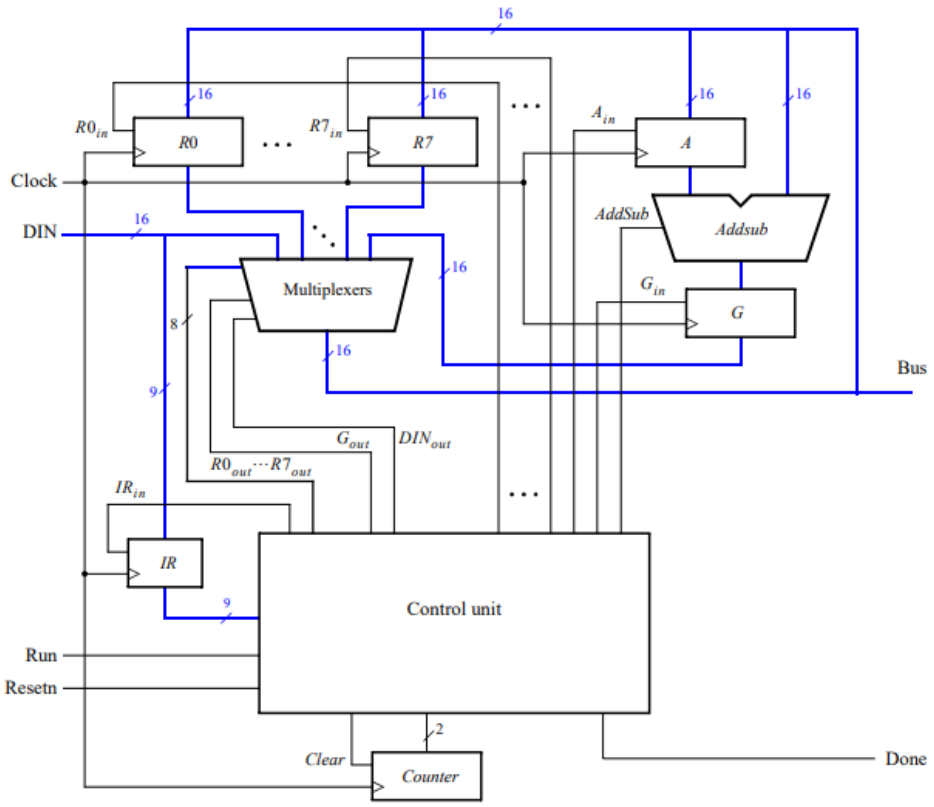
RELATÓRIO LABORATÓRIO DE ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES 2 - Prática 2

Alunos:
Julio Cesar Rocha
Iagor de Sousa

INTRODUÇÃO:

Para a segunda prática, tivemos de projetar um processador simples ligado à memória para realização de algumas instruções. O processador deveria conter 7 registradores para guardar os resultados obtidos a partir das instruções da memória (R0 a R7), além de outros registradores auxiliares para o seu multi-ciclo (registradores: A, G, IR). Foi necessário a implementação da Unidade Lógica e Aritmética (ULA) para realizar as operações das instruções, um Multiplexador de 10 entradas com seleção de 10 bits. Por fim um contador "Counter" para administrar a contagem de instruções e um módulo "IR" dedicado às instruções. O processador já estava parcialmente projetado no exercício do problema, faltando completar algumas lógicas e adicionar suporte às instruções de *"or, slt, sll e srl"*:

Operações	Função Realizada
or *Rx,Ry*	*Rx ← [Rx] | [Ry]*
slt *Rx,Ry*	*If (Rx < Ry) [Rx] = 1 else [Rx] = 0*
sll *Rx,Ry*	*Rx = [Rx] << [Ry]*
srl *Rx,Ry*	*Rx = [Rx] >>[Ry]*



DESENVOLVIMENTO:

Todos os códigos e arquivos apresentados neste relatório, se encontram na pasta do projeto.

Começamos tentando entender o que seria necessário para o funcionamento do processador, ao mesmo tempo, estávamos analisando os módulos que já estavam semi-implementados pelo próprio exercício (processor, upcount, dec3to8, regn). Com o entendimento do problema, e o que deveria ser feito, começamos à completar os Time_Step das instruções adicionais. A Tabela Time_Step indica quais variáveis seriam atribuídas aos registradores de saída, entrada e quando a ULA deveria ser acionada.

	T_1	T_2	T_3
(mv): I_0	RY_{out}, RX_{in}, *Done*		
(mvi): I_1	DIN_{out}, RX_{in}, *Done*		
(add): I_2	RX_{out}, A_{in}	RY_{out}, G_{in} ***ULA(000)***	G_{out}, RX_{in}, *Done*
(sub): I_3	RX_{out}, A_{in}	RY_{out}, G_{in}, ***ULA(001)***	G_{out}, RX_{in}, *Done*
(or): I_4	RX_{out}, A_{in}	RY_{out}, G_{in} ***ULA(010)***	G_{out}, RX_{in}, *Done*
(slt): I_5	RX_{out}, A_{in}	RY_{out}, G_{in}, ***ULA(011)***	G_{out}, RX_{in}, *Done*
(sll): I_6	RX_{out}, A_{in}	RY_{out}, G_{in} ***ULA(100)***	G_{out}, RX_{in}, *Done*
(srl): I_7	RX_{out}, A_{in}	RY_{out}, G_{in}, ***ULA(101)***	G_{out}, RX_{in}, *Done*

Com a tabela feita, partimos para a implementação restante do “processor” (Módulo do processador). Criamos variáveis referentes aos registradores para auxiliar no processo:

```
module processor(DIN, Resetn, Clock, Run, Done, BusWires);
	input [15:0] DIN;
	input Resetn, Clock, Run;
	output reg Done;
```

```
output [15:0] BusWires;


// Registradores
wire [15:0] R0, R1, R2, R3, R4, R5, R6, R7, A, G, saidaULA;
wire [8:0] IR;

// Declare variables
wire [1:0] Tstep_Q;

reg [2:0] sinal_ULA;

wire [0:7] Xreg, Yreg;

reg [0:7] regIn, regOut;
reg aIn, gIn, gOut, dinOut, Clear, irIn;

wire [2:0] I;

reg waiting;

assign I = IR[8:6];
```

A variável “Tstep_Q” se refere ao ciclo atual da instrução, variando de 0 a 4 (à depender da instrução). O registrador de 3 bits “sinal_ULA” é referente à operação que será realizada na ULA:

Bits de operação ULA “sinal_ULA”:

```
000 = add
001 = sub
010 = or
011 = slt
100 = sll
101 = srl
```

“Xreg” e “Yreg” se referem aos registradores X e Y selecionados. “regIn” e “regOut” se referem aos registradores que serão entrada (entrada do multiplexador) e saída (seleção do multiplexador). os registradores “In” são para dar enable nos registradores, os registradores “Out” estão como sinal de controle para selecionar as saídas dos registradores para o BusWires. A variável de 3 bits “I” foi criada para armazenar os 3 bits da instrução atual, sendo eles:

Bits de instrução “I”:

```
000: mv
001: mvi
010: add
011: sub
100: or
101: slt
110: sll
111: srl
```

O registrador “Clear” tem a funcionalidade de inicialização e reinicialização da contagem de “Tstep_Q” para 0, por fim, o registrador “waiting” tem como função auxiliar na instrução “mvi”, em razão da instrução “mvi” depender de 2 instruções do processador (a primeira referente à instrução MVI, e o registrador destino, a segunda se refere ao imediato).

Prosseguindo com o código “processor.v”, temos:

```
always @(Tstep_Q or I or Xreg or Yreg) begin
        if(~waiting) begin
            regIn = 8'b0;
            dinOut = 1'b0;
        end

        aIn = 1'b0;
        gIn = 1'b0;
        irIn = 1'b0;
        gOut = 1'b0;

        regOut = 8'b0;

        if(Run) begin
              .........
```

Caso o processador não esteja “aguardando” a instrução de imediato por causa do mvi, os registradores de “regIn” e “dinOut” são zerados. Sempre que haver uma mudança em uma das seguintes variáveis: “Tstep_Q”, “I”, “Xreg”, “Yreg”, resetamos os registradores e variáveis referentes ao controle de outros registradores, para que não seja necessário resetar dentro de cada caso de instrução e time step que será mostrado mais para frente . O processador só irá funcionar/seguir adiante com a instrução e time_step, caso a variável de entrada “Run” esteja acionada (Run = 1). Então caso Run esteja acionada, temos:

```
case (Tstep_Q)
                // STEP 0
                2'b00: begin
                    irIn = 1'b1;
                    Done = 1'b0;
                    Clear = 1'b0;
                end

                // STEP 1
                2'b01: begin
                    if (waiting) begin
                        Done = 1'b1;
                        waiting = 0;
                    end
                    else case(I)
                        3'b000: begin                       //Instrucao mv
                            regOut = Yreg;
                            regIn = Xreg;
                            Done = 1'b1;
                            end
                        3'b001: begin                       //Instrucao mvi
                                waiting = 1;
                                regIn = Xreg;
                                dinOut = 1'b1;
                            end
                        3'b010: begin                       //Instrucao add
                            aIn = 1'b1;
```

```
                            regOut = Xreg;
                            end
                        3'b011: begin                           //Instrucao sub
                            aIn = 1'b1;
                            regOut = Xreg;
                            end
                        3'b100: begin                           //Instrucao or
                            aIn = 1'b1;
                            regOut = Xreg;
                            end
                        3'b101: begin                           //Instrucao slt
                            aIn = 1'b1;
                            regOut = Xreg;
                            end
                        3'b110: begin                           //Instrucao sll
                            aIn = 1'b1;
                            regOut = Xreg;
                            end
                        3'b111: begin                           //Instrucao srl
                            aIn = 1'b1;
                            regOut = Xreg;
                            end
                    endcase
                end

                // STEP 2
                2'b10: if(I > 3'b001) begin
                        gIn = 1'b1;
                        regOut = Yreg;

                        case(I)
                            3'b010: sinal_ULA = 3'b000;
                            3'b011: sinal_ULA = 3'b001;
                            3'b100: sinal_ULA = 3'b010;
                            3'b101: sinal_ULA = 3'b011;
                            3'b110: sinal_ULA = 3'b100;
                            3'b111: sinal_ULA = 3'b101;
                    endcase
                end

               // STEP 3
                2'b11: begin if(I > 3'b001) begin
                        gOut = 1'b1;
                        regIn = Xreg;
                        Done = 1'b1;
                    end
                    Clear = 1'b1;
                end
            endcase
        end
      end

      ULA moduloULA(sinal_ULA, A, BusWires, saidaULA);

      multiplex mult(DIN, R0, R1, R2, R3, R4, R5, R6, R7, G, BusWires, {dinOut,
regOut, gOut});
```

Para o *Step 0,* temos o início de uma nova instrução, para isso, zeramos a variável “Done” que indica se uma instrução foi completa (1) ou não (0), zeramos o “Clear” que indica se devemos resetar o time step (1) ou não (0), e por fim settamos “irIn” = 1, para indicar que uma nova instrução acaba de chegar ao processador.

Para o *Step 1,* começamos avaliando o registrador “waiting”, ele indica se estamos “aguardando” mais uma instrução (referente ao imediato) por causa da instrução “mvi”, se estamos “aguardando” o imediato, então, quando o imediato for enviado, o registrador “waiting” estará como

“True”, podendo então entrar na condição e concluir a instrução de “mvi” resetando “waiting” e settando “Done” = 1. Caso “waiting” seja “False”, prosseguimos para analisar caso a caso a instrução, nesta parte, tudo que fazemos é a atribuição de valores aos registradores de entrada, saída, e os sinais de controle dos registradores (aIn, gIn, gOut, dinOut), e caso uma instrução seja completada, já no Step 1, settamos “Done” = 1.

Para o *Step 2*, só iremos settar o bit de “gIn” e “regOut” caso o bit “I” de instrução seja maior que “001”, ou seja, somente se a instrução atual for maior que a instrução “mvi”, pois as duas primeiras instruções de “mv” (000) e “mvi” (001) são completadas já no Step 1. Então se o bit “I” de instrução for maior que “001” realizamos a atribuição necessária como apresentada na tabela “Tabela Time_Step_Q” e após isso seguimos para analisarmos caso a caso, qual será a operação realizada pela ULA.

Por fim, em *Step 3*, fazemos a mesma análise feita no Step 2, caso o bit “I” de instrução seja maior que “001” é feito as atribuições indicadas na tabela “Tabela Time_Step_Q”, após passar por todos os “steps”, atribuímos ao registrador “Clear” o valor 1 para no próximo ciclo resetar os “steps”.

Após passar por todos os Steps, mandamos os dados para o módulo “ULA” onde são realizadas as operações:

```
module ULA(sinal_ULA, A,  BusWires, G);
	input [2:0] sinal_ULA;
	input [15:0] A, BusWires;
	output reg [15:0] G;

	/*
	000 = add
	001 = sub
	010 = or
	011 = slt
	100 = sll
	101 = srl
	*/

	always@(*)begin
		case(sinal_ULA)
		3'b000: G = A + BusWires;
		3'b001: G = A - BusWires;
		3'b010: G = A | BusWires;
		3'b011: begin
					if(A < BusWires)
							G = 16'b1;
					else
							G = 16'b0;
				end
		3'b100: G = A << BusWires;
		3'b101: G = A >> BusWires;
			endcase
	end

endmodule
```

Depois de realizar os cálculos necessários na ULA, prosseguimos para o multiplexador, onde é selecionado o sinal de controle e então os dados são transferidos para “BusWires”:

```
module multiplex(In1, In2, In3, In4, In5, In6, In7, In8, In9, In10, Out,
Control);
	input [9:0] Control;
	input [15:0] In1, In2, In3, In4, In5, In6, In7, In8, In9, In10;
```

```
	output reg [15:0] Out;

	always@(*) begin
		case(Control)
			10'b1000000000: Out = In1;
			10'b0100000000: Out = In2;
			10'b0010000000: Out = In3;
			10'b0001000000: Out = In4;
			10'b0000100000: Out = In5;
			10'b0000010000: Out = In6;
			10'b0000001000: Out = In7;
			10'b0000000100: Out = In8;
			10'b0000000010: Out = In9;
			10'b0000000001: Out = In10;
		endcase
	end

endmodule
```

Após todos esses processos, temos a atribuição dos registradores e o fim do módulo "processor":

```
// Aribuicao dos registradores
	regn reg_0(BusWires, regIn[0], Clock, R0, Resetn);
	regn reg_1(BusWires, regIn[1], Clock, R1, Resetn);
	regn reg_2(BusWires, regIn[2], Clock, R2, Resetn);
	regn reg_3(BusWires, regIn[3], Clock, R3, Resetn);
	regn reg_4(BusWires, regIn[4], Clock, R4, Resetn);
	regn reg_5(BusWires, regIn[5], Clock, R5, Resetn);
	regn reg_6(BusWires, regIn[6], Clock, R6, Resetn);
	regn reg_7(BusWires, regIn[7], Clock, R7, Resetn);
	regn reg_A(BusWires, aIn, Clock, A, Resetn);
	regn reg_G(saidaULA, gIn, Clock, G, Resetn);
	regn_9 reg_IR(DIN[15:7], irIn, Clock, IR, Resetn);
endmodule
```

Um segundo módulo similar ao módulo "regn" que já estava implementado no PDF disponibilizado pela professora, foi criado, dessa vez, chamado de "**regn_9**", ele tem a função de resetar o registrador passado como parâmetro (registrador de instrução de 9 bits) caso Resetn = 1, ou atribuir à esse registrador, a próxima instrução.

Com o módulo de processador ("processor.v") pronto, criamos a memória ROM-1-PORT, inicializando com o arquivo "memory_init.mif", em que, para cada endereço havia uma instrução do caso de teste disponibilizado pela professora:

```
DEPTH = 32;                   -- The size of memory in words
WIDTH = 16;                   -- The size of data in bits
ADDRESS_RADIX = HEX;          -- The radix for address values
DATA_RADIX = BIN;             -- The radix for data values
CONTENT                       -- start of (address : data pairs)
BEGIN

00 : 0010000000000000;  -- MVI R0, #2
01 : 0000000000000010;
02 : 0010010000000000;  -- MVI R1, #3
03 : 0000000000000011;
04 : 0100010000000000;  -- ADD R1, R0
05 : 0010100000000000;  -- MVI R2, #6
```

```
06 : 0000000000000110;
07 : 0110100010000000;  -- SUB R2, R1
08 : 0000110100000000;  -- MV R3, R2
09 : 0100000110000000;  -- ADD R0, R3
0A : 1000010000000000;  -- OR R1, R0
0B : 0110010000000000;  -- SUB R1, R0
0C : 0100010110000000;  -- ADD R1, R3
0D : 1100010110000000;  -- SLL R1, R3
0E : 1110010110000000;  -- SRL, R1, R3
0F : 0010000000000000;  -- MVI R0, #0
10 : 0000000000000000;
11 : 1010000010000000;  -- SLT R0, R1
12 : 1010010010000000;  -- SLT R1, R1
13 : 0100000110000000;  -- ADD R0, R3
14 : 0100010100000000;  -- ADD R1, R2
15 : 0000000000000000;
16 : 0000000000000000;

END;
```

Por fim, instanciamos a memória, o processador e um módulo contador em um outro módulo chamado “pratica_2”:

```
module pratica_2(MClock, PClock, Resetn, Run, Bus, Done);
	input MClock, PClock, Resetn, Run;
	output [15:0]Bus;
	output Done;

	wire [4:0] n;
	wire [15:0] memoryOut;

	upcount_5 count(Resetn, MClock, n);
	memory mem(n, MClock, memoryOut);
	processor proc(memoryOut, Resetn, PClock, Run, Done, Bus);
endmodule
```

Um segundo módulo similar ao que já estava implementado no problema “upcount”, foi criado, dessa vez chamado de “upcount_5”. No módulo “upcount_5” ao invés de termos uma variável “Q” de 2 bits, temos uma variável de “Q” de 5 bits que funciona como o endereço atual da memória, variando de 0 a 32.

SIMULAÇÃO:

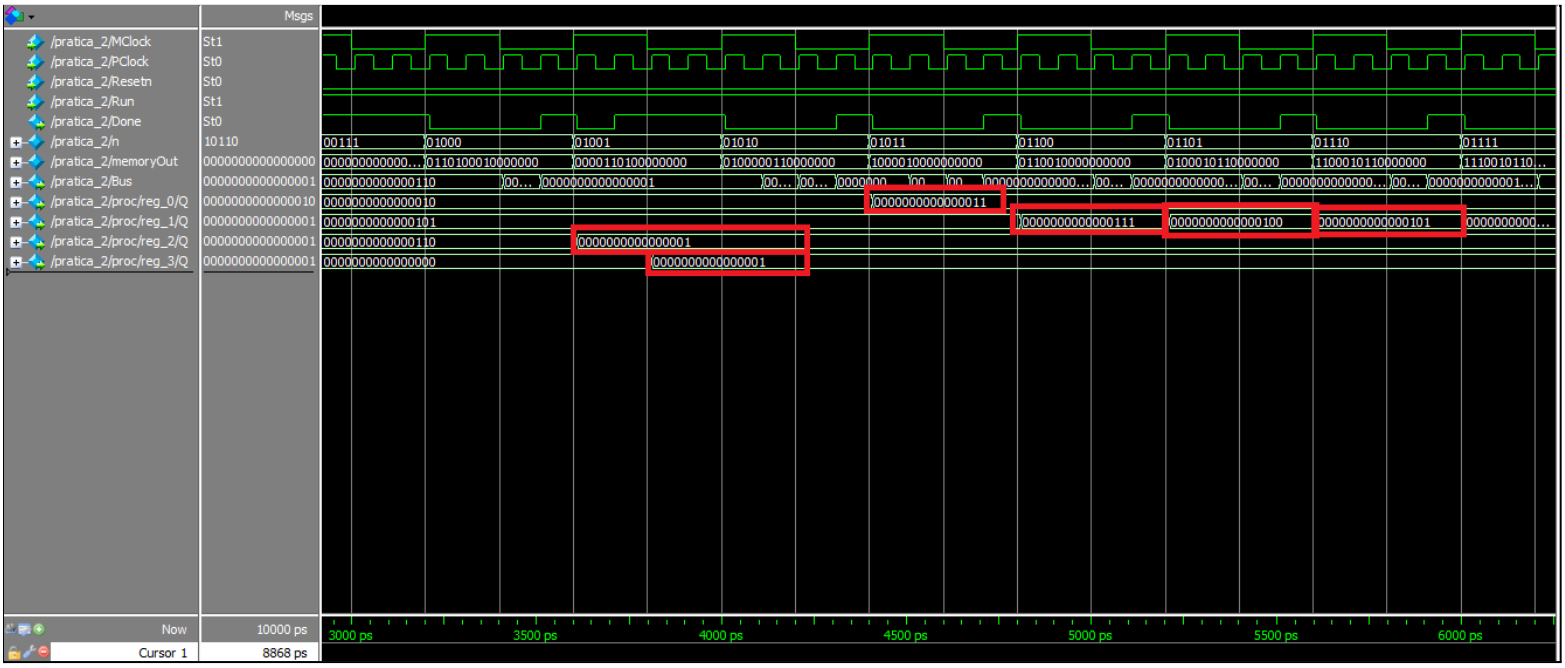
Para simulação, foi gerado um script com os comandos necessários para se colocar no terminal para criar as formas de ondas:

```
force -freeze sim:/pratica_2/Resetn 1 0, 0 {50 ps} -r 10000
force -freeze sim:/pratica_2/Run 1 0
```

```
force -freeze sim:/pratica_2/MClock 1 0, 0 {200 ps} -r 400
force -freeze sim:/pratica_2/PClock 1 10, 0 {60 ps} -r 100
add wave -position end  sim:/pratica_2/MClock
add wave -position end  sim:/pratica_2/PClock
add wave -position end  sim:/pratica_2/Resetn
add wave -position end  sim:/pratica_2/Run
add wave -position end  sim:/pratica_2/Done
add wave -position end  sim:/pratica_2/n
add wave -position end  sim:/pratica_2/memoryOut
add wave -position end  sim:/pratica_2/Bus
add wave -position end  sim:/pratica_2/proc/reg_0/Q
add wave -position end  sim:/pratica_2/proc/reg_1/Q
add wave -position end  sim:/pratica_2/proc/reg_2/Q
add wave -position end  sim:/pratica_2/proc/reg_3/Q
```

O clock utilizado para a memória (MClock) foi de 4x maior que o clock utilizado para o processador (PClock), por razão dos ciclos necessários em uma instrução, sendo: MClock: 400ps e PClock: 100ps. Utilizamos o “Resetn” para inicializar o processador e seus registradores com 0, ficando ativo apenas no início da simulação por 50ps. O “Run” foi utilizado para dar prosseguimento às instruções, já que o processador só funcionaria caso o “Run” esteja ativo (como descrito no detalhamento do código da prática 2 em inglês - parte 1). Com isso temos as seguintes formas de ondas:

A 1a instrução faz com que o imediato de valor 2 seja carregado no registrador 0 (mvi).
A 2a instrução faz com que o imediato de valor 3 seja carregado no registrador 1 (mvi).
A 3a instrução faz a soma do valor de R0 com R1, salvando seu resultado em R1 (10 + 11 = 101) (add).
A 4a instrução faz com que o imediato de valor 6 seja carregado no registrador 2 (mvi).



A 5a instrução faz a subtração do valor de R2 com R1, salvando seu resultado em R2 (110 - 101 = 001) (sub).
A 6a instrução faz com que o valor de R2 seja passado para R3 (mv).
A 7a instrução faz a soma do valor de R0 com R3, salvando seu resultado em R0 (10 + 01 = 11) (add).
A 8a instrução faz a operação “or” entre o registrador R1 e R0, salvando seu resultado em R1 (or).
A 9a instrução faz a subtração do valor de R1 com R0, salvando seu resultado em R1 (111 - 011 = 100) (sub).
A 10a instrução faz a soma do valor de R1 com R3, salvando seu resultado em R1 (100 + 001 = 101) (add).

A 11a instrução faz o shift lógico à esquerda, sendo R1 << R3 , salvando seu resultado em R1 (sll).
A 12a instrução faz o shift lógico à direita, sendo R1 >> R3 , salvando seu resultado em R1 (srl).
A 13a instrução faz com que o imediato de valor 0 seja carregado no registrador 0 (mvi).
A 14a instrução faz a comparação entre R0 e R1 (R0 < R1?), salvando seu resultado em R0 (slt).
A 15a instrução faz a comparação entre R1 e R1 (R1 < R1?), salvando seu resultado em R1 (slt).
A 16a instrução faz a soma do valor de R0 com R3, salvando seu resultado em R3 (1 + 1 = 10) (add).
e por fim, a 17a instrução faz a soma do valor de R1 com R2, salvando seu resultado em R1 (0 + 1 = 1) (add).

Os valores finais dos registradores então ficam:
R0 = 2
R1 = 1
R2 = 1
R3 = 1

Caso queira ver as imagens em uma escala maior e em qualidade melhor ,as imagens se encontram na pasta junto ao projeto.