Nome _____
Matricula _____

```
assign pc_4 = pc+4;
 assign shiftleft2 =  signalextended << 2;
 assign add_pc_branch_target = pc_4 + shiftleft2;
 assign and_branch = branch & zero;
 assign new_pc = (and_branch) ?
                        add_pc_branch_target : pc_4;
PC prog_counter(clk, res, new_pc, pc);
InstructionMem instructionmem(res, pc>>2,
instruction);
ControlUnit uc(instruction[31:26], regdst, alusrc,
memtoreg, regwrite, memread,memwrite, branch,
aluop);
assign signalextended = (instruction[15]) ?
                    {16'hFFFF,instruction[15:0]}:
                    {16'd0,instruction[15:0]}
assign muxRegDst = (regdst)?
                instruction[15:11]:instruction[20:16];
Register_Bank register_bank(res,
clk,instruction[25:21],instruction[20:16],
muxRegDst, writedata, regwrite,  data1, data2);
AluControl alucontrol(aluop,
instruction[5:0],aluctrl);
assign alu_B = (alusrc) ? signalextended:data2 ;
Alu alu(aluctrl, data1, alu_B, aluout, zero);
DataMem
datamem(res,clk,memread,memwrite,data2,
aluout>>2, readData);
assign writedata = (memtoreg) ? readData:aluout ;
```

```
--------------------ALU  CTRL-----------------------
module AluControl(input [1:0] aluop,input [5:0]
funct,output [3:0] alucontrol);
always @(aluop or funct)
 begin
   case (aluop)
     0: alucontrol <= 4'd2; // ADD para sw e lw
     1: alucontrol <= 4'd6; // SUB para branch
     default:
          begin
          case (funct)
                    32: alucontrol <= 4'd2; // ADD
                    34: alucontrol <= 4'd6; // SUB
                    36: alucontrol <= 4'd0; // AND
                    37: alucontrol <= 4'd1; // OR
                    39: alucontrol <= 4'd12; // NOR
                    42: alucontrol <= 4'd7; // SLT
          default: alucontrol <= 4'd15; // Nada
acontece


module Register_Bank(input res,input clk,input
[4:0] read1,input [4:0] read2,input [4:0] writereg,
input [31:0] writedata, input regwrite,output [31:0]
data1,output [31:0] data2);
reg [31:0] Registradores [31:0];
  assign data1 = Registradores[read1];
  assign data2 = Registradores[read2];
  always @(posedge clk or posedge res)
    if (regwrite)
    begin
      Registradores[writereg] <= writedata;
    end
module DataMem(  input res,input clk, input
MemRead,input MemWrite,input [31:0]
writeData,input [31:0] address,  output[31:0]
readData);
     reg [31:0] memory [127:0];
   always @(address)
    begin
     if (MemRead)
           readData <= memory[address];
    end
  always @(posedge clk, posedge res)
    if (MemWrite)
         memory[address] <= writeData;
```

```
module InstructionMem (
  input res,  input [31:0] address, output [31:0]
instruction_out  );

reg [31:0] instruction[31:0];

  assign instruction_out = instruction[address];
```

```
module PC(input clk, input res, input [31:0] pc_in,
output [31:0] pc);
reg [31:0] pc_reg;
 assign pc = pc_reg;
  always@(posedge clk, posedge res)
    begin
          if(res==1)
            pc_reg <= 0 ;
          else if(clk==1)
             pc_reg <= pc_in;
          end
---------   ALU ----------------------------
module Alu( input [3:0] alucontrol,  input [31:0] A,
input [31:0] B, output [31:0] aluout, output zero);

assign zero = (aluout == 0);
always @(alucontrol, A, B)
   begin
      case (alucontrol)
          0: aluout <= A & B; // AND
          1: aluout <= A | B; // OR
          2: aluout <= A + B; // ADD
          6: aluout <= A - B; // SUB
          7: aluout <= A < B ? 32'd1:32'd0; //SLT
          12: aluout <= ~(A | B); // NOR
          default: aluout <= 0; //default 0, Nada
endcase
```

---------- Control Unit ----------

```
module ControlUnit(input [5:0] opcode,output regdst, output alusrc, output memtoreg, output regwrite, output memread, output memwrite,  output branch, output [1:0] aluop);
```

```
always
@(opcode)
   begin
case(opcode)
6'd0: // R type
begin
regdst <= 1 ;
alusrc <= 0 ;
memtoreg <= 0 ;
regwrite <= 1 ;
memread <= 0 ;
memwrite<= 0 ;
branch <= 0 ;
aluop <= 2 ;


 end
6'd4: // beq
begin
 regdst <= 0 ;
alusrc <= 0 ;
memtoreg <= 0 ;
regwrite <= 0 ;
memread <= 0 ;
memwrite <= 0 ;
branch <= 1 ;
 aluop <= 1 ;


end
```

```
6'd8:
// addi
begin
regdst <= 0 ;
alusrc <= 1 ;
memtoreg <= 0 ;
regwrite <= 1 ;
memread <= 0 ;
memwrite <= 0 ;
branch <= 0 ;
aluop <= 0 ;
end
6'd35: // lw
begin
regdst <= 0 ;
alusrc <= 1 ;
memtoreg <= 1 ;
regwrite <= 1 ;
memread <= 1 ;
memwrite <= 0 ;
branch <= 0 ;
aluop <= 0 ;


 end
```

```
6'd43: // sw
begin
regdst <= 0 ;
alusrc <= 1 ;
memtoreg <= 0 ;
regwrite <= 0 ;
memread <= 0 ;
memwrite <= 1 ;
branch <= 0 ;
aluop <= 0 ;


end
default://nop
  begin
regdst <= 0 ;
alusrc <= 0 ;
memtoreg <= 0 ;
regwrite <= 0 ;
memread <= 0 ;
memwrite <= 0 ;
branch <= 0 ;
aluop <= 0 ;
end
```

Tabela com as saídas da Unidade de Controle

Sinal				
regdst				
branch				
MemRead				
MemtoReg				
ALUop				
MemWrite				
ALUSrc				
RegWrite				

1. Suponha Ri = i para os registradores. Executar 7 ciclos. Qual o valor final de R1 ? Qual valor final de PC ?
0: ADDI R1,R1,5
4: ADD R1,R1,R2
8: BEQ R1,R8,-2
12: BEQ R1,R10,2
16: ADDI R1,R1,3
20: ADD R7,R4,R1
24: ADD R1,R1,R2

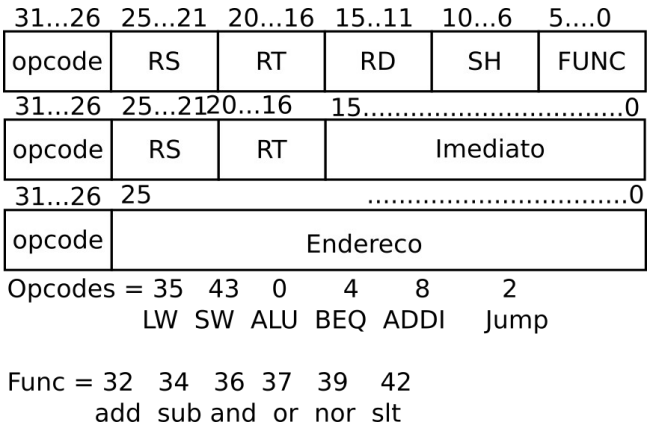
2. Codificar as instruções em hexadecimal as instruções das linhas 0, 8 e 20 da questão 1.

3. **Modifique o código Verilog** para Incluir a Instrução BLEZ. Faça anotações no desenho do Datapath. Preencha todas as linhas do datapath com a execução do BLEZ R3,-2 Preencha a tabela da unidade de controle também.

4. **Modifique o código Verilog** para Incluir a Instrução SRLV. Faça anotações no desenho do Datapath. Preencha todas as linhas do datapath com a execução do SRLV R3,R4,R5 Preencha a tabela da unidade de controle também. Note que ela é diferente da SLL ou SRL vista em aula. Atenção.

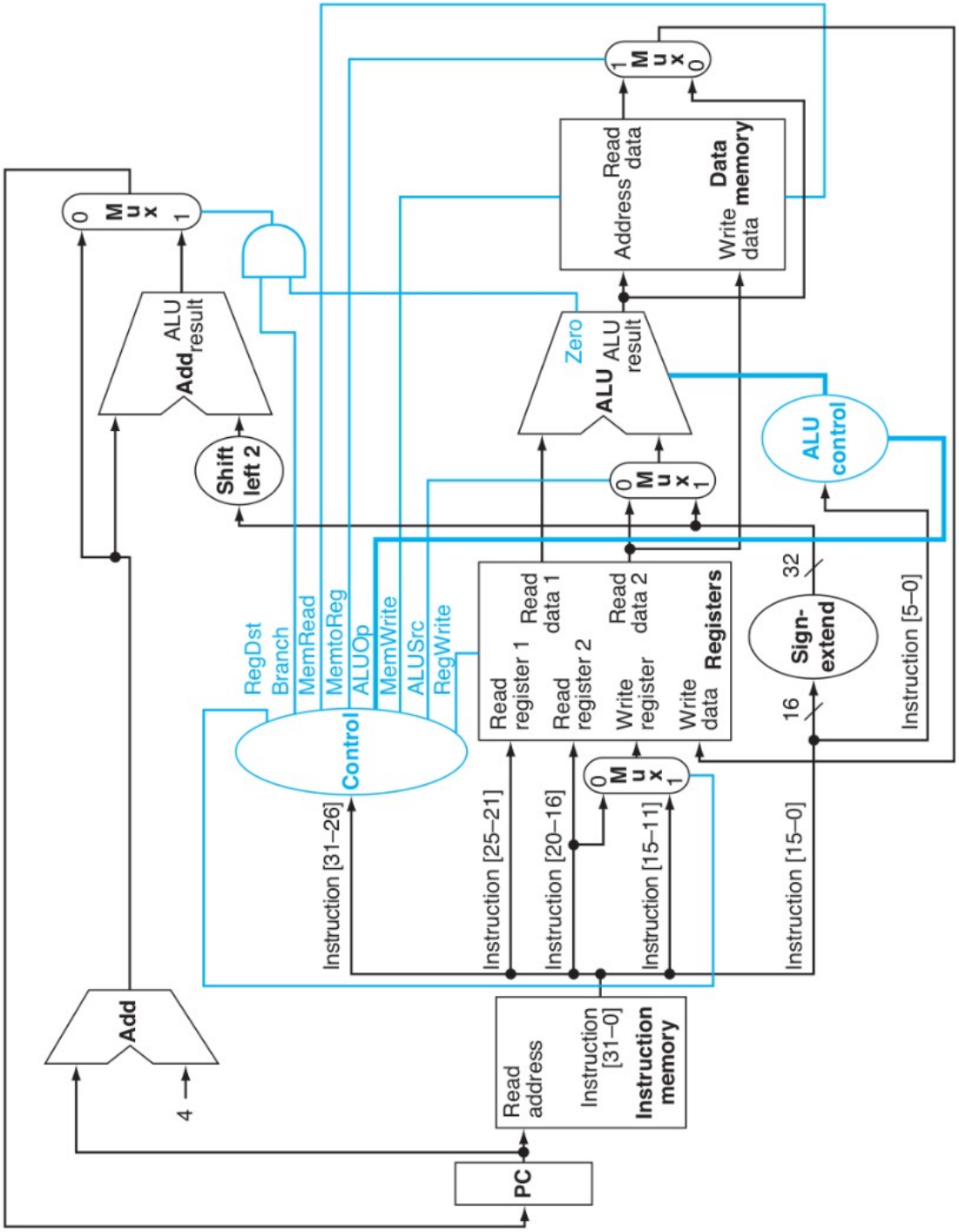
Modifique as linhas 4-16 da questão 1 para
4: SRLV R1,R1,R2
8: BLEZ R1,2
12: SUB R1,R1,R8
16: BLEZ R1,-2

Qual o valor de R1 após 10 ciclos.



BLEZ -- *Branch on less than or equal to zero*

Description:	Branches if the register is less than or equal to zero
Operation:	if $s <= 0 advance_pc (offset << 2)); else advance_pc (4);
Syntax:	blez $s, offset
Encoding:	`0001 10ss sss0 0000 iiii iiii iiii iiii`



SRL -- *Shift right logical*

Description:	Shifts a register value right by the shift amount (shamt) and places the value in the destination register. Zeroes are shifted in.
Operation:	$d = $t >> h; advance_pc (4);
Syntax:	srl $d, $t, h
Encoding:	`0000 00-- ---t tttt dddd dhhh hh00 0010`

SRLV -- *Shift right logical variable*

Description:	Shifts a register value right by the amount specified in $s and places the value in the destination register. Zeroes are shifted in.
Operation:	$d = $t >> $s; advance_pc (4);
Syntax:	srlv $d, $t, $s
Encoding:	`0000 00ss ssst tttt dddd d000 0000 0110`