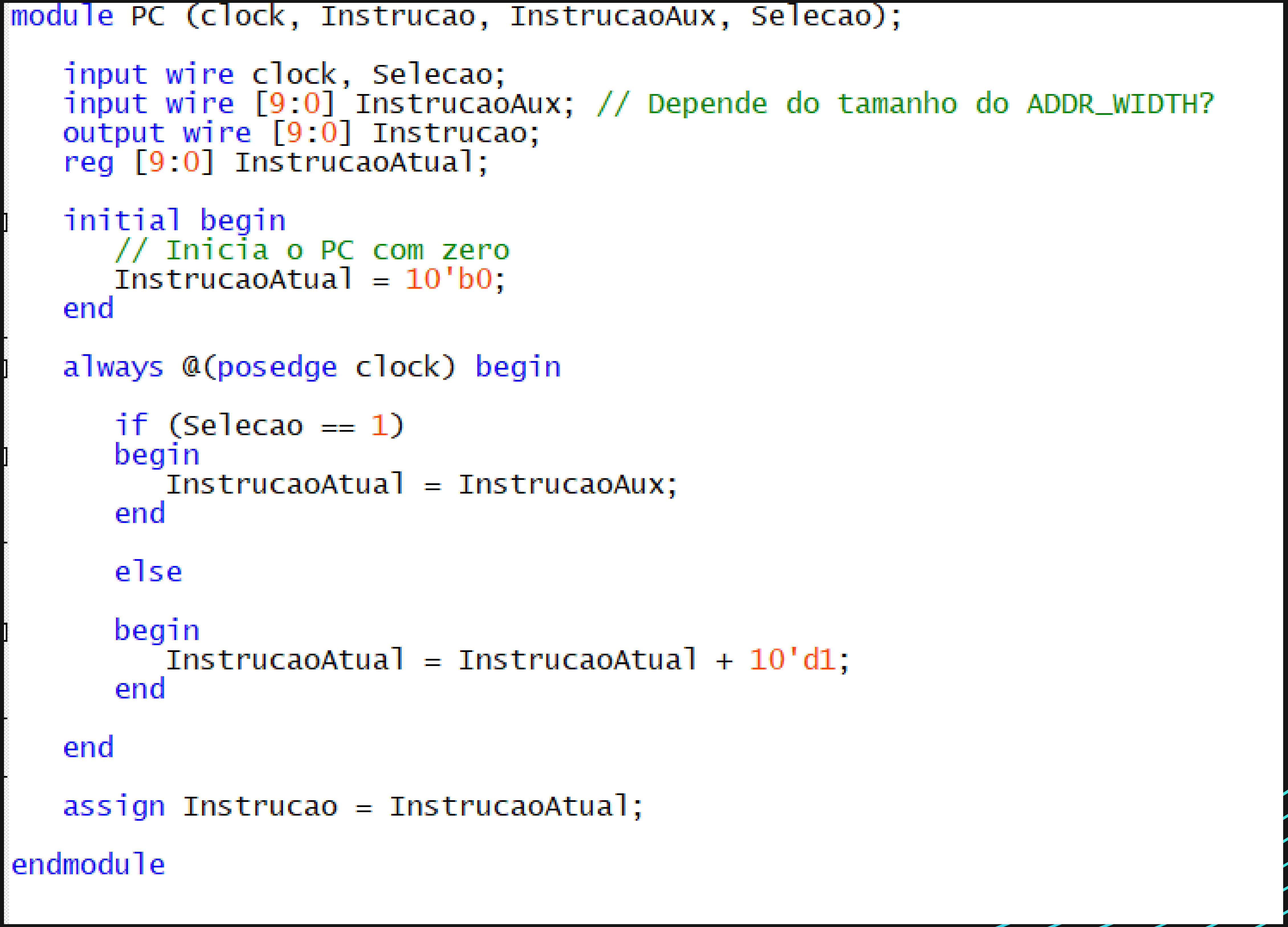


Date: June 30, 2023 UnidadedeControleULA.v Project: mips

```verilog
                  RegJR <= 0;
               end
               6'b001101: begin // SLET
                  RegControle <= 4'b0111;
                  RegJALR <= 0;
                  RegJR <= 0;
               end
               6'b001110: begin // SGT
                  RegControle <= 4'b0101;
                  RegJALR <= 0;
                  RegJR <= 0;
               end
               6'b001111: begin // SGET
                  RegControle <= 4'b0110;
                  RegJALR <= 0;
                  RegJR <= 0;
               end
            endcase
         end
         3b'001: begin // SOMA
            RegControle <= 4'b0010;
            RegJALR <= 0;
            RegJR <= 0;
         end
         3b'010: begin // SUB
            RegControle <= 4'b0011;
            RegJALR <= 0;
            RegJR <= 0;
         end
         3b'011: begin // AND
            RegControle <= 4'b0010;
            RegJALR <= 0;
            RegJR <= 0;
         end
         3b'100: begin // OR
            RegControle <= 4'b0001;
            RegJALR <= 0;
            RegJR <= 0;
         end
         3b'101: begin // SLT
            RegControle <= 4'b0100;
            RegJALR <= 0;
            RegJR <= 0;
         end
      endcase
   end

   assign ControleALU = RegControle;
   assign JALR = RegJALR;
   assign JR = RegJR;


endmodule
```

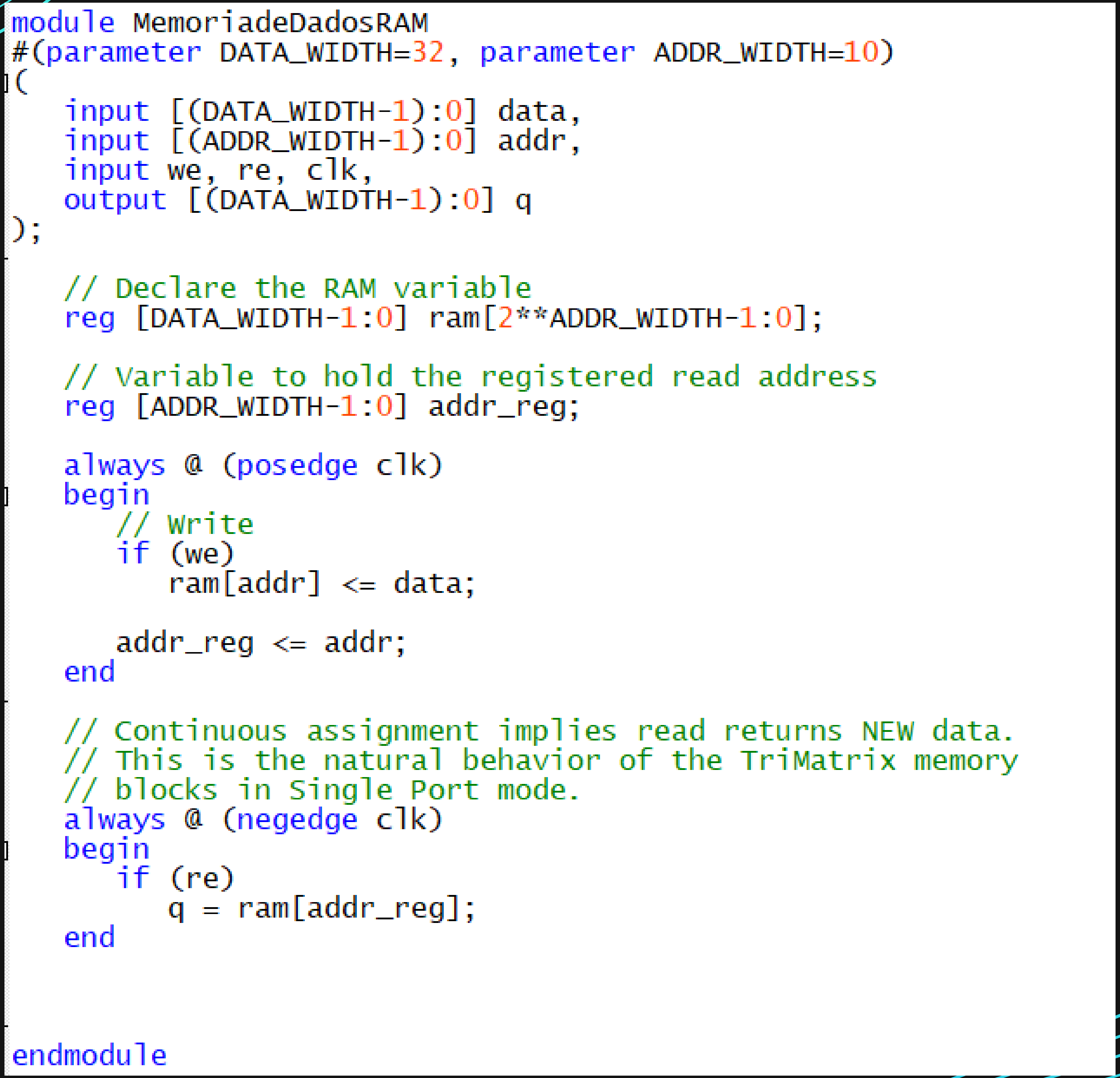
Page 2 of 2 Revision: mips

PC (PROGRAM COUNTER)



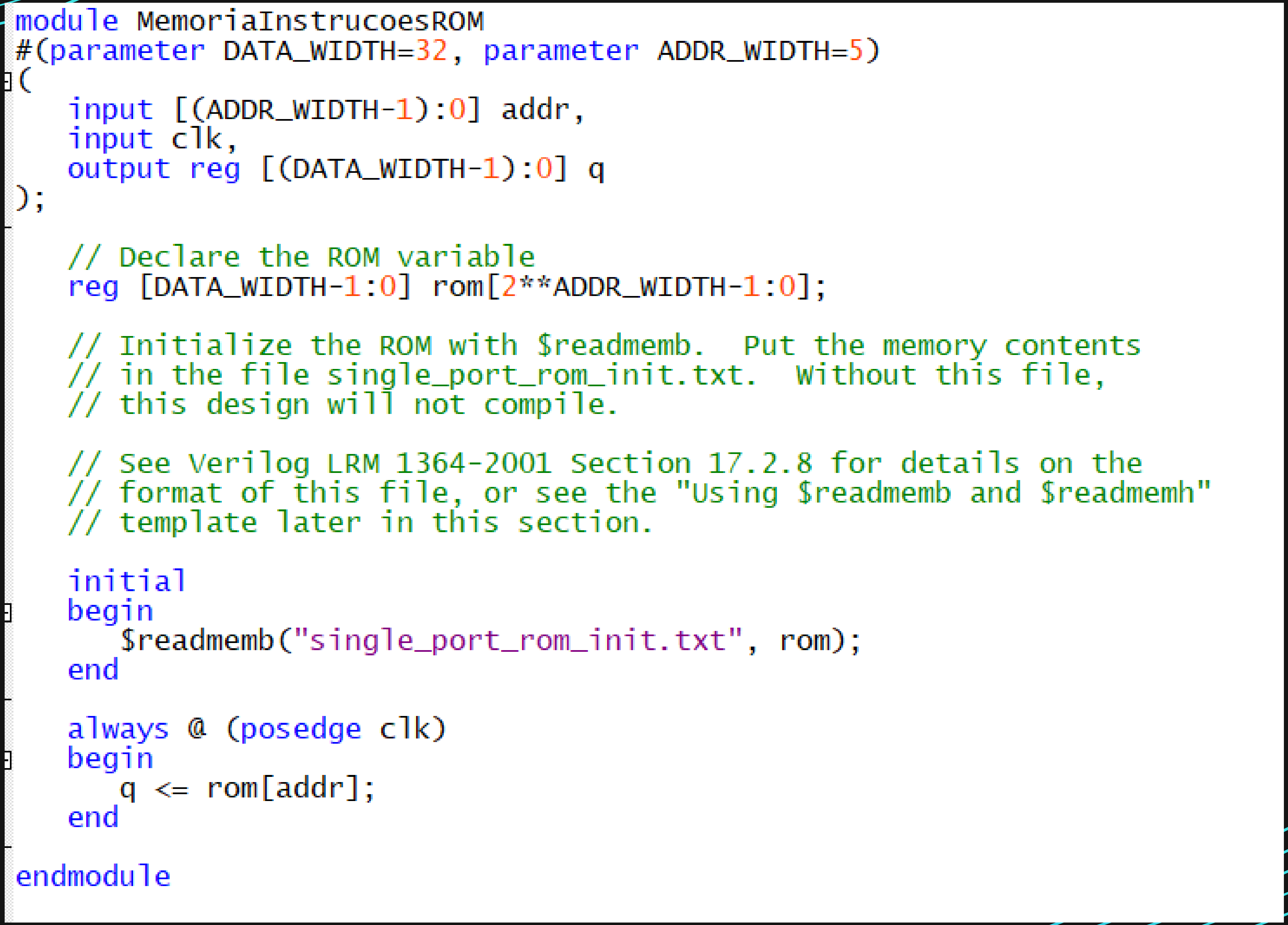
```
module PC (clock, Instrucao, InstrucaoAux, Selecao);

   input wire clock, Selecao;
   input wire [9:0] InstrucaoAux; // Depende do tamanho do ADDR_WIDTH?
   output wire [9:0] Instrucao;
   reg [9:0] InstrucaoAtual;

   initial begin
      // Inicia o PC com zero
      InstrucaoAtual = 10'b0;
   end

   always @(posedge clock) begin

      if (Selecao == 1)
      begin
         InstrucaoAtual = InstrucaoAux;
      end

      else

      begin
         InstrucaoAtual = InstrucaoAtual + 10'd1;
      end

   end

   assign Instrucao = InstrucaoAtual;

endmodule
```

MEMÓRIA DE DADOS (RAM)



```verilog
module MemoriadeDadosRAM
#(parameter DATA_WIDTH=32, parameter ADDR_WIDTH=10)
(
   input [(DATA_WIDTH-1):0] data,
   input [(ADDR_WIDTH-1):0] addr,
   input we, re, clk,
   output [(DATA_WIDTH-1):0] q
);

   // Declare the RAM variable
   reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];

   // Variable to hold the registered read address
   reg [ADDR_WIDTH-1:0] addr_reg;

   always @ (posedge clk)
   begin
      // Write
      if (we)
         ram[addr] <= data;

      addr_reg <= addr;
   end

   // Continuous assignment implies read returns NEW data.
   // This is the natural behavior of the TriMatrix memory
   // blocks in Single Port mode.
   always @ (negedge clk)
   begin
      if (re)
         q = ram[addr_reg];
   end


endmodule
```

MEMÓRIA DE INSTRUÇÕES (ROM)



```
module MemoriaInstrucoesROM
#(parameter DATA_WIDTH=32, parameter ADDR_WIDTH=5)
(
   input [(ADDR_WIDTH-1):0] addr,
   input clk,
   output reg [(DATA_WIDTH-1):0] q
);

   // Declare the ROM variable
   reg [DATA_WIDTH-1:0] rom[2**ADDR_WIDTH-1:0];

   // Initialize the ROM with $readmemb.  Put the memory contents
   // in the file single_port_rom_init.txt.  Without this file,
   // this design will not compile.

   // See Verilog LRM 1364-2001 Section 17.2.8 for details on the
   // format of this file, or see the "Using $readmemb and $readmemh"
   // template later in this section.

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

   always @ (posedge clk)
   begin
      q <= rom[addr];
   end

endmodule
```